Notes on CO-450: Combinatorial Optimization

University of Waterloo

DAVID DUAN

Last Updated: December 15, 2020 (Draft)

Contents

Part I

Minimum Spanning Trees

1 Minimum Spanning Trees

We wish to develop algorithms to find minimum spanning trees. Our plan is as follows.

1. Characterize spanning trees and minimum spanning trees.
2. Use such characterizations to derive algorithms (and prove their correctness).
3. Alternatively, prove their correctness using linear programming techniques.

1.1 Spanning Trees.

Definition 1.1. Given graph $G = (V, E)$, a subgraph T is a **spanning tree** of G if $V(T) = V(G)$ and T is a tree, i.e., it is connected and acyclic.

Theorem 1.2. *Let $G = (V, E)$ be connected and T a subgraph of G with $V(T) = V$. The following are equivalent (henceforth, TFAE):* [a]

1. *T is a spanning tree of G.*
2. *T is **minimally connected**, i.e., removing any edge from T disconnects T.*
3. *T is **maximally acyclic**, i.e., adding any edge to T creates a cycle in T.*
4. *T is connected and has $|V| - 1$ edges.*
5. *For all $u, v \in V$, there exists a unique u, v-path in T, denoted $T_{u,v}$.*

[a] Recall an undirected graph is a tree iff there exists exactly one simple path between each pair of vertices. Compare this with the last statement.

Definition 1.3. For $G = (V, E)$ and $A \subseteq V$, the **cut** of G induced by A is defined as $\delta(A) := \{e \in E : e \text{ has one end in } A \text{ and the other end in } V \setminus A\}$.

Theorem 1.4. *A graph $G = (V, E)$ is connected iff for all $A \subseteq V$ with $\emptyset \neq A \neq V$, we have $\delta(A) \neq \emptyset$. In words, a graph is connected iff for any proper subset of the vertex set, there is at least one edge leaving/entering it.*

Proof. It is easy to see that, if $\delta(A) = \emptyset$ with $u \in A$ and $v \notin A$, then there is no u, v-path in G and hence, if $\emptyset \neq A \neq V$, G is disconnected. Now suppose G is not connected. Choose $u, v \in V$ such that is no u, v-path in G. Define $A := \{w \in V : \text{there exists a } u, w\text{-path in } G\}$.[1] Then $u \in A$ and $v \notin A$, so $\emptyset \neq A \neq V$. We claim that $\delta(A) = \emptyset$. Suppose not, then there exists $pq \in \delta(A) \subseteq E$ with $p \in A$ and $q \notin A$. Then adding e, q to any path from u to p gives a path from u to q, contradicting the fact that $q \notin A$. □

[1] A is the connected component that contains u. Thus, there cannot be anything entering or leaving A.

1.2 *Minimum Spanning Trees.*

The **minimum spanning tree** (henceforth, **MST**) problem is given as follows. Unless otherwise noted, we assume the graph G is connected throughout this chapter.

Problem (MST). *Given a connected graph $G = (V, E)$ and costs $c : E \to \mathbb{R}$,*[a] *return a spanning tree T of G of minimum cost, where the cost of T is given by*

$$c(T) = \sum_{e \in E(T)} c_e.$$

[a] Alternatively, $c \in \mathbb{R}^E$ or $c \in \mathbb{Z}^E$. The cost for edge $e \in E$ is denoted c_e or $c(e)$.

The following lemma is used in the proof of Theorem 1.6.

Lemma 1.5. *If T is a spanning tree of G and e' is not an edge in T, then $T + e'$ contains exactly one cycle C. Moreover, if e is on the cycle C, then $T - e + e'$ is also a spanning tree of G.*

Proof. Since T is maximally acyclic, adding e' is guaranteed to form a cycle C. Now let T_1, T_2 be the two components of $T - e$. Suppose $e' = \{u, v\}$ where $u \in V(T_1)$ and $v \in V(T_2)$. Let $x \in V(T_1)$ and consider some $y \in V(T_1) \cup V(T_2)$. If $y \in V(T_1)$, there exists a x, y-path since T_1 is connected. If $y \in V(T_2)$, since T_1 and T_2 are each connected, there exists an x, u-path P_1 and a v, y-path P_2. Then $P_1 + uv (= e') + P_2$ forms an x, y-path. In either case, $T - e + e'$ is connected. Since it also has $n - 1$ edges, it is a spanning tree of G. □

Theorem 1.6. *Let $G = (V, E)$, $c : E \to \mathbb{R}$, and T a spanning tree of G. TFAE:*

1. *T is a MST.*
2. *Let $uv \in E \setminus E(T)$ be arbitrary. Then for any edge e on $T_{u,v}$, we have $c_{uv} \geq c_e$.*
3. *Let $e \in E(T)$. If T_1, T_2 are the two connected components in $T - e$, then e is a minimum cost edge in $\delta(T_1) = \delta(T_2)$.*

Let's first look at what statement (2) and (3) really mean.

$\neg 2 \Rightarrow \neg 1$: Consider the left figure below, where the subgraph T in orange is a spanning tree of the graph. Since $c(v_3 v) = 3 > 2 = c(uv)$, we could remove $v_3 v$ and add uv to obtain another spanning tree but with less cost.

$\neg 3 \Rightarrow \neg 1$: Consider the right figure below, where the subgraph T is orange is an MST of the graph and T_1, T_2 are the components induced by removing e from T. The purple edges are those in $\delta(T_1) = \delta(T_2)$. If any of them has a smaller cost than e, then we could remove e and add that edge, which yields another spanning tree but with less cost.

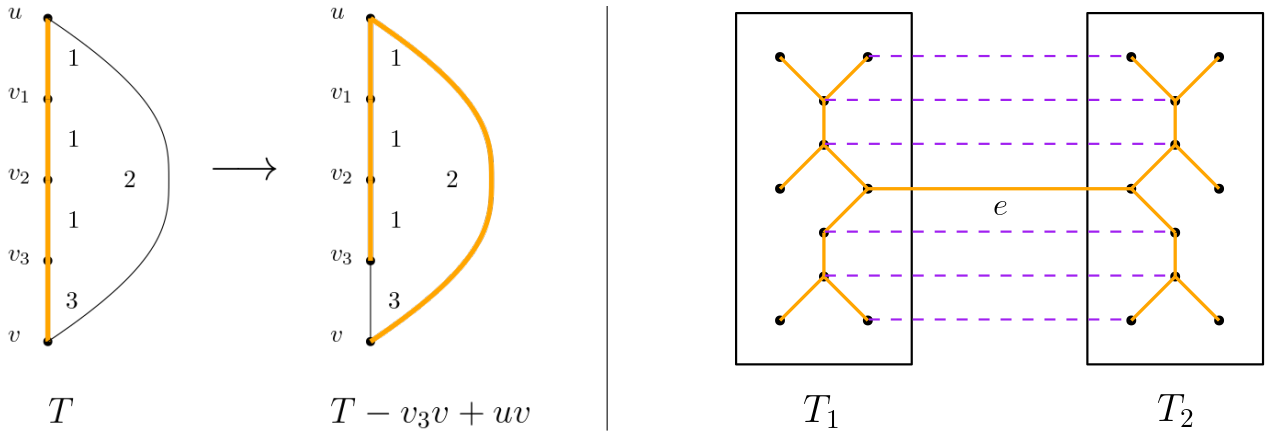


Figure 1.1: Left: ¬2 ⇒ ¬1. Right: ¬3 ⇒ ¬1.

Proof.

¬2 $\implies$ ¬1: Suppose there exist $uv \in E \setminus E(T)$ and $e \in T_{uv}$ such that $c_e > c_{uv}$. By Lemma 1.5, $T' := T + uv - e$ is a spanning tree of G with less cost than T, so T is not a MST.

¬3 $\implies$ ¬2: Let $e \in E(T)$ and T_1, T_2 be two connected components of $T - e$. Suppose e is not the minimum cost edge in $\delta_G(T_1)$, i.e., there exist some edge uv where $u \in T_1$, $v \in T_2$, and that $c_e > c_{uv}$. Since u and v are in different components of $T - e$, $uv \notin E(T)$.[2] Also, since u and v are disconnected by removing e, edge e must be on the path $T_{u,v}$. We have found the desired edges $e \in T_{u,v}$ and $uv \notin E(T)$ with $c_e > c_{uv}$, the negation of Statement 2.

3 $\implies$ 1: Let T satisfy (3) and T^* be a MST with largest $k := |E(T) \cap E(T^*)|$ [3]. If $k = n - 1$, $T = T^*$ and we are done. Otherwise, there exists some $e \in E(T) \setminus E(T^*)$. Let T_1, T_2 be the connected components of $T - e$. Since T^* is connected, there exists $e^* \in E(T^*) \cap \delta(T_1)$.[4] Since $e \notin E(T^*)$, $e^* \neq e$. By (3), e is the minimum cost edge in $\delta(T_1)$, i.e., $c_e \leq c_{e^*}$. Then $T' := T^* - e^* + e$ is also a spanning tree (Lemma 1.5) and $c(T') = c(T^*) - c_{e^*} + c_e \leq c(T^*)$. Thus, $c(T')$ is also a MST. But then $|E(T) \cap E(T')| > |E(T) \cap E(T^*)|$,[5] contradiction. □

[2] Otherwise, we have a cycle in T which contains e and uv.
[3] Variable k here denotes the number of edges in T^* coinciding with T.
[4] By Theorem 1.4, if $\delta_{T^*}(T_1) := E(T^*) \cap \delta(T_1) = \emptyset$, then T^* is disconnected. Contradiction.
[5] T' is a MST containing one more edge from $E(T)$ (namely e) than T^*.

1.3 Kruskal's Algorithm for Minimum Spanning Trees.

Algorithm 1: Kruskal's Algorithm

input: $G = (V, E)$ connected, costs $c : E \to \mathbb{R}$
output: H, a minimum spanning tree of G
main:
1: Initialize $H = (V, \emptyset)$, i.e., a forest of isolated vertices
2: **while** H is not a spanning tree of G **do**
3: e = cheapest edge whose endpoints are in different connected components of H
4: Add e to H
5: **return** H

Lemma 1.7. *The subgraph H returned by the algorithm is a spanning tree.*

Proof. First, note that e at step 3 always exists, as otherwise H is connected and $V(H) = V$ so H is a spanning tree. Now every time step 4 gets executed, the number of connected components of H reduces by 1, and H is acyclic. Thus, the while loop terminates in $O(n)$ iterations with a connected, acyclic graph, i.e., a spanning tree. □

Proposition 1.8. *The subgraph H returned by the algorithm is an MST.*

Proof. Suppose the returned H is not a MST. Then there exists some $uv \in E \setminus E(H)$ and $e \in H_{u,v}$ with $c_{u,v} < c_e$. When $e_i = uv$ is tested at line 4 of the implementation, $H_{u,v}$ still does not exist. Then uv would have been added to H. □

It is easy to show that Kruskal's runs in polynomial time of $n = |V|$ and $m = |E|$. [6]

Algorithm 2: Kruskal's Algorithm, implementation

input: $G = (V, E)$ connected, costs $c : E \to \mathbb{R}$
output: H, a minimum spanning tree of G
main:
1: Initialize $H = (V, \emptyset)$ ▷ $O(1)$
2: Sort the edges so that $c_1 \leq \cdots \leq c_m$. ▷ $O(m \log m)$
3: **for** $i = 1, \ldots, m$ **do** ▷ $O(m)$ iterations
4: **if** endpoints u, v of e_i are in different connected components of H **then** ▷ $O(n)$
5: Add e_i to H
6: **return** H ▷ Overall: $O(m \log m) + O(mn) \subseteq O(mn)$

[6] A nice visualization of Kruskal's algorithm on wikipedia.

1.4 Proof of Correctness with Linear Programming.

We want to do a second proof of correctness using linear programming because:

1. It shows a nice techniques that can be used in other settings.
2. These techniques can lead to "good" approaches for more challenging problems.

An Integer Program Formulation

Let $x_e \in \{0,1\}$ be a variable that indicates whether edge e is in the MST. Recall MSTs are acyclic and have $n-1$ edges. Thus, we have

- Objective function: min $\sum_{e\in E} c_e x_e (= c^T x)$.
- Constraint 1, we use exactly $n-1$ edges from E: $x(E) = \sum_{e\in E} x_e = n-1$.
- Constraint 2, the graph needs to be acyclic: $x(F) \leq n - \kappa(F)$ for all $F \subseteq E$(see below).
- Constraint 3, indicator variables: $x \in \{0,1\}^E$.

Acyclic Constraint

Consider $F \subseteq E$. How many edges from F can a spanning tree use?

Claim. *Let $\kappa(F)$ be the number of connected components of (V,F). Then a spanning tree can contain at most $n-\kappa(F)$ edges of F.*

Proof. For any component H of (V,F), we can use at most $n_H - 1 = |V(H)| - 1$ edges without creating a cycle. Suppose there are $\kappa(F)$ components, $H_1, \ldots, H_{\kappa(F)}$. Then we can use at most $n_{H_1} - 1 + \cdots + n_{H_{\kappa(F)}} - 1 = (n_{H_1} + \cdots + n_{H_{\kappa(F)}}) - \kappa(F) = n - \kappa(F)$ edges. □

Remark. Consider $F = \{e\}$. Then $\kappa(F) = n-1$ and $x(F) \leq n - \kappa(F)$ becomes $x_e \leq 1$. Thus, we have an implicit upper bound constraint that $x_e \leq 1$ for all $e \in E$. As a result, we can modify Constraint 3 to $x \geq 0, x \in \mathbb{Z}^E$ as the $x \leq 1$ condition is implied by Constraint 2.

To summarize, this is the integer program we get for the MST problem:

$$\begin{array}{lll} \min & c^T x & \\ s.t. & x(E) = n-1 & \\ & x(F) \leq n - \kappa(F) & \forall F \subseteq E \\ & x \geq 0, x \in \mathbb{Z}^E & \end{array}$$

Note that any spanning tree T corresponds to a feasible solution to this IP. We will next consider the LP relaxation of this IP and show that the spanning tree produced by Kruskal's is optimal for the primal-dual pair using complementary slackness conditions. This proves that Kruskal's algorithm produces MSTs.

The LP Relaxation and Its Dual

The LP relaxation of the IP above is given by

$$\begin{array}{lll} (\mathrm{P_{ST}}): \min & c^T x & \\ s.t. & x(E) = n-1 & (y_E) \\ & x(F) \le n - \kappa(F) \quad \forall F \subset E & (y_F) \\ & x \ge 0 & \end{array}$$

Note we made two changes:

1. We dropped the integer constraint $x \in \mathbb{Z}^E$.
2. We dropped the case of $F = E$ in Constraint 2 as it is implied by Constraint 1.

The dual of $(\mathrm{P_{ST}})$ is $(\mathrm{D_{ST}})$ given by

$$\begin{array}{lll} (\mathrm{D_{ST}}): \max & \sum_{F \subseteq E} [(n - \kappa(F)) y_F] & \\ s.t. & \sum_{F: e \in F} y_F \le c_e & \forall e \in E \\ & y_F \le 0 & \forall F \subset E \end{array}$$

Since the input is connected and the feasibility region is bounded, the Fundamental Theorem of LPs tells us that there is always an optimal solution. Our goal is to show that the solution returned by Kruskal's algorithm is optimal for $\mathrm{P_{ST}}$ and $\mathrm{D_{ST}}$.

Proof of Correctness using LP

Ordered the set of edges so that $c_{e_1} \le \cdots \le c_{e_m}$. Consider $E_i = \{e_1, \ldots, e_i\}$, which are the edges considered by Kruskal in the first i iterations of the for loop. Define the dual variables for each $F \subseteq E$ as follows:

- $\bar{y}_{E_i} = c_{e_i} - c_{e_{i+1}} \le 0$, for all $i = 1, \ldots, m-1$.
- $\bar{y}_{E_m} = y_E = c_{e_m}$.
- $\bar{y}_F = 0$ for all other F.

Lemma 1.9. *$\bar{y}$ is feasible for $\mathrm{D_{ST}}$ and satisfies all dual constraints (except the sign ones) at equality, that is, $\sum_{F: e_i \in F} \bar{y}_F = c_{e_i}$ for $i \in [m]$.*

Proof. Since $c_{e_i} \le c_{e_i+1}$ for all i and $\bar{y}_F = 0$ for all other F, $\{\bar{y}_{E_i}\}_{i=1}^{m-1} \cup \{\bar{y}_F\}_{\text{all other } F}$ satisfy the sign constraint $y_F \le 0, \forall F \subset E$. ($y_E$ is free so we don't care.) For the second part, fix e_k. Since E_i contains $\{e_1, \ldots, e_i\}$, e_k is contained in $\{E_k, E_{k+1}, \ldots, E_m\}$. Then

$$\sum_{F: e_k \in F} y_F = y_{E_k} + y_{E_{k+1}} + \cdots + y_{E_m} = c_{e_k} - \cancel{c_{e_{k+1}}} + \cancel{c_{e_{k+1}}} - \cancel{c_{e_{k+2}}} + \cdots + -\cancel{c_{e_m}} + \cancel{c_{e_m}} = c_{e_k}.$$

as desired. □

Lemma 1.10. *$T_i := (V, E_i \cap E(T))$ is a maximally acyclic subgraph of $H_i := (V, E_i)$ for $i \in [m]$.*

Proof. Suppose for a contradiction that $\exists e_j \in E_i \setminus E(T_i)$ such that $T_i + e_j$ is acyclic. Note that $j \le i$, so that $E(T_{j-1}) \subseteq E(T_i)$. Then if $T_i + e_j$ is acyclic, $T_{j-1} + e_j$ must be acyclic as well. Recall that in Kruskal's algorithm we ordered the set of edges first then iterate through them in that order, adding an edge to T as long as T remains acyclic. In particular, T_k is the forest computed by Kruskal's algorithm at the end of the k-th iteration for every k. The above implies that Kruskal's algorithm would have added e_j in to T on the j-th iteration, which is a contradiction because $e_j \notin E(T_i)$ and hence $e_j \notin E(T_j)$ as $E(T_j) \subseteq E(T_i)$. □

Let $\bar{x}$ be the characteristic vector of tree T constructed by Kruskal's algorithm. Note that $\bar{x}(S) = \sum_{e \in S} \bar{x}_e = |E(T) \cap S|$ for every $S \subseteq E$.

Lemma 1.11. *$\bar{x}(E_i) = n - \kappa(E_i)$ for every $i < |E|$.*

Proof. Let $C_1, \ldots, C_{\kappa(E_i)}$ be the connected components of (V, E_i). Since $T_i := (V, E(T) \cap E_i)$ is a maximally acyclic subgraph of $H_i := (V, E_i)$, the graph $(V(C_j), E(T_i) \cap E(C_j))$ is a spanning tree of C_j, so $|E(T_i) \cap E(C_j)| = |V(C_j)| - 1$. Thus,

$$\bar{x}(C_j) = |E(T) \cap E(C_j)| \ge |E(T_i) \cap E(C_j)| = |V(C_j)| - 1.$$

On the other hand, since T is a tree and therefore $\bar{x}$ is a feasible solution of $\mathrm{P_{ST}}$, we have that $\bar{x}(E(C_j)) \le n - \kappa(E(C_j))$, where $\kappa(E(C_j)) = n - |V(C_j)| + 1$ since C_j is connected. [7] Thus, $\bar{x}(E(C_j)) \le n - (n - |V(C_j)| + 1) = |V(C_j)| - 1$. Combined with the equation above, we have $\bar{x}(C_j) = |V(C_j)| - 1$, which implies that

$$\forall i < |E| : \bar{x}(E_i) = \sum_{j=1}^{\kappa(E_i)} \bar{x}(C_j) = \sum_{j=1}^{\kappa(E_i)} (|V(C_j)| - 1) = n - \kappa(E_i).$$

□

Proposition 1.12. *$\bar{x}$ and $\bar{y}$ satisfy the CS conditions for $\mathrm{P_{ST}}$ and $\mathrm{D_{ST}}$, which in turn shows that the tree T returned by Kruskal's algorithm is a MST.*

Proof.

- "*For every $F \subseteq E$, either $\bar{x}(F) = n - \kappa(F)$ or $\bar{y}_F = 0$*": Let $F \subseteq E$. If $F = E$, then $\bar{x}(E) = n - 1$. If $F = E_i$ for some $i < m$, then $\bar{x}(E_i) = n - \kappa(E_i)$. For any other F, $\bar{y}_F = 0$.
- "*For every $e \in E$, either $\sum_{F : e \in F} \bar{y}_F = c_e$ or $\bar{x}_e = 0$*": see Lemma 1.9. □

[7] Suppose $n = |V(G)| = 20$ and $|V(C_j)| = 5$. Then the graph $(V, E(C_j))$ contains $20 - 5 + 1$ components because each vertex in $V \setminus V(C_j)$ is an isolated vertex and thus a component and C_j is a component.

1.5 Maximum Cost Forest.

Roughly speaking, an algorithm is **greedy** if at each step it always chooses the locally optimal solution. It is not hard to tell that not all problems can be solved with this approach. So when does greedy work?

Maximum Cost Forest (MCF)

Given $G = (V, E)$, a **forest** is a subgraph (V, F) with $F \subseteq E$ that is acyclic. We will refer to a forest by its set of edges.

Problem (Maximum Cost Forest)**.** *Given $G = (V, E)$ and $c : E \to \mathbb{R}$, find a forest F maximizing $c(F) := \sum_{e \in F} c_e$.*

Algorithm for MCF

Given $G = (V, E)$ and $c : E \to \mathbb{R}$, suppose we want to construct a MCF for G. Intuitively, we can ignore the set of edges that have non-positive costs, because adding those edges will not give us a better solution. Define $E^+ := \{e \in E : c_e > 0\}$ and $E^- := \{e \in E : c_e \leq 0\}$.

Lemma 1.13. *Let F^* be the MCF of $G^+ := (V, E^+)$. Then F^* is also a MCF of G.*

Proof. Clearly F^* is a forest of G. Let F be a MCF of G. Then

$$\sum_{e \in F} c_e = \sum_{e \in F \cap E^+} c_e + \sum_{e \in F \cap E^-} c_e \leq \sum_{e \in F \cap E^+} c_e \leq \sum_{e \in F^* \cap E^+} c_e,$$

where the last inequality follows since $F \cap E^+$ is a forest of G^+ and F^* is a MCF of G^+. □

From now on, we may assume $c_e > 0$ for all $e \in E$. Now let $G = (V, E)$ have connected components defined by the vertex set $(V_1, \ldots, V_p)$ for some $p \geq 1$. We define $\gamma(G)$ to be a uniquely defined minimal set of edges that need to be added to G to make it connected. The following algorithm computes a MCF of G:

Algorithm 2 Maximum Cost Forest

input: $G = (V, E)$, costs $c : E \to \mathbb{R}_+$
output: F, a MCF of G wrt c
main:

1: Let $G' \leftarrow (V, E')$ be the graph with vertex set V and $E' = E \cup \gamma(G)$.
2: Let $c'_e \leftarrow -c_e (< 0)$ for all $e \in E$, $c'_e \leftarrow M > 0$ for all $e \in \gamma(G)$.
3: Find MST T of G' (e.g., with Kruskal's) wrt c'.
4: Return $F \leftarrow T \setminus \gamma(G)$

In words, minimally connect the components; negate all costs for edges in the original graph and assign a positive weight for the newly-added edges; find an MST for the new graph; return the subset of edges that were in the original graph.

Correctness Proof

Intuition. Observe we connected G minimally to obtain G' and assigned positive costs to new edges $\gamma(G)$ and negative costs to old edges E. A MST T of G' then consists of all edges in $\gamma(G)$ and the most expensive edges in E wrt the original cost. Thus, we can return $T \setminus \gamma(G)$ and obtain a MCF of G.

Proposition 1.14. *The procedure above produces an MCF (wrt original costs c).*

Proof. Let F and T be defined as in the algorithm. Note that F is a forest since it is a subset of edges of a tree. Also note that any spanning tree of G' must contain all edges in $\gamma(G)$. Now let F^* be a MCF of G wrt c and let T^* be any spanning tree of G' containing F^*. Our goal is to show that F returned by the algorithm has the same cost as F^*.

First, observe

$$\begin{aligned}
\sum_{e\in T^*} c'_e &= \sum_{e\in(T^*\cap\gamma(G))} c'_e + \sum_{e\in(T^*\cap E)} c'_e && E(G') = E(G) \dot\cup \gamma(G) \\
&= M|\gamma(G)| + \sum_{e\in F^*} c'_e + \underbrace{\sum_{e\in(T^*\cap E)\setminus F^*} c'_e}_{<0} \\
&\leq M|\gamma(G)| + \sum_{e\in F^*} c'_e. && e \in E \implies c'_e < 0
\end{aligned}$$

Since F^* is a maximum cost forest in G, we have

$$\begin{aligned}
\sum_{e\in F^*} c_e \geq \sum_{e\in F} c_e &\iff \sum_{e\in F^*} c'_e \leq \sum_{e\in F} c'_e \\
&\iff M|\gamma(G)| + \sum_{e\in F^*} c'_e \leq M|\gamma(G)| + \sum_{e\in F} c'_e = \sum_{e\in\gamma(G)} c'_e + \sum_{e\in F} c'_e = \sum_{e\in T} c'_e.
\end{aligned}$$

Together, they imply that $\sum_{e\in T^*} c'_e \leq \sum_{e\in T} c'_e$. But since T is a MST of G' wrt c', we then have $\sum_{e\in T^*} c'_e \geq \sum_{e\in T} c'_e$. Together, we have

$$\sum_{e\in F^*} c_e = \sum_{e\in F} c_e$$

which concludes the proof. □

A More Direct Approach

In fact, we can modify Kruskal to obtain a direct algorithm for MCF:

Algorithm 3: Kruskal's Algorithm, for Maximum Cost Forest

input: $G = (V, E)$, costs $c : E \to \mathbb{R}$
output: H, a maximum cost forest of G
main:
1: Initialize $H = (V, \emptyset)$, i.e., a forest of isolated vertices
2: **while** $\exists e \in E : c_e > 0$ and whose endpoints are in different components of H **do**
3: $e \leftarrow$ highest cost such edge
4: Add e to H
5: **return** H

In words, we keep adding the most expensive edge to the forest as long as it does not create any cycle. It should easy to convince yourself (after understanding Kruskal's algorithm and its proofs) that if G is connected, we obtain a maximum cost spanning tree; otherwise, the algorithm returns a forest consists of maximum cost spanning trees.

Reducing MCF to MST

Suppose $G = (V, E)$ is connected. If you have the above algorithm (for MCF) as a black box, you can solve the MST problem with the following procedure:

1. Take $c'_e = -(c_e - M) > 0$ for all $e \in E$
2. Compute the MCF wrt c'.

In words, transform the cost vector to become positive and then compute an MCF wrt the new cost. The result is then an MST wrt the original cost (note G is connected).

Proposition 1.15. *The procedure above produces a MST (wrt original costs c).*

Proof. By definition of c', each MST of G wrt $-c'$ is also a MST wrt to c with their weights shifted by a constant $(n-1) \cdot M$. Thus, it suffices to show that the procedure produces a *maximum* spanning tree of G wrt c'.

Consider the forest F returned by the procedure. Since all edges have positive costs wrt c', any subgraph of G that is not maximally acyclic cannot be optimal. Thus, F must be a spanning tree of G. Since all spanning trees are forests, F must be a maximum spanning tree, as otherwise it cannot be a MCF. □

1.A Extra Comments.

Derivation of the Dual MST LP

Consider the following LP:

$$\begin{array}{lll} (\mathrm{P_{ST}}): \min & c^T x & \\ s.t. & x(E) = n-1 & (y_E) \\ & x(F) \le n - \kappa(F) \quad \forall F \subset E & (y_F) \\ & x \ge 0 & \end{array}$$

Some easy stuff first:

- The primal is minimization, so the dual is maximization.
- The constraint corresponding to y_E is an equality, so y_E is free (and thus omitted).
- The constraint corresponding to y_F is of the $\le$ type, so $y_F \le 0$ for all $F \subset E$.
- All primal variables are ≥ 0, so all dual constraints are of the $\le$ type.

The primal sign constraint $x \ge 0$ can be expanded as $x_e \ge 0, \forall e \in E$, each of which introduces a dual constraint of the $\le$ type. We know the RHS of each constraint is c_e.

For the LHS, imagine the primal LP in matrix form and take the transpose. Recall

$$x(F) = \sum_{e \in F} x_e = \sum_{e \in (E \setminus F)} 0 + \sum_{e \in F} x_e$$

where $x_e = 1$ if e is included in the tree and 0 otherwise. If we abuse the notation a bit and let "$e \in_? F$" denotes an indicator variable (analogy: "`e in F`" in Python where `e` is an element and `F` is a set), we can rewrite this as

$$x(F) = \sum_{e \in_? E} x_e (e \in_? F) = x_{e_1}(e_1 \in_? F) + x_{e_2}(e_2 \in_? F) + \cdots .$$

Thus, the rows of the form $x(F) \le n - \kappa(F)$ are essentially characteristic vectors, i.e., 1 if $e \in F$ and 0 otherwise:

$$\begin{bmatrix} e_1 \in_? F_1 & e_2 \in_? F_1 & e_3 \in_? F_1 & e_4 \in_? F_1 & \cdots \\ & & \vdots & & \\ e_1 \in_? F_r & e_2 \in_? F_r & e_3 \in_? F_r & e_4 \in_? F_r & \cdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \end{bmatrix} \le \begin{bmatrix} n - \kappa(F_1) \\ \vdots \\ n - \kappa(F_r) \end{bmatrix}$$

When you instead sum across the e-th column of the matrix for the dual, there is a contribution of 1 per row iff $e \in F$ where F is the edge set associated with that row. For

example, the 1st, 3rd, and 4th row/entry contributes to the sum of the first column:

$$\begin{bmatrix} e_1 \in F_1 \\ e_1 \notin F_2 \\ e_1 \in F_3 \\ e_1 \in F_4 \\ e_1 \notin F_5 \end{bmatrix} \implies 1y_1 + 0y_2 + 1y_3 + 1y_4 + 0y_5 = \sum_{F:e\in F} y_F \leq c_{e_1}.$$

Thus, these constraints are of the form

$$\forall e \in E : \sum_{F:e\in F} y_F \leq c_e.$$

Next, expand Constraint 2 as

$$\begin{aligned} x(F_1) &\leq n - \kappa(F_1) && (y_{F_1}) \\ &\cdots \\ x(F_\ell) &\leq n - \kappa(F_\ell) && (y_{F_\ell}) \\ x(E) &= n - 1 && (y_E) \end{aligned}$$

Recall the dual objective function is $\min b_1 y_1 + \cdots + b_m y_m$. In this case, we have

$$\max \left(\sum_{F\subset E} (n - \kappa(F)) y_F \right) + (n - \kappa(E)) y_E$$

We can merge the second term into the summation. Thus, the dual objective becomes

$$\max \sum_{F\subseteq E} (n - \kappa(F)) y_F$$

Altogether, the dual LP is given by

$$\begin{array}{lll} (\mathrm{D_{ST}}) : \max & \sum_{F\subseteq E} (n - \kappa(F)) y_F & \\ s.t. & \sum_{F:e\in F} y_F \leq c_e & \forall e \in E \\ & y_F \leq 0 & \forall F \subset E \end{array}$$

An Alternative LP for MST

$$\begin{array}{lll} (\mathrm{P'_{ST}}) : \min & c^T x & \\ s.t. & x(E) = n - 1 & \\ & x(E(S)) \leq |S| - 1 & \forall \varnothing \subsetneq S \subsetneq V \\ & x \geq 0 & \end{array}$$

The equivalence between $\mathrm{P'_{ST}}$ and $\mathrm{P_{ST}}$ is explored in A1Q5.

1.B Practice Problems.

Claim. *Let G be an undirected connected weighted graph with at least one cycle. Let C be a cycle of G and let e be an edge that has strictly greater cost than all other edges in the cycle. Show that e does not belong to any MST of G.*

Proof. Suppose there exists a MST T of G containing e. Let T_1, T_2 be subtrees obtained by deleting e from T. Then $T \cap \delta(V(T_1)) = \{e\}$. Note that every cut has even intersection with every cycle, so there exists another edge $f \in \delta(V(T_1)) \cap C$, $f \neq e$. Since $f \notin T$ and $c(f) < c(e)$, we can replace e with f to get a smaller spanning tree. Contradiction. □

Claim. *Prove that for any weighted undirected graph such that the weights are distinct, the MST is unique.*

Proof. Let T_1, T_2 be distinct MST. Let $e = \{u, v\}$ be the cheapest edge that belongs to only one of the two spanning trees (it exists since they are distinct). We may assume that $e \in E(T_1)$. Now consider $T_2 \cup \{e\}$. It contains a cycle C which contains e. If any edge f of that cycle has cost strictly greater than e, then $T_2 \cup \{e\} \setminus \{f\}$ is a tree with cost $c(T_2) + c_e - c_f < c(T_2)$, contradicting the fact that T_2 is an MST. Thus, all edge f of $C \setminus \{e\}$ satisfies $c_f < c_e$ (since the weights are distinct). Since there must be an edge $f \in C \setminus \{e\}$ such that $f \notin E(T_1)$, this contradicts the choice of e. □

Claim. *Show that the following algorithm finds an MST of a connected graph G in polytime: Start with $H = G$. At each step, find a max-cost edge e such that $H \setminus e$ is connected and delete it from H. If no such edge exists, then stop and return H.*

Proof. Since the graph is connected, the algorithm clearly computes a spanning tree, since it deletes edges until the graph is minimally connected. Let $E = \{e_1, e_2, \ldots\}$ be such that $c(e_i) \geq c(e_{i+1})$ for all i. Let $E_0 = E$ and for $i \geq 1$, let $E_i := E_{i-1} - e_i$ is e_i is not a cut edge in E_{i-1}, otherwise $E_i = E_{i-1}$. By induction, it is sufficient to argue that (V, E_i) contains some MST of (V, E_{i-1}) for all $i \geq 1$.

Suppose this is not true. Then there is some i such that $E_i = E_{i-1} - e_i$ and every MST of (V, E_{i-1}) contains e_i. Let T be some MST of (V, E_{i-1}). Since e_i is not a cut edge of E_{i-1}, there exists an edge f such that $T \cup f$ contains a cycle that also contains e_i. Note that $c(f) \leq c(e_i)$ by the ordering of the edges. Thus, $T \cup \{f\} \setminus \{e_i\}$ is a spanning tree in (V, E_{i-1}) with cost at most $c(T)$. Contradiction. □

Part II

Matroids

2 MATROID THEORY

In this chapter, we will define and study matroids and focus on how greedy algorithms work on matroids. Our plan is as follows.

1. Identify the underlying structure for MCF and its algorithm. Abstract the ideas.
2. Define matroids and develop algorithms and characterizations of matroids:
   (a) Independent set definition.
   (b) Augmentation property.
   (c) Circuit characterization.
   (d) Basis characterization.

Revisit: Maximum Cost Forest

Let us refer to a forest by its edges. Let $\mathcal{I} \subseteq 2^E$ be the set of all possible forests (2^E denotes the power set of E). We can write the algorithm in a more generic form.

Algorithm 4: Kruskal's Algorithm, for Maximum Cost Forest, Generic Form

input: $G = (V, E)$ connected, costs $c : E \to \mathbb{R}$
output: F, (the edge set of) a maximum cost forest of G
main:

```
F ← ∅
while ∃e ∈ E : F ∪ {e} ∈ I and c_e > 0 do
    Choose such e with largest c_e
    F ← F ∪ {e}
return F
```

Observe the set of forests $\mathcal{I}$ satisfies the following properties:

1. $\emptyset \in \mathcal{I}$: the empty set is a trivial forest.
2. If $F \in \mathcal{I}$ and $F' \subseteq F$, then $F' \in \mathcal{I}$: any subset of a forest is acyclic and thus a forest.
3. Fix $A \subseteq E$ and consider the subgraph $G' = (V, A)$. Label its connected components $H_1, \ldots, H_k$. Every inclusion-wise maximal element of $\mathcal{I}$, that is, a maximally acyclic subgraph (i.e., a forest) of G', contains exactly $|V(H_i)| - 1$ edges from each H_i. Therefore, they all have the same cardinality.

It turns out that these are precisely the properties we need that make the greedy algorithms work. Let us introduce the notion of matroid, which is a structure that abstracts and generalizes the notion of *linear independence* in vector spaces.

2.1 Matroid.

Independent Set Definition

There are many equivalent ways to define a (finite) matroid. We will start with the one using independent sets. Let 2^S denote the power set of S.

Definition 2.1. Let S be a (finite) ground set and $\mathcal{I} \subseteq 2^S$. If $\mathcal{I}$ satisfies

(M1) $\varnothing \in \mathcal{I}$;

(M2) *if $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$,*

then the pair $(S, \mathcal{I})$ is called an **independence system**, the elements $I \in \mathcal{I}$ are called **independent sets**, and the elements of $2^S \setminus \mathcal{I}$ are said to be **dependent**. Minimal dependent sets are called **circuits**; maximal independent sets are called **bases**.

Definition 2.2. Let $(S, \mathcal{I})$ be an independence system. For $A \subseteq S$, the maximal independent subsets of A are called the **bases of A.**

Definition 2.3. If $(S, \mathcal{I})$ satisfies M1 and M2 and additionally satisfies

(M3) *for all $A \subseteq S$, every inclusion-wise maximal element of $\mathcal{I}$ contained in A, i.e., the bases of A, has the same cardinality,*

then the pair $\mathcal{M} = (S, \mathcal{I})$ is called a **matroid**.

Three Examples

Ex.1: Let $\mathcal{I}$ be the set of all forests. Then $(E(G), \mathcal{I})$ is called the **graphic/forest matroid**. ◊

Ex.2: Let $S = \{1, \ldots, n\}$, $r \in \{0, \ldots, n\}$, and $\mathcal{I}$ be the set of all subsets of S with at most r elements. Then $U_n^r = (S, \mathcal{I})$ is called the **uniform matroid of rank r**. Observe:

1. $|\varnothing| = 0 \leq r \implies \varnothing \in \mathcal{I}$. OK.
2. $(A \in \mathcal{I}) \wedge (B \subseteq A) \implies |B| \leq |A| \leq r \implies B \in \mathcal{I}$. OK.
3. For all $A \subseteq S$, every basis of $\mathcal{I}$ contained in A must have size $\min(|A|, r)$. OK. ◊

Ex.3. Let N be an $m \times n$ matrix and $S = \{1, \ldots, n\}$ (column indices). Define $\mathcal{I} := \{A \subseteq S :$ columns indexed by A are linearly independent$\}$. Then $(S, \mathcal{I})$ is called a **linear matroid**.

1. $\varnothing \in \mathcal{I}$: yes, $\varnothing$ is linearly independent. OK.
2. Any subset of a linear independent set is linearly independent. OK.
3. Follows from linear algebra; bases of vector space translate to bases of $A \subseteq S$. OK. ◊

2.2 *Maximum Weight Independent Set Problem (for Independence Systems).*

Consider the following generic optimization problem.

Problem (Maximum Weight Independent Set)**.** *Given an independence system $\mathcal{M} = (S, \mathcal{I})$ and weights $c : S \to \mathbb{R}^+$, find $A \in \mathcal{I}$ maximizing $c(A) := \sum_{e \in A} c_e$.*

We can solve this using a generic greedy algorithm given below.

Algorithm 5: Generic greedy algorithm

input: independence system $\mathcal{M} = (S, \mathcal{I})$, costs $c : S \to \mathbb{R}^+$
output: $A \in \mathcal{I}$ maximizing $c(A) := \sum_{e \in A} c_e$.
main:

```
F ← ∅
while ∃e ∈ S : F ∪ {e} ∈ I and c_e > 0 do
    Choose such e with largest c_e
    F ← F ∪ {e}
return F
```

Optimality Proof

This greedy algorithm does not always yield the optimal solution. In fact, it finds the optimal solution if and only if $\mathcal{M} = (S, \mathcal{I})$ is a matroid.

Theorem 2.4 (Rado, Edmonds)**.** *The greedy algorithm finds the maximum weight independent set when $\mathcal{M}$ is a matroid.*

Proof. A later result implies this proof. See Theorem 2.9. □

Theorem 2.5 (Rado, Edmonds)**.** *Let $\mathcal{M} = (S, \mathcal{I})$ be an independence system. Then the greedy algorithm finds an optimal independent set for all [a] $c : S \to \mathbb{R}^+$ iff $\mathcal{M}$ is a matroid.*

[a] If $\mathcal{M}$ is not a matroid, the greedy might still find an optimal solutions for some c, but there always exists some c' for which the greedy algorithm fails to find the optimal solution.

Proof. $\Longrightarrow$: We show the contrapositive. Suppose $\mathcal{M}$ is not a matroid. Then $\mathcal{M}$ does not satisfy M3. Let $A \subseteq S$ such that A_1, A_2 are two bases of A with $|A_1| < |A_2|$. Define

$$c_e = \begin{cases} 1 + \varepsilon & e \in A_1 \\ 1 & e \in A_2 \\ 0 & \text{otherwise} \end{cases}$$

In this case, the greedy algorithm outputs A_1 with total weight

$$\sum_{e \in A_1} c_e = (1+\varepsilon)|A_1|,$$

as each step elements in A_1 have the highest weight and A_1 is inclusion-wise maximal. Note that A_2 is another candidate with total weight

$$\sum_{e \in A_2} c_e = |A_2|.$$

So if we choose ε small enough where

$$\varepsilon < \frac{|A_2|}{|A_1|} - 1,$$

then A_1 is not a maximum weighted independent set, which proves the contrapositive.

$\Longleftarrow$: Follows immediately from Theorem 2.4. □

Example. For a more explicit example for the $\Longrightarrow$ direction, consider $|A_1| = 2$ and $|A_2| = 3$. Then $\varepsilon < 3/2 - 1 = 0.5$. Thus, if we have $c_e = 1.4 < 1.5 = 1 + \varepsilon$ for $e \in A_1$ and $c_e = 1$ for $e \in A_2$, the greedy algorithm outputs A_1 with a total weight of $1.4 + 1.4 = 2.8$ which is inferior to A_2 with a total weight of $1 + 1 + 1 = 3$. ◊

Rank

For an independence system $(S, \mathcal{I})$, we use $r(A)$ and $\rho(A)$ to denote the size of the largest and smallest basis of $A \subseteq S$, respectively.

Definition 2.6. Let $\mathcal{M} = (S, \mathcal{I})$ be an independence system. For $A \subseteq S$, we define the **rank** of A, denoted $r(A)$, as

$$r(A) := \max\{|B| : B \subseteq A, B \in \mathcal{I}\}.$$

Additionally, we define $\rho(A)$, sometimes called the **low rank** of A, as

$$\rho(A) := \min\{|B| : B \text{ is a basis of } A\}$$

The following statement follows directly from the definition of matroid:

Proposition 2.7. *$\mathcal{M} = (S, \mathcal{I})$ is a matroid iff $\rho(A) = r(A)$ for all $A \subseteq S$.*

Proof. Omitted. □

Rank Quotient

Definition 2.8. Let $\mathcal{M} = (S, \mathcal{I})$ be an independence system. The **rank quotient** of $\mathcal{M}$, denoted $q(S, \mathcal{I})$, is defined as

$$q(S, \mathcal{I}) := \min_{A \subseteq S} \frac{\rho(A)}{r(A)}.$$

Note that we always have $q(S, \mathcal{I}) \leq 1$ and $\mathcal{M}$ is a matroid iff $q(S, \mathcal{I}) = 1$.

The following theorem implies that if $\mathcal{M}$ is a matroid, then the greedy algorithm finds an optimal solution (which proves Theorem 2.4).

Theorem 2.9 (Jenkins). *Let $(S, \mathcal{I})$ be an independence system. Let $\mathrm{GR}(S, \mathcal{I})$ be the total weight of the independent set found by the greedy algorithm and $\mathrm{OPT}(S, \mathcal{I})$ be the optimal solution weight. Then*

$$\mathrm{GR}(S, \mathcal{I}) \geq q(S, \mathcal{I}) \cdot \mathrm{OPT}(S, \mathcal{I}).$$

In particular, when $(S, \mathcal{I})$ is a matroid, $\mathrm{GR}(S, \mathcal{I}) = \mathrm{OPT}(S, \mathcal{I})$.

Proof. Let $S = \{e_1, \ldots, e_m\}$ with $c_{e_1} \geq \cdots \geq c_{e_m}$ and let $S_j = \{e_1, \ldots, e_j\}$ for all $j = 1, \ldots, m$. Let G be the solution obtained by the greedy algorithm and let σ be the optimal solution. Note that $G, \sigma \in \mathcal{I}$ and $G, \sigma \subseteq S$.

Define $G_j = G \cap S_j$ and $\sigma_j = \sigma \cap S_j$. Let $G_0 = \emptyset = \sigma_0$. Then

$$\begin{aligned}
c(G) &= \sum_{e_j \in G} c_{e_j} \\
&= \sum_{j=1}^{m} (\underbrace{|G_j| - |G_{j-1}|}_{\star}) c_{e_j} && \star = 1 \text{ if } e_j \in G \text{ and } 0 \text{ otherwise} \\
&= \sum_{j=1}^{m-1} |G_j|(c_{e_j} - c_{e_{j+1}}) + c_{e_m}|G_m| && \text{algebra} \\
&= \sum_{j=1}^{m} |G_j| \underbrace{(c_{e_j} - c_{e_{j+1}})}_{\Delta_j \geq 0}, && \text{algebra} \\
&= \cdots
\end{aligned}$$

where $c_{e_m+1} = 0$.

At step j, the greedy algorithm computes a maximal independent subset of S_j, so G_j is a

basis of S_j. By definition, $|G_j| \geq \rho(S_j)$. Define $\Delta_j := c_{e_j} - c_{e_{j+1}}$. Then

$$\begin{aligned}
c(G) = \cdots = \sum_{j=1}^{m} |G_j|\Delta_j & && \text{cont'd from last page} \\
&\geq \sum_{j=1}^{m} \rho(S_j)\Delta_j && |G_j| \geq \rho(S_j) \\
&\geq \sum_{j=1}^{m} q(S,\mathcal{F})r(S_j)\Delta_j && \text{definition of } q(S,\mathcal{F}) \\
&\geq q(S,\mathcal{F}) \sum_{j=1}^{m} |\sigma_j|\Delta_j && \sigma_j = (\sigma \cap S_j) \subseteq S_j \\
&= q(S,\mathcal{F})c(\sigma), &&
\end{aligned}$$

i.e., $\mathrm{GR}(S,\mathcal{F}) \geq q(S,\mathcal{F}) \cdot \mathrm{OPT}(S,\mathcal{F})$ as desired. □

Remark. It is easy to see that the greedy algorithm terminates in polynomial time assuming that we can test independence efficiently. ◊

Remark. This is an approximation algorithm. See the last chapter for more information. ◊

2.3 *Augmentation Property of Matroids.*

Augmentation Property

Given two independent sets of different sizes, we can find an element from the larger set such that including it in the smaller set does not violates its independence constraint.

Theorem 2.10. *Let $\mathcal{M} = (S, \mathcal{I})$ be an independent system. Then $\mathcal{M}$ is a matroid iff it satisfies M3': $\forall X, Y \in \mathcal{I}$ such that $|X| > |Y|$, $\exists x \in X \setminus Y : Y \cup \{x\} \in \mathcal{I}$.*

Proof. M3: $\forall A \subseteq S$, every basis of A has the same cardinality. We show that M3 $\iff$ M3'.

M3 $\implies$ M3': Let $X, Y \in \mathcal{I}$ such that $|X| > |Y|$. Consider $A := X \cup Y$. Y is clearly not a basis of A since X is an independent subset of A with greater cardinality. Thus, Y is not inclusion-wise maximal and we can choose some element of A to augment it. But our only choice is from $X \setminus Y$, yielding the desired result.

M3' $\implies$ M3: Fix a basis B of A. If any other basis B' of A is smaller, we can add an element to B' from $B \subseteq A$ and remain independent. If any other basis B'' of A is larger, than our original basis B could have been augmented with an element of $B'' \setminus B$. Both contradict the definition of bases (as all bases must have the same size). □

Example: How to use M3'

Let $G = (V, E)$, $W \subseteq V$ a stable set.[1] For each $v \in W$, we define $k_v \in \mathbb{Z}^+$ as an upper bound of number of edges from any $F \subseteq E$ that v may incident to.

Claim. *$(E, \mathcal{I})$ given by $\mathcal{I} := \{F \subseteq E : |\delta(v) \cap F| \leq k_v$ for all $v \in W\}$ is a matroid.*

Proof. M1 and M2 hold (easy). Suppose $X, Y \subseteq E$, $X, Y \in \mathcal{I}$, and $|X| > |Y|$. Consider the set of vertices that have used up its allocation in Y: $W_Y = \{v \in W : |\delta(v) \cap Y| = k_v\}$. By the hand-shaking lemma, $2|X| = \sum_{v \in V} |X \cap \delta(v)|$. We can split the RHS as:

$$2|X| = \sum_{v \in W_Y} |X \cap \delta(v)| + \sum_{v \in W \setminus W_Y} |X \cap \delta(v)| + \sum_{v \notin W} |X \cap \delta(v)|$$
$$2|Y| = \sum_{v \in W_Y} |Y \cap \delta(v)| + \sum_{v \in W \setminus W_Y} |Y \cap \delta(v)| + \sum_{v \notin W} |Y \cap \delta(v)|$$

By construction, for each $v \in W_Y$, $|Y \cap \delta(v)| = k_v \geq |X \cap \delta(v)|$. But $|X| > |Y|$, which means there must be some $x \in X \setminus Y$ satisfying $x \in \delta(v)$ for some $v \notin W_Y$, such that $Y \cup \{x\} \in \mathcal{I}$. By M3', this is a matroid. □

[1] A **stable set** refers to a set of vertices no two of which are adjacent. We sometimes call it "independent set" in graph theory but this term is already used in matroid theory.

2.4 Circuit Characterization of Matroids.

Alternatively, one may describe a matroid by indicating the set of circuits of $\mathcal{M}$, denoted $\mathcal{C}$, since a set $A \subseteq S$ is independent iff it does not contain any of the circuits as a subset:

$$A \in \mathcal{I} \iff \nexists C \in \mathcal{C} : C \subseteq A.$$

Suppose you are given a set of circuits $\mathcal{C} \subseteq 2^S$. How can you tell whether $\mathcal{C}$ induces a matroid? We start with the following result. For intuition, consider the graphic matroid and recall that adding any $e \in E$ to a forest A introduces at most 1 cycle.

Proposition 2.11. *Let $\mathcal{M} = (S, \mathcal{I})$ be a matroid. Then for all $A \in \mathcal{I}$, for all $e \in S$, $A \cup \{e\}$ contains at most one circuit.*

Proof. Let A be a minimal (smallest) counterexample. Then there exists some e such that $A \cup \{e\}$ contains two distinct circuits C_1, C_2. We must have $e \in C_1 \cap C_2$ as otherwise A already contains a circuit. By the minimality of A, $A \cup \{e\} = C_1 \cup C_2$. Moreover, by minimality of circuits, neither of them can be a subset of the other. Thus, we can choose $e_1 \in C_1 \setminus C_2$ and $e_2 \in C_2 \setminus C_1$.

Consider $A' := (C_1 \cup C_2) \setminus \{e_1, e_2\}$. If A' has a circuit C, then $C \neq C_1, C_2$ as we deleted an element from both circuits. On the other hand, $A - e_1 + e$ contains C_2 and C since $e_1 \neq C_2$ and $A + e \supseteq A'$. Then A was not a minimal counterexample as we could've taken $A - e_1$. Thus, A' is independent. In addition, A is a basis of $C_1 \cup C_2$ as it is maximally independent. But so is A', as adding neither e_1 or e_2 results in a circuit. Thus, A and A' are bases with $|A'| < |A|$, contradicting M3. □

Theorem 2.12. *Let $\mathcal{C} \subseteq 2^S$. Then $\mathcal{C}$ is a set of circuits of a matroid iff*

(C1) $\varnothing \notin \mathcal{C}$

(C2) If $C_1, C_2 \in \mathcal{C}$, $C_1 \subseteq C_2$, then $C_1 = C_2$.

(C3) If $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$, $e \in C_1 \cap C_2$, then $\exists C \in \mathcal{C}$ s.t. $C \subseteq (C_1 \cup C_2) \setminus \{e\}$.

Proof. $\implies$ C1 and C2 are clear. Suppose C3 is violated. Then $A := (C_1 \cup C_2) \setminus \{e\} \in \mathcal{I}$. In particular, $A \cup \{e\}$ has 2 distinct circuits, contradicting the previous theorem.

$\impliedby$ Define $\mathcal{I} := \{A \subseteq S : \nexists C \in \mathcal{C}, C \subseteq A\}$. M1 and M2 follows by definition. Suppose M3 is false. We can choose A_1, A_2 bases of $A \subseteq S$ such that $|A_1| < |A_2|$, maximizing $|A_1 \cap A_2|$. Since A_1 is a basis, we can pick $e \in A_1 \setminus A_2$. Then $A_2 \cup \{e\}$ contains a unique circuit C.

Let $f \in C \setminus A_1$. Then $A_3 := (A_2 \cup \{e\}) \setminus \{f\} \in \mathcal{I}$ since we removed the unique circuit. But then $|A_3 \cap A_1| > |A_2 \cap A_1|$, contradicting the choice of A_1 and A_2. □

2.5 Basis Characterization of Matroids.

Another way one could describe a matroid is to give the set of bases of $\mathcal{M}$, denoted $\mathcal{B}$, since a set $A \subseteq S$ is independent iff it is a subset of some basis B in $\mathcal{B}$:

$$A \in \mathcal{I} \iff \exists B \in \mathcal{B} : A \subseteq B.$$

Now suppose you are given a set of circuits $\mathcal{B} \subseteq 2^S$. The following theorem tells us whether $\mathcal{B}$ induces a matroid. (B2 (and B2′) is known as the **basis exchange property**.)

Theorem 2.13. *Let $\mathcal{B} \subseteq 2^S$. Then $\mathcal{B}$ is the set of bases of a matroid $(S, \mathcal{I})$ iff*

(B1) $\mathcal{B} \neq \varnothing$;

(B2) $\forall B_1, B_2 \in \mathcal{B}, x \in B_1 \setminus B_2 : \exists y \in B_2 \setminus B_1 : (B_1 \setminus \{x\}) \cup \{y\}) \in \mathcal{B}$.

Proof.

($\Longrightarrow$) By M1, $\mathcal{I}$ is not empty. Thus, there exists a maximal independent set implying B1. For bases B_1, B_2 and $x \in B_1 \setminus B_2$, we have that $B_1 \setminus \{x\}$ is independent. By the augmentation property, there is some $y \in B_2 \setminus B_1$ such that $(B_1 \setminus \{x\}) \cup \{y\}$ is independent. Since it matches the size of B_1 and B_2, it is also a basis.

($\Longleftarrow$) Let $\mathcal{B}$ satisfy B1 and B2. We claim that all elements of $\mathcal{B}$ have the same cardinality. Suppose otherwise. Let $B_1, B_2 \in \mathcal{B}$ with $|B_1| > |B_2|$ such that $|B_1 \cap B_2|$ is maximum. Let $x \in B_1 \setminus B_2$. By B2, there exists some $y \in B_2 \setminus B_1$ with $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$, contradicting the maximality of $|B_1 \cap B_2|$. Thus, all elements of $\mathcal{B}$ have the same size.

Now let $\mathcal{F} := \{F \subseteq S : \exists B \in \mathcal{B}, F \subseteq B\}$ be the union of all subsets of some $B \subseteq \mathcal{B}$. Then $(S, \mathcal{F})$ is an independence system with the family of bases $\mathcal{B}$. To show that $(S, \mathcal{F})$ satisfies M3, let $X, Y \in \mathcal{F}$ with $|X| > |Y|$. Let $X \subseteq B_1$ and $Y \subseteq B_2$, with $B_1, B_2 \in \mathcal{B}$ chosen such that $|B_1 \cap B_2|$ is maximum. If $B_2 \cap (X \setminus Y) \neq \varnothing$, we are done because we can augment Y. (M3′)

We claim that the other case, $B_2 \cap (X \setminus Y) = \varnothing$, is impossible. Suppose $B_2 \cap (X \setminus Y) = \varnothing$. Then we get

$$|B_1 \cap B_2| + |Y \setminus B_1| + |(B_2 \setminus B_1) \setminus Y| = |B_2| = |B_1| \geq |B_1 \cap B_2| + |X \setminus Y|.$$

But $|X \setminus Y| > |Y \setminus X| \geq |Y \setminus B_1|$, which implies $(B_2 \setminus B_1) \setminus Y \neq \varnothing$. Choose $y \in (B_2 \setminus B_1) \setminus Y$. By B2, there exists an $x \in B_1 \setminus B_2$ with $(B_2 \setminus \{y\}) \cup \{x\} \in \mathcal{B}$, contradicting the maximality of $|B_1 \cap B_2|$. □

Another form of the basis exchange property is given below.

Theorem 2.14. *Let $\mathcal{B} \subseteq 2^S$. Then $\mathcal{B}$ is the set of bases of a matroid $(S, \mathcal{I})$ iff*

(B1) $\mathcal{B} \neq \emptyset$;

(B2') $\forall B_1, B_2 \in \mathcal{B}, y \in B_2 \setminus B_1 : \exists x \in B_1 \setminus B_2 : (B_1 \setminus \{x\}) \cup \{y\}) \in \mathcal{B}$.

Proof.

($\Longrightarrow$) By M1, $\mathcal{I}$ is not empty. Thus, there exists a maximal independent set implying B1. For B2, let $B_1, B_2 \in \mathcal{B}$ and $y \in \mathcal{B}_2 \setminus B_1$ and consider $B_1 \cup \{y\}$. Since B_1 is a basis, $B_1 \cup \{y\}$ contains a unique circuit C and $y \in C$. Then $C \cap B_1$ is not empty, as otherwise $C = \{y\}$ which contradicts $B_2 \in \mathcal{I}$. Since $y \in B_2$, we have that $x \neq y$. Then $(B_1 \setminus \{x\}) \cup \{y\}$ has no circuit. Thus, $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{I}$. Since it has the same size as B_1, it must also be a basis. Thus, $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$ as desired.

($\Longleftarrow$) Let $\mathcal{I} \subseteq 2^S$ where $I \in \mathcal{I}$ iff $I \subseteq B$ for some $B \in \mathcal{B}$. Note B1 implies M1, and M2 holds by definition of $\mathcal{I}$. It suffices to show that M3' holds.

Let $I_1, I_2 \in \mathcal{I}$ be a counterexample, i.e., $|I_1| > |I_2|$ but for all $e \in I_1 \setminus I_2$, $I_2 \cup \{e\} \notin \mathcal{I}$. Assume I_1, I_2 are chosen so that $|I_1 \cap I_2|$ is maximum. By definition of $\mathcal{I}$, there exists $B_1, B_2 \in \mathcal{B}$ such that $I_1 \subseteq B_1$ and $I_2 \subseteq B_2$. Assume B_1, B_2 are chosen so that $|B_1 \setminus (B_2 \cup I_1)|$ is minimum.

If $I_2 \setminus B_1$ is empty (i.e., $I_2 \subseteq B_1$), then for any $e \in I_1 \setminus I_2$, $I_2 \cup \{e\} \subseteq B_1$. Thus, $I_2 \cup \{e\}$ is independent. Contradiction. Thus, $I_2 \setminus B_1$ is not empty. Let $y \in I_2 \setminus B_1$.

By B2', there exists $x \in B_1 \setminus B_2$ such that $B_1' := (B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. If $x \in B_1 \setminus (B_2 \cup I_1)$, then $x \notin I_1$, so $I_1 \subseteq B_1'$ and

$$|B_1' - (B_2 \cup I_1)| < |B_1 - (B_2 \cup I_1)|,$$

contradicting our choice of B_1. Thus, we may assume $x \in I_1 \setminus B_2$. Let $I_1' := (I_1 \setminus \{x\}) \cup \{y\}$. Then $|I_1'| = |I_1|$ and I_1' is independent as $I_1' \subseteq B_1'$. Since $|I_1' \cap I_2| > |I_1 \cap I_2|$, there exists $e \in I_1' \setminus I_2$ such that $I_2\{e\}$ is independent. Since $I_1' \setminus I_2 \subseteq I_1 \setminus I_2$, we get $e \in I_1 \setminus I_2$. This contradicts our initial assumption. □

2.A Examples of Matroids - Intuition.

If you are hopelessly confused by matroids as I was, try the following detailed example.

Graphic Matroid (Forest Matroid)

Given a graph $G = (V, E)$:

- The ground set is the set of edges E.
- We look at sets of edges $A \subseteq E$.
- A set $A \subseteq E$ is independent if the graph (V, A) does not contain any cycles.
- Minimal dependent sets, i.e., the circuits, correspond to the cycles of the graph.
- Maximal independent sets of $A \subseteq E$, i.e., bases of A, correspond to spanning forests of the subgraph (V, A).

Verify the axioms and characterizations:

- M1: $(V, \varnothing)$ has no edges and is trivially independent.
- M2: If (V, A) has no cycles, then (V, A') for $A' \subseteq A$ has no cycles as well.
- M3: Given a set of edges $A \subseteq E$, a basis B of A consists of components $B_1, \ldots, B_k$ where B_i is a spanning tree of component A_i of A. Since spanning trees of the same component have the same size, all bases of A have the same size.
- M3': Consider two independent sets $A, B \in \mathcal{I}$ such that $|A| > |B|$. If $|B| \leq 1$, then the graph $(V, B \cup \{x\})$ for any $x \in A \setminus B$ consists of one or two edges only and thus cannot form any cycle. If $|B| > 1$, the only case that we cannot find any $x \in A \setminus B$ where $(V, B \cup \{x\})$ is a forest such that every component of B is one edge away from forming a cycle and A is unfortunately made up of those edges. But it takes at least three edges to form a cycle and we have $|A| > |B|$, so this is impossible.
- The circuit characterization is trivial because they correspond to cycles.
- It is helpful to look at the bases characterization using graphic matroids. Suppose T_1 and T_2 are two spanning trees of a connected graph G and $e_1 \in E(T_1)$. Clearly, if $e_1 \in E(T_2)$ then we could exchange e_1 with itself. Now suppose that $e_1 \notin T_2$. Then $T_2 + e_1$ contains a unique cycle C, each edge of which can be "traded" to T_1. To be more precisely, for every $e_2 \in C$, $T_2 + e_1 - e_2$ is still a spanning tree. It is then easy to see that adding back some edge from $C - e$ can reconnect the two connected components resulted from removing e_1 from T_1. [2]

[2]Reference: here.

2.B Practice Problems.

Claim. *Let $D = (V, A)$ be a digraph and N be its incidence matrix. Let M be the linear matroid defined by N over $\mathbb{Q}$. Prove that M is the graphical matroid of the undirected graph corresponding to D.*

Proof. Let $G = (V, E)$ be the underlying undirected graph of D. We show that the dependent sets $M(N)$ corresponds to non-empty edge sets of even subgraphs of G (subgraphs with all vertex degrees even). This will show that a set is dependent iff it has a cycle.

Let $C = \{e_1, \ldots, e_n\}$ be an edge set of an even subgraph of G. Decompose C into edge disjoint cycles $C_1, \ldots, C_k$ and give arbitrary orientations for each cycle. Let $a_i = 1$ if e_i appears forward in the orientation for the cycle it is in, otherwise $a_i = -1$. Then $\sum_{i=1}^{n} a_i N[e_i] = 0$. Thus, C is a dependent set of M.

Conversely, let $C = \{e_1, \ldots, e_n\}$ be a dependent set of M. Then there exists a non-trivial linear combination of $N[e_1], \ldots, N[e_n]$ over $\mathbb{Q}$ where the sum is 0. By scaling, we may assume each coefficient in this linear combination is integer and all integers are relatively prime. So we may assume at least one of the coefficient is an odd integer. In other words, there exists $a_1, \ldots, a_n \in \mathbb{Z}$ such that $a_1, \ldots, a_n$ are relatively prime and $\sum_{i=1}^{n} a_i N[e_i] = 0$. By taking modulo 2, we can see that this is a non-trivial linear combination of $K[e_1], \ldots, K[e_n]$ over $\mathbb{Z}_2$ where K is the incidence matrix of G, i.e.,

$$\sum_{i=1}^{n} a_i' K[e_i] \equiv 0 \mod 2$$

where $a_i' = a_i \bmod 2$ and $K[e_i] = N[e_i] \bmod 2$. This implies that, in C, every vertex is incident to an even number of edges of the undirected graph induced by C. Thus, C is an edge set of even subgraphs of G. □

Claim. *Let $G = (V, E)$ be an undirected graph and $\mathcal{I} := \{J \subseteq E : \kappa(G - J) = \kappa(G)\}$, where $\kappa(G)$ is the number of connected components of G. Show that $\mathcal{M} = (E, \mathcal{I})$ is a matroid.*

Proof. Let $\mathcal{M}' := \mathcal{M}(G) = (E, \mathcal{I}')$ be the graphic matroid on G and let $J \subseteq E$. Note that $r_{\mathcal{M}'}(E - J) = |V| - \kappa(G - J)$. Thus, $J \subseteq \mathcal{I}$ iff $r_{\mathcal{M}'}(E - J) = |V| - \kappa(G) = r_{\mathcal{M}'}(E)$. Thus, M is the dual matroid of $\mathcal{M}'$. As a remark, note that $r_{\mathcal{M}}(X) = |X| - \kappa(G - X) + \kappa(G)$ for any subset X of E. □

Claim. *Consider an undirected graph $G = (V, E)$ and define*

$$\mathcal{I} := \{J \subseteq E : \text{ each component of } (V, J) \text{ has at most one cycle}\}.$$

Show that $(E, \mathcal{I})$ is a matroid.

Proof. M1 and M2 hold trivially. For M3, let $A \subseteq E$ be a set of edges and consider the graph $G[A] = (V, A)$ induced by A. Let $(V_1, A_1), \ldots, (V_p, A_p)$ be the connected components of this graph. Consider the i-th component for $1 \leq i \leq p$ and let J be an inclusion-wise maximal subset of A.

We claim that $J \cap A_i$ has cardinality $|V_i| - 1$ if (V_i, A_i) has no cycles, and it has $|V_i|$ otherwise.

Since (V_i, A_i) is connected and J is maximal, it clearly has at least $|V_i| - 1$ edges from A_i. J also has no more than $|V_i|$ edges from A_i as otherwise we could pick a maximal tree T in $J \cap A_i$ which has $|V_i| - 1$ edges. Each of the at least two edges in $(J \cap A_i) \setminus T$ gives rise to a cycle, contradicting the independence of J.

So, if (V_i, A_i) is itself a tree, then $|J \cap A_i| = |V_i| - 1$ and we are done. Suppose, therefore, that (V_i, A_i) has a circuit C, and assume for contradiction that $J \cap A_i$ is a tree, and therefore has only $|V_i| - 1$ edges. $C \setminus J$ has at least one edge, say e, and adding e to J creates a unique cycle in component $(V_i, J \cap A_i)$, i.e., $J \cup e$ is independent, contradiction.

For connected component i, define $\delta_i = 1$ if (V_i, A_i) is acyclic. We just proved that

$$|J| = |V| - \sum_{i=1}^{p} \delta_i$$

for all maximal independent sets $J \subseteq A$. □

3 POLYMATROID

In this chapter, we look at polymatroids, polytopes associated with a submodular function. Our plan is as follows.

1. Motivate and define submodular functions.
2. Motivate and define of polymatroids.
3. Study the optimization problem over polymatroids.

Motivation

Let S be a set of possible sensor locations and let $f : 2^S \to \mathbb{R}$ denote the amount of information obtainable by placing sensors at locations in S. Fix a set $A \subseteq S$. The marginal gain of information by adding one sensor at location s to A is given by $f(A \cup \{s\}) - f(A)$.

Now consider another set $B \supseteq A$. Since B contains more sensors where some of them might cover the area near the new sensor s, we expect the marginal gain of information by adding s to B to be no more than that of A, i.e.,

$$A \subseteq B \implies f(A \cup \{s\}) - f(A) \geq f(B \cup \{s\}) - f(B).$$

In other words, this function f has the property of *diminishing marginal gain* and we say it is a *submodular function*.

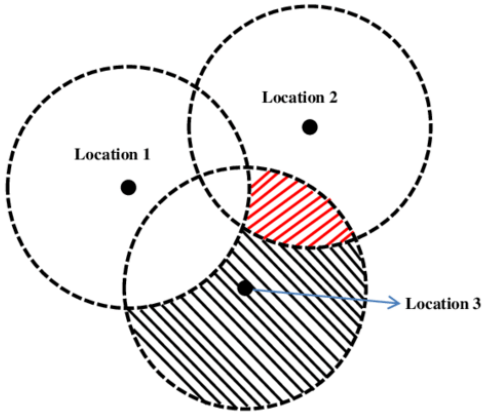


Figure 3.1: Sensor example.

In the example below, let $A = \{3\}, B = \{1, 2\}, s = 3$. Then

$$\underbrace{f(\{1,3\}) - f(\{1\})}_{\text{red + black}} \geq \underbrace{f(\{1,2,3\}) - f(\{1,2\})}_{\text{black}}.$$

3.1 Submodular Functions.

A *submodular function* is a set function whose value, informally, has the property that the difference in the incremental value of the function that a single element makes when added to an input set decreases as the size of the input set increases.

Definition 3.1. A function $f : 2^S \to \mathbb{R}$ is called **submodular** if

$$\forall A, B \subseteq S : f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

The following proposition shows this definition is equivalent to the one we gave in the motivation (last page).

Proposition 3.2. *A function $f : 2^S \to \mathbb{R}$ is submodular iff for all $A, B \subseteq S$ with $A \subseteq B$, and for all $j \in S \setminus B$, we have $f(A \cup \{j\}) - f(A) \geq f(B \cup \{j\}) - f(B)$.*

Proof. Define $\rho_j(X) = f(X \cup \{j\}) - f(X)$ for all $X \subseteq S$ and $j \in S$. We want to show that f is submodular iff $\rho_j(A) \geq \rho_j(B)$, for all $A, B \subseteq S$ with $A \subseteq B$, and for all $j \in S \setminus B$.

($\Longrightarrow$) Let $A, B \subseteq S$ with $A \subseteq B$, $j \in S \setminus B$. Since f is submodular,

$$f(A \cup \{j\}) + f(B) \geq f(A) + f(B \cup \{j\}),$$

which implies $f(A \cup \{j\}) - f(A) \geq f(B \cup \{j\}) - f(B)$ as desired.

($\Longleftarrow$) Let $A, B \subseteq S$ be arbitrary and define $A \setminus B := \{e_1, \ldots, e_p\}$. Since $\rho_j(A) \geq \rho_j(B)$,

$$\rho_{e_i}(A \cap B \cup \{e_1, \ldots, e_{i-1}\}) \geq \rho_{e_i}(B \cup \{e_1, \ldots, e_{i-1}\})$$

or equivalently,

$$f(A \cap B \cup \{e_1, \ldots, e_i\}) - f(A \cap B \cup \{e_1, \ldots, e_{i-1}\}) \geq f(B \cup \{e_1, \ldots, e_i\}) - f(B \cup \{e_1, \ldots, e_{i-1}\})$$

for all $1 \leq i \leq p$. Summing this inequality over all i yields the desired inequality

$$f(A) - f(A \cap B) \geq f(A \cup B) - f(B).$$

□

Remark. *Another characterization is given by*

$$f(A \cup \{x\}) + f(A \cup \{y\}) \geq f(A) + f(A \cup \{x\} \cup \{y\}) \quad \forall A \subseteq S, \forall x \neq y \in S \setminus A.$$

We omit the proof of correctness.

3.2 Example of Submodular Functions.

Graph Cut

Given an undirected graph $G = (V, E)$ and a non-negative capacity function $c \in \mathbb{R}_+^E$, the cut capacity function $f : 2^V \to \mathbb{R}_+$ defined by $f(S) = c(\delta(S))$ is a symmetric submodular function; $\delta(S)$ is the set of all edges in E with exactly one endpoint in S.

To see this, consider $A, B \subseteq V$ arbitrary. For simplicity, we assume a uniform $c_e = 1$ for all $e \in E$. We can categorize every edge $e \in \delta(A) \cup \delta(B)$:

1. e goes between $A \setminus (A \cap B)$ and $B \setminus (A \cap B)$, so $e \in \delta(A)$ and $e \in \delta(B)$.
2. e goes between $A \setminus (A \cap B)$ and $V \setminus (A \cup B)$, so $e \in \delta(A)$.
3. e goes between $B \setminus (A \cap B)$ and $V \setminus (A \cup B)$, so $e \in \delta(B)$.
4. e goes between $A \cap B$ and $V \setminus (A \cup B)$, so $e \in \delta(A)$ and $e \in \delta(B)$.

Now consider $f(A) + f(B)$ vs $f(A \cap B) + f(A \cup B)$:

- Every e of type 1 gets counted twice on the left but zero on the right.
- Every e of type 2 or 3 gets counted once on the left and once on the right.
- Every e of type 4 gets counted twice on the left and twice on the right.

It follows that $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$ and hence f is a submodular function.

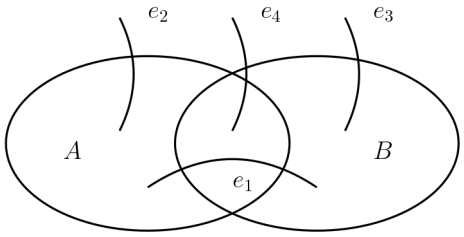


Figure 3.2: Graph cut: Edge e_i is of type i.

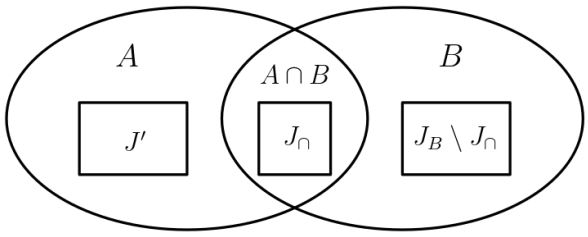


Figure 3.3: Matroid rank: Sets and bases.

Example 2: Matroid Rank

The rank function r is also submodular.

Proposition 3.3. *Let $\mathcal{M} = (S, \mathcal{I})$. Then $r(A)$ is submodular.*

Proof. Let $A, B \subseteq S$. Let $J_\cap$ be a basis of $A \cap B$,. Extend $J_\cap$ be a basis J_B of B. Extend J_B to a basis $J_\cup$ of $A \cup B$. Note that $|J_\cap| = r(A \cap B)$, $|J_B| = r(B)$, and $|J_\cup| = r(A \cup B)$. Define $J' = J_\cup \setminus (J_B \setminus J_\cap)$. It is not hard to see that $J' \in \mathcal{I}$ and $J' \subseteq A$. It follows that $r(A) + r(B) \geq |J'| + |J_B| = |J_\cup| - (|J_B| - |J_\cap|) + |J_B| = r(A \cup B) + r(A \cap B)$. □

3.3 Polymatroids.

A **halfspace** in $\mathbb{R}^n$ is a set of points $x \in \mathbb{R}^n$ that satisfy $a^T x \leq b$ for some $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$:

$$H = \{x \in \mathbb{R}^n \mid a^T x \leq b; a \in \mathbb{R}^n, b \in \mathbb{R}\}.$$

A **polyhedron** in $\mathbb{R}^n$ is a set of all points $x \in \mathbb{R}^n$ that satisfy a finite set of linear inequalities:

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b; A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m\}.$$

A **polytope** is a *bounded* polyhedron. It can be shown that every polytope P is the convex hull of a finite number of points (and vice versa).

Motivation

Let $\mathcal{M} = (S, \mathcal{I})$ be a matroid, $c \in \mathbb{R}_+^S$ be the costs, and $x : S \to \{0, 1\}$ be decision variables. Consider the following LP formulation for matroids.

$$\begin{array}{lll} (\mathrm{P_M}) := \max & c^T x & \\ s.t. & x(A) \leq r(A) & \forall A \subseteq S \\ & x \geq 0 & \end{array}$$

Claim. *x^J is feasible for this $\mathrm{P_M}$ for any $J \in \mathcal{I}$.*

Proof. By definition, $r(A)$ is the size of the maximal independent set contained in A. For any independent set $J \in \mathcal{I}$, an arbitrary set $A \subseteq S$ can contain no more than $r(A)$ element from J, i.e., $x^J(A) \leq r(A)$. □

Recall the rank function for matroids is submodular. The feasible region associated with $\mathrm{P_M}$ is an example of a *polymatroid*, a polytope associated with a submodular function.

Definition

Definition 3.4. Let $f : 2^S \to \mathbb{R}^+$ be submodular. The polyhedron given by

$$\{x \in \mathbb{R}^S \mid x(A) \leq f(A) \text{ for all } A \subseteq S; x \geq 0\}$$

is called a **polymatroid**.

The following shows that we may assume f is *normalized* (i.e., $f(\varnothing) = 0$) and *monotone*.

Lemma 3.5. *Let $g : 2^S \to \mathbb{R}_+$ be a submodular function and let $P(g)$ be the corresponding non-empty polymatroid. Show there exists a submodular and monotone function $f : 2^S \to \mathbb{R}_+$ with $f(\varnothing) = 0$ and $P(g) = P(f)$.*

Proof. Let $P := \{x \in \mathbb{R}^S \mid \forall A \subseteq S : x(A) \leq g(A); x \geq 0\}$ be a non-empty polymatroid defined by a submodular function $g : 2^S \to \mathbb{R}$ on some finite set S. Note that P is non-empty, so there exists $x \in P$ such that $0 \leq x(A) \leq g(A)$ for all $A \subseteq S$.

Define a new function $f : 2^S \to \mathbb{R}$ by

$$f(A) := \begin{cases} 0 & A = \varnothing \\ \min_{A \subseteq B \subseteq S} g(B) & A \neq \varnothing \end{cases}$$

We claim that $P(f) = P$ and f is a monotone submodular function.

Claim. *$P(f) = P$.*

Proof. Clearly, we have $P(f) \subseteq P$ as $f \leq g$. To prove that $P \subseteq P(f)$, let $x \in P$. Suppose $x \neq P(f)$, then there exists $A \subseteq S$ such that $A \neq \varnothing$ and $x(A) > f(A)$. Let $B := \arg\min_{A \subseteq B \subseteq S} g(B)$ so that $f(A) = g(B)$. But then $x(A) \leq x(B) \leq g(B) = f(A)$, a contradiction. ∎

Claim. *f is monotone.*

Proof. Let $\varnothing \neq A \subseteq B \subseteq S$. A superset of B is also a superset of A, so $g(A) \leq g(B)$. ∎

Claim. *f is submodular.*

Proof. Let $X, Y \subseteq S$ be such that $X := \arg\min_{A \subseteq X \subseteq S} g(X)$ and $Y := \arg\min_{B \subseteq Y \subseteq S} g(Y)$. Note that $A \cap B \subseteq X \cap Y$ and $A \cup B \subseteq X \cup Y$, so $f(A \cap B) + f(A \cup B) \leq g(X \cap Y) + g(X \cup Y)$. By submodularity of g, this value less than or equal to $g(X) + g(Y) = f(A) + f(B)$. ∎

The proof is complete. □

3.4 Optimization over Polymatroids.

Primal-Dual Pair

Let f be a monotone submodular function with $f(\varnothing) = 0$. Suppose we want to optimize wrt some vector $c \in \mathbb{R}^S$ (note: no ≥ 0 constraint). Consider the following LP and its dual:

$$\begin{array}{|rll|}\hline (\mathrm{P_f}) = \max & c^T x & \\ s.t. & x(A) \leq f(A) & \forall A \subseteq S \\ & x \geq 0 & \\ \hline \end{array} \qquad \begin{array}{|rll|}\hline (\mathrm{D_f}) = \min & \sum_{A \subseteq S} f(A) y_A & \\ s.t. & \sum_{A: e \in A} y_A \geq c_e & \forall e \in S \\ & y \geq 0 & \\ \hline \end{array}$$

Greedy Algorithm

Let the ground set be $S = \{e_1, \ldots, e_n\}$ such that

$$\overbrace{c_{e_1} \geq \cdots \geq c_{e_k}}^{\text{first } k \text{ elements have positive weights}} > 0 \geq \overbrace{c_{e_{k+1}} \geq \cdots \geq c_{e_n}}^{\text{the rest } n-k \text{ elements have 0 or negative weights}} .$$

Define $S_j := \{e_1, \ldots, e_j\}$, $S_0 = \varnothing$. The primal greedy algorithm will pick the solution

$$x(e_j) = \begin{cases} f(S_j) - f(S_{j-1}) & j = 1, \ldots, k \\ 0 & j > k \end{cases}$$

The dual greedy algorithm will choose

$$y(A) = \begin{cases} c(e_j) - c(e_{j+1}) \geq 0 & A = S_j, j = 1, \ldots, k-1 \\ c(e_k) & A = S_k \\ 0 & \text{otherwise} \end{cases}$$

We show that greedy approach works for polymatroid optimizations.

Theorem 3.6. *x, y produced by the primal/dual greedy algorithm are optimal for $\mathrm{P_f}$, $\mathrm{D_f}$, respectively.*

Proof. We show that x, y are feasible for $\mathrm{P_f}, \mathrm{D_f}$ and satisfy the CS conditions.

Claim. *x is feasible for $\mathrm{P_f}$.*

Proof. Let $A \subseteq S$. We show for $0 \leq j \leq k$ inductively that $x(A \cap S_j) \leq f(A \cap S_j)$. When $j = 0$, $S_j := \varnothing$, and we have $x(\varnothing) = 0 \leq 0 = f(\varnothing)$. Now suppose the statement holds for

some $j \geq 0$. If $e_j \neq A$, then

$$x(A \cap S_{j+1}) = x(A \cap S_j) \leq f(A \cap S_j) = f(A \cap S_{j+1}).$$

Otherwise, $e_j \in A$, and

$$\begin{aligned} x(A \cap S_{j+1}) &= x(A \cap S_j) + x(e_j) \\ &= x(A \cap S_j) + f(S_{j+1}) - f(S_j) \\ &\leq f(A \cap S_j) + f(S_{j+1}) - f(S_j) \\ &\leq f(A \cap S_{j+1}) \end{aligned}$$

where the last inequality follows since

$$\begin{aligned} f(A \cap S_{j+1}) + f(S_j) &\geq f((A \cap S_{j+1}) \cap S_j) + f((A \cap S_{j+1}) \cup S_j) \\ &= f(A \cap S_j) + f((A \cap S_j) \cup \{e_j\} \cup S_j)) \\ &= f(A \cap S_j) + f(S_{j+1}). \end{aligned}$$

Since f is monotone, $x(A) = x(A \cap S_k) \leq f(A \cap S_k) \leq f(A)$. Thus, x is indeed feasible for the primal LP. ■

Claim. *y is feasible for* $\mathrm{D_f}$.

Proof. Now consider the dual LP. Fix any $e_j \in S$ for $1 \leq j \leq k$ (recall $c(e_j)$ for $1 \leq j \leq k$ are the only possible non-zero ones). Then

$$\sum_{A : e_j \in A} y(A) = \sum_{S_i : e_j \in S_i} c(e_i) - c(e_{i+1}) = c(e_j).$$

Here $c(e_{n+1})$ is understood to be 0. Thus, the dual solution is feasible. Moreover, the dual constraint for $e \in S$ is tight iff $c(e) > 0$. ■

Claim. *x, y satisfies the CS conditions.*

Proof. Observe $x(e) > 0$ can only happen when $e = e_j, 1 \leq j \leq k$. But for those elements, we showed the dual constraints are tight.

Suppose now that the dual variable $y_A > 0$ for some $A \subseteq S$ is non-zero. This can only happen when $A = S_j, 1 \leq j \leq k$. We claim that $x(S_j) = f(S_j)$ for $1 \leq j \leq k$. Observe

$$x(S_j) = \sum_{i=j}^{1} x(S_j) - x(S_{j-1}) = \sum_{i=j}^{1} f(S_i) - f(S_{j-1}) = f(S_j).$$

Since x, y are feasible and satisfy CS conditions, they are optimal for the respective LPs. □

Corollary 3.7. *Let $\mathcal{M} = (S, \mathcal{I})$, $c \in \mathbb{R}^S$, and $J \in \mathcal{I}$. Then J is an inclusion-wise minimal, maximum-weight independent set iff the following hold:*

1. $e \in J \implies c_e > 0$
2. $e \notin J, J \cup \{e\} \in \mathcal{I} \implies c_e \leq 0$
3. $e \notin J, f \in J, (J \cup \{e\}) \setminus \{f\} \in \mathcal{I} \implies c_e \leq c_f$.

Proof. $\implies$: This direction is clear:

1. If $e \in J$ and $c_e \leq 0$, then we should remove e to get a better result.
2. If $e \notin J$ and $J \cup \{e\} \in \mathcal{I}$ and $c_e > 0$, then we should include it to get a better result.
3. If $e \notin J, f \in J, (J \cup \{e\}) \setminus \{f\} \in \mathcal{I}, c_e > c_f$ then we should include e instead of f in J.

$\impliedby$: Consider $\mathrm{P_r}$, the LP above with function $f = r$, the rank function of matroids. Let y be the solution of the dual greedy algorithm and let x^J be the characteristic vector of J. We argue that the these three conditions imply that x^J, y satisfy the CS conditions, which implies that x^J is of maximum weight.

By definition of the dual,

$$\sum_{A : e_j \in A} y_A = c(e_j)$$

for all $j \leq k$. Condition 1 implies that $x^J(e_j) = 0$ for all $j > k$ as they have non-positive weights so they cannot be chosen for J. Thus, for all $j = 1, \ldots, n$, either the primal variable is zero or the dual constraint is satisfied at equality:

$$x(e_j) = 0 \quad \vee \quad \sum_{A : e_j \in A} y_A = c(e_j).$$

It remains to show that $y_A = 0$ or $x^J(A) = r(A)$ for every $A \subseteq S$. Let $A \subseteq S$ by arbitrary with $y_A > 0$. By definition of y this implies $A = S_j$ for some $j \leq k$. Consider the set

$$x^J(S_j) = |J \cap S_j| = |J_j|.$$

Suppose that $|J_j| < r(S_j)$, then J_j is not a basis of S_j, so there is some $e \in S_j \setminus J$ such that $J_j \cup \{e\} \in \mathcal{I}$. Consider the set $J \cup \{e\}$.

- If $J \cup \{e\} \in \mathcal{I}$, then by statement 2, $c_e \leq 0$. But then $e \in S_j$, $j \leq k$ implies $c_e > 0$. Contradiction.
- Now suppose $J \cup \{e\} \notin \mathcal{I}$. Extend $J_j \cup \{e\}$ to a basis J' of $J \cup \{e\}$. Notice that J also is a basis of $J \cup \{e\}$ as $J \in \mathcal{I}$ and $J \cup \{e\} \notin \mathcal{I}$. (See next page for a diagram.)

Since both J' and J are bases of $J \cup \{e\}$ and we are working with a matroid, they must have the same size. Since J' contains an element e that is not in J, there must exist exactly one element $f \in J$ that is not in $J \setminus J'$.

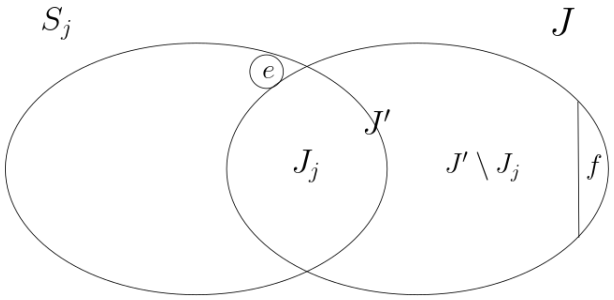


Figure 3.4: A visualization of Case 2.

By Statement 3, $J' = (J \cup \{e\}) \setminus \{f\} \in \mathcal{I}$ implies

$$c_e \leq c_f.$$

But we also had

$$y_A = y(S_j) = c(e_j) - c(e_{j+1}) > 0.$$

Recall S_j contains the j heaviest edges. Then $f \notin S_j$ implies $c(e_{j+1}) \geq c(f)$ as its weight is bounded above by the $(j+1)$-th edge weight. Since $e \in S_j$, we get

$$c(e) \geq c(e_j) > c(e_{j+1}) \geq c(f),$$

which contradicts Statement 3. Thus, we must have $|J_j| = r(S_j)$ and hence $x^J(S_j) = r(S_j)$ holds, satisfying the CS condition. It follows that x^J is optimal and hence J is a maximum weight independent set. Finally, by Statement 1, J is inclusion-wise minimal. □

Given from a matroid, we can construct new matroids using the following operations. Let $\mathcal{M} = (S, \mathcal{I})$ be a matroid.

4.1 Deletion and Truncation.

Deletion

Let $B \subseteq S$. Then $\mathcal{M} \setminus B := (S', \mathcal{I}')$ defined as below is a matroid:

$$\boxed{\begin{aligned} &\mathcal{M} \setminus B := (S', \mathcal{I}') \\ &\quad S' := S \setminus B \\ &\quad \mathcal{I} := \{A \subseteq (S \setminus B) : A \in \mathcal{I}\} \end{aligned}}$$

In words, we deleted any element in B from S and any independent set I containing any element from B from $\mathcal{I}$. In forest matroids this corresponds to deleting some edges of a graph, and in linear matroids to deleting some columns of a matrix. It is easy to see that the bases unaffected by the deletion are preserved.

Truncation

Let $k \in \mathbb{Z}$. Then $\mathcal{M}' := (S, \mathcal{I}')$ given below is a matroid:

$$\boxed{\begin{aligned} &\mathcal{M}' := (S, \mathcal{I}') \\ &\mathcal{I}' := \{A \in \mathcal{I} : |A| \leq k\} \end{aligned}}$$

In words, sets with size $\geq k$ that were independent before are no longer considered independent. To prove M3, let $A \subseteq S$ and J a maximal subset of A that is in $\mathcal{I}'$. Then if $k \geq r(A)$, then J is a basis of A in M, so $|J| = r(A)$. If $k < r(A)$, then clearly $|J| = k$. In either case, the size of J depends only on A, not on the choice of J. Moreover, we have proved that its rank function r' satisfies $r'(A) = \min\{r(A), k\}$ for all $A \subseteq S$.

Example. Truncating a graphical matroid with $k = 3$ returns a matroid where only forests with at most three edges are considered independent. ◊

4.2 *Disjoint Union.*

Let $\mathcal{M}_i = (S_i, \mathcal{I}_i)$, $i = 1, \ldots, k$, be matroids, where the ground sets are disjoint, i.e., $S_i \cap S_j = \emptyset$ for all $i \neq j$. Then $\mathcal{M}_1 \oplus \cdots \oplus \mathcal{M}_k := (S, \mathcal{I})$ given below is a matroid:

$$\begin{aligned} \mathcal{M}_1 \oplus \cdots \oplus \mathcal{M}_k &:= (S, \mathcal{I}) \\ S &:= \textstyle\bigcup_{i=1}^{k} S_i \\ \mathcal{I} &:= \{A \subseteq S \mid A \cap S_i \in \mathcal{I}_i, \forall i = 1, \ldots, k\} \end{aligned}$$

Claim. *$\mathcal{M}_1 \oplus \cdots \oplus \mathcal{M}_k$ is a matroid.*

Proof. M1 and M2 holds (easy). For M3, let B be a basis of $A \subseteq S$ and $B_i = B \cap S_i$ for all i. Then $B_i \in \mathcal{I}_i$ and in particular, it is a basis of $A \cap S_i$. (If there exists $e \in (A \cap S_i)$ such that $B_i \cup \{e\} \in \mathcal{I}_i$, then $B \cup \{e\}$ is a basis for A, contradiction.) Thus, $|B| = \sum_{i=1}^{k} r_i(A \cap S_i)$, so all bases of A have the same size, satisfying M3. □

Example. Let $S = S_1 \dot\cup \cdots \dot\cup S_k$ ("categories") and $b_1, \ldots, b_k \in \mathbb{Z}_+$ ("capacities"). We say a subset $A \subseteq S$ is independent when $|A \cap S_i| \leq b_i$ for every i. Then $\mathcal{M} = (S, \mathcal{I})$ with

$$\mathcal{I} := \{A \subseteq S : |A \cap S_i| \leq b_i, \forall i = 1, \ldots, k\}$$

is called a **partition matroid**. To verify that it's a matroid, we can write $M = M_1 \oplus \cdots \oplus M_k$ where each $M_i = (S_i, \{J \subseteq S_i : |J| \leq b_i\})$ is a uniform matroid.

A basis of a partition matroid is a set whose intersection with every block S_i has size exactly b_i (which makes it inclusion-wise maximal). A circuit of the matroid is a subset of a single block S_i with size exactly $b_i + 1$. The rank of the matroid is $\sum_i s_i$. ◊

Example. In fact, we've seen an example of a partition matroid before: 2.3.

Let $G = (V, E)$, $W \subseteq V$ a stable set.[1] For each $v \in W$, we define $k_v \in Z^+$ as an upper bound of number of edges it may incident to among a set of edges F.

Claim. *$(E, \mathcal{I})$ given by $\mathcal{I} := \{F \subseteq E : |\delta(v) \cap F| \leq k_v$ for all $v \in W\}$ is a matroid.*

Observe that the $\delta(v)$'s are disjoint because W is a stable set. Thus, we can treat

$$\{\delta(v_i)\}_{v_i \in W} \cup X,$$

where X represents the edges not included in $\delta(v)$'s, as a partition of the ground set. ◊

[1] Recall a **stable set** in graph theory refers to a set of vertices no two of which are adjacent. We used to call it "independent set" in graph theory but this term is already used in matroid theory.

4.3 Contraction.

Let $B \subseteq S$ and J be a basis of B. Then $\mathcal{M}/B = (S', \mathcal{I}')$ given below is a matroid:

$$\boxed{\begin{aligned} &\mathcal{M}/B := (S', \mathcal{I}') \\ &\quad S' = S \setminus B \\ &\quad \mathcal{I}' := \{A \subseteq (S \setminus B) : A \cup J \in \mathcal{I}\} \end{aligned}}$$

In words, given a set of elements B to be removed from S and J a basis of B, a set A is independent in the new matroid if the union of A and J were independent in the original matroid. Note the construction of new matroid does not depend on the choice of basis J.

A natural example is the forest matroid.

Proposition 4.1. *If $\mathcal{M}$ is a forest matroid of $G = (V, E)$, $B \subseteq E$, then $\mathcal{M}/B$ is a forest matroid of G/B.*

Proof. Suppose $\mathcal{M}$ is the forest matroid of G and we contract a set of edges $B \subseteq E(G)$ from G to form G'. Then the forests of G' are precisely the sets that form a forest with a forest of B in G. In this special case, $\mathcal{M}'$ is the forest matroid of G'. □

Theorem 4.2. *$\mathcal{M}/B$ is a matroid that does not depend on the choice of J, and its rank function is $r_{\mathcal{M}/B}(A) = r_{\mathcal{M}}(A \cup B) - r_{\mathcal{M}}(B)$.*

Proof. M1 and M2 are easy to show. For M3, let $A \subseteq S' := S \setminus B$. Let J' be a basis of A in the new independence system $\mathcal{M}/B$. By definition of contraction, $J \cup J' \in \mathcal{I}$. We claim that $J \cup J'$ is a basis of $A \cup B$ in matroid $\mathcal{M}$. Suppose otherwise, so there exists an element $e \in A \cup B$ such that $J \cup J' \cup \{e\} \in \mathcal{I}$. But $e \notin B$ since J is maximally independent in B and $e \notin A$ because J' is maximally independent in $\mathcal{M}/B$. It follows that $|J \cup J'| = r_{\mathcal{M}}(A \cup B)$ which implies $|J'| = r_{\mathcal{M}}(A \cup B) - |J| = r_{\mathcal{M}}(A \cup B) - r_{\mathcal{M}}(B)$ as desired. □

4.4 Duality.

$\mathcal{M}^* = (S, \mathcal{I}^*)$ given below is a matroid:

$$\boxed{\begin{aligned}\mathcal{M}^* &:= (S, \mathcal{I}^*)\\ \mathcal{I}^* &:= \{A \subseteq S : S \setminus A \text{ has a basis of } \mathcal{M}\}\\ &= \{A \subseteq S : r(S \setminus A) = r(S)\}.\end{aligned}}$$

In words, a set $A \subseteq S$ is independent in the dual matroid if the set $S \setminus A$ still has a basis of $\mathcal{M}$. Equivalently, every subset A of S such that the rank of $S \setminus A$ is exactly the rank of the original matroid is independent.

Example. Consider the uniform matroid $\mathcal{M} = U_n^r$ with $A \subseteq S = \{1, \ldots, n\}$. So when does $A \in \mathcal{I}^*$?. By definition, $A \in \mathcal{I}^* \iff S \setminus A$ has a basis of $\mathcal{M}$. A basis of $\mathcal{M}$ is any subset of S with exactly r elements. Thus, as long as $|A| \leq n - r$, we are still left with a basis. Therefore, the dual matroid is given by $\mathcal{M}^* = U_n^{n-r}$. ◊

Theorem 4.3. *$\mathcal{M}^*$ is a matroid with rank fn $r^*(A) = |A| + r_{\mathcal{M}}(S \setminus A) - r_{\mathcal{M}}(S)$.*

Proof. M1 and M2 are easy to show. For M3, let $A \subseteq S$, J^* be a $\mathcal{M}^*$-basis of A, and J be a $\mathcal{M}$-basis of $S \setminus A$. Extend J to an $\mathcal{M}$-basis J' of $S \setminus J^*$.

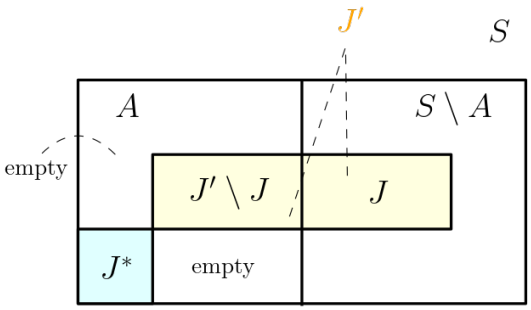


Figure 4.1: A visualization of the sets and bases.

By definition of contraction, $J^* \in \mathcal{I}^*$ implies that $r(S \setminus J^*) = r(S)$, so J' is an $\mathcal{M}$-basis of S. We now show that $A \setminus J^* \subseteq J'$, i.e., there's nothing left in the left half of the diagram above outside of J^* and $J' \setminus J$.

Suppose otherwise that there exists $e \in (A \setminus J^*) \setminus J'$. Since $J' \subseteq S \setminus (J^* \cup \{e\})$ and J' is an $\mathcal{M}$-basis of S, $J^* \cup \{e\} \in \mathcal{I}^*$ by definition of contraction. This contradicts the fact that J^* was an $\mathcal{M}^*$-basis of A. It follows that $|J'| = |A \setminus J^*| + |J| = |A| - |J^*| + |J|$ which implies

$$|J^*| = |A| - |J'| + |J| = |A| - r_{\mathcal{M}}(S) + r_{\mathcal{M}}(S \setminus A).$$

□

Example. Consider the forest matroid $\mathscr{M}$ of a planar graph G. Let $\mathscr{M}^*$ be the dual matroid. Observe

$$\begin{aligned} A \in \mathcal{I} &\iff r(E \setminus A) = r_E \\ &\iff G[E \setminus A] = (V, E \setminus A) \text{ has a spanning tree} \\ &\iff V(E \setminus A) \text{ is connected} \end{aligned}$$

Cycles in G^* correspond to edge sets $\delta(S)$ in G. Thus, the minimal dependent sets in $\mathscr{M}^*$ are cycles and $\mathscr{M}^*$ is the forest matroid for G^*. ◊

Remark. You should read this chapter after studying all materials on matchings.

5.1 Matroid Intersection.

Problem. *Let $\mathcal{M}_1 = (S, \mathcal{I}_1)$ and $\mathcal{M}_2 = (S, \mathcal{I}_2)$ be two matroids over the same ground set. The* ***matroid intersection problem*** *comes in two versions:*

- *Unweighted: Find $A \in \mathcal{I}_1 \cap \mathcal{I}_2$ maximizing $|A|$.*
- *Weighted: Given $c \in \mathbb{R}_+^S$, find $A \in \mathcal{I}_1 \cap \mathcal{I}_2$ maximizing $c(A)$.*

Motivation

Let $G = (V, E)$ be bipartite with bipartition V_1, V_2. Define

$$\mathcal{I}_1 = \{A \subseteq E : |A \cap \delta(v)| \leq 1, \forall v \in V_1\}$$
$$\mathcal{I}_2 = \{A \subseteq E : |A \cap \delta(v)| \leq 1, \forall v \in V_2\}$$

Then $(E, \mathcal{I}_1)$ and $(E, \mathcal{I}_2)$ are partition matroids. Review 4.2 if necessary.

Observe the problem of finding $A \in \mathcal{I}_1 \cap \mathcal{I}_2$ maximizing $|A|$ is exactly the maximum cardinality matching problem on bipartite graphs.

Min-Max Theorem

Let $J \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a feasible solution to the matroid intersection problem and let $A \subseteq S$ be arbitrary. Our goal is to give an upper bound for the size of J based on some function of A. We do this by splitting J into an $\mathcal{M}_1$ component and an $\mathcal{M}_2$ component.

A natural bound is given by

$$|J| = \overbrace{|J \cap A|}^{\in \mathcal{I}_1} + \overbrace{|J \cap \bar{A}|}^{\in \mathcal{I}_2} \leq r_1(A) + r_2(\bar{A}).$$

Edmonds showed in 1971 that this inequality is actually an equality.

Theorem 5.1 (Edmonds)**.** *Let $M_i = (S, \mathcal{I}_i), i = 1, 2$ be matroids. Then*

$$\max\{|J| : J \in \mathcal{I}_1 \cap \mathcal{I}_2\} = \min\{r_1(A) + r_2(\bar{A}) : A \subseteq S\}.$$

Proof. Skipped as this does not add much to our understanding.

Example

Let $G = (V, E)$ be bipartite with bipartition V_1, V_2. Define

$$\mathcal{I}_1 = \{A \subseteq E : |A \cap \delta(v)| \leq 1, \forall v \in V_1\}$$
$$\mathcal{I}_2 = \{A \subseteq E : |A \cap \delta(v)| \leq 1, \forall v \in V_2\}$$

Let M be a maximum cardinality matching of G. By the matroid intersection theorem, there exists $A \subseteq E$ achieves the minimum, i.e.,

$$\max\{|J| : J \in \mathcal{I}_1 \cap \mathcal{I}_2\} = |M| = r_1(A) + r_2(E \setminus A) = \min\{r_1(A) + r_2(E \setminus A) : A \subseteq E\}.$$

Suppose we are given the following bipartite graph. Let A be the set of red edges.

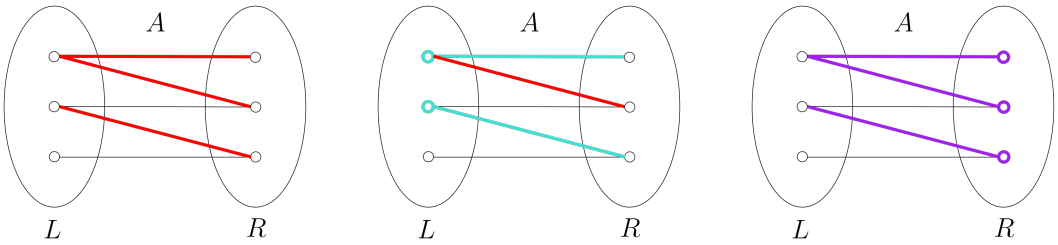


Figure 5.1: Left: A. Middle: B_1 and U_1. Right: B_2 and U_2.

Let B_1 be a $\mathcal{M}_1$-basis of A and B_2 be a $\mathcal{M}_2$-basis of A. Let $U_i = B_i \cap V_i, i = 1, 2$.

Claim. *$|U_1| = r_1(A)$, $|U_2| = r_2(E \setminus A)$, $U_1 \cup U_2$ is a vertex cover of G, i.e., any edge in G is incident with at least one vertex in $U_1 \cup U_2$.*

Proof. Observe that by definition, every vertex of $A \cap V_1$ is incident to at most one vertex of B_1. But every vertex of $A \cap V_1$ is incident to at least one edge by maximal independence. Thus, $|U_1| = |B_1| = r_1(A)$. A similar argument shows $|U_2| = |B_2| = r_2(E \setminus A)$.

Next, every edge e either lives in A or $E \setminus A$. In the case of the former, a vertex in U_1 is incident to it; in the case of the latter, a vertex in U_2 is incident to it. Thus, $U_1 \cup U_2$ is a vertex cover. □

Note this result implies **Konig's Theorem**: $\nu(G) = \tau(G)$ when G is bipartite.

5.2 *Matroid Intersection Algorithm.*

We can derive an algorithm for the matroid intersection problem by exploring its connection with matchings in bipartite graphs. Recall a matching M is not maximum if we can find an M-augmenting path. The same idea applies here.

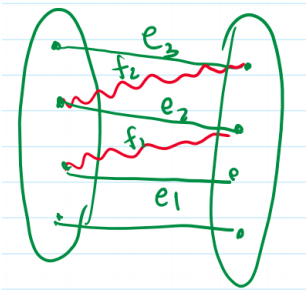


Figure 5.2: Example.

Let J be a matching and $P = e_1, f_1, e_2, f_2, e_3$ be an augmenting path, where $e_i \neq J$ and $f_i \in J$. Denote the two ends of P by v_1, v_6 (bottom-right, top-left). Let us describe this path in matroid languages. First, since e_1 is J-exposed and no edges in J is touching v_1, we have $J \cup \{e_1\} \in \mathcal{I}_2$. Similarly, we have $J \cup \{e_3\} \in \mathcal{I}_1$. Next, the augmenting path is given by taking the symmetric differences between M and P. Observe we have

- $J \cup \{e_i\} \setminus \{f_i\} \in \mathcal{I}_1$;
- $J \cup \{e_{i+1}\} \setminus \{f_i\} \in \mathcal{I}_2$.

Let's formalize these. Denote the six conditions below by $\star$.

Definition 5.2. Let $J \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $P = \{e_1, f_1, \ldots, e_m, f_m, e_{m+1}\}$ such that

- $\forall i : e_i \notin J$
- $\forall i : f_i \in J$
- $J \cup \{e_1\} \in \mathcal{I}_2$
- $J \cup \{e_{m+1}\} \in \mathcal{I}_1$
- $\forall i : J \cup \{e_i\} \setminus \{f_i\} \in \mathcal{I}_1$
- $\forall i : J \cup \{e_{i+1}\} \setminus \{f_i\} \in \mathcal{I}_2$

Define $J' = J \triangle P = J \cup \{e_1, \ldots, e_{m+1}\} \setminus \{f_1, \ldots, f_m\}$.

Lemma 5.3. *If P is the smallest subset of S satisfying $\star$, then $J' \in \mathcal{I}_1 \cap \mathcal{I}_2$.*

Proof. We claim that $J \cup \{e_i\} \notin \mathcal{I}_1$ for all $i = 1, \ldots, m$. Suppose not. Then $\{e_1, f_1, \ldots, e_{i-1}, f_{i-1}, e_i\}$ also satisfies $\star$ and is a smaller set than P. Contradiction.

Now pick $A = J \cup \{e_1, \ldots, e_{m+1}\}$ and define $A_i := A \setminus \{f_m, \ldots, f_i\}$. In words, A_i is the set obtained by removing the last i elements that were previously in J from A, e.g., $A_{m+1} = A$ (we haven't removed any element from $\{f_i\}_{i=1}^m$ yet) and $A_1 = J'$ (we remove all elements from $\{f_i\}_{i=1}^m$). Let C_i be the $\mathcal{M}_i$-circuit in $J \cup \{e_i\}$ for all $i = 1, \ldots, m$, which exists by the previous claim. We claim that $C_i \subseteq A_{i+1} = A \setminus \{f_m, \ldots, f_{i+1}\}$.

Suppose not, i.e., some f_k with $i+1 \leq k \leq m$ is in C_i. Then removing f_k from C_i gives an independent set in $\mathcal{M}_1$. In other words, there is $k > i$ such that $J \cup \{e_i\} \setminus \{f_k\} \in \mathcal{I}_1$. But then $\{e_1, f_1, \ldots, e_i, f_k, e_{k+1}, \ldots, f_m, e_{m+1}\}$ also satisfies $\star$. Contradiction. Thus, $C_i \subseteq A_{i+1}$.

Since $C_i \subseteq A_{i+1} = A_i \cup \{f_i\}$, we have $C_i \setminus \{f_i\} \subseteq A_i$. By $\star$, $C_i \setminus \{f_i\} \in \mathcal{I}_1$, so we can extend it to an $\mathcal{M}_1$-basis of A_i, call it B_i. But $B_i \supseteq (C_i \setminus \{f_i\})$ means $(B_i \cup \{f_i\}) \supseteq C_i$. Moreover, $B_i \subseteq A_i \subseteq A_{i+1} = A_i \cup \{f_i\}$. Then B_i is an $\mathcal{M}_1$-basis of A_{i+1}, because if it's not, then the only extra element from A_{i+1} compared to A_i is f_i but $(B_i \cup \{f_i\}) \supseteq C_i$ is dependent.

Therefore, $r_1(A_i) = r_1(A_{i+1})$. Since this holds for all i, we have $r_1(J') = r_1(A_1) = r_1(A_{m+1}) = r_1(A)$. By $\star$, $J \cup \{e_{m+1}\} \in \mathcal{I}_1$ so $r_1(A) \geq |J| + 1 = |J'|$. But $r_1(A) = r_1(J') \leq |J'|$. Thus, $J' \in \mathcal{I}_1$.

A symmetric argument shows that $\mathcal{I}' \in \mathcal{I}_2$. □

Augmenting Paths

So how exactly do we find such an "augmenting path" P? We can model this with a DAG. First, recall the following definition.

Definition 5.4. Let $J \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $P = \{e_1, f_1, \ldots, e_m, f_m, e_{m+1}\}$ such that

- $\star_1$ $\forall i : e_i \notin J$
- $\star_2$ $\forall i : f_i \in J$
- $\star_3$ $J \cup \{e_1\} \in \mathcal{I}_2$
- $\star_4$ $J \cup \{e_{m+1}\} \in \mathcal{I}_1$
- $\star_5$ $\forall i : J \cup \{e_i\} \setminus \{f_i\} \in \mathcal{I}_1$
- $\star_6$ $\forall i : J \cup \{e_{i+1}\} \setminus \{f_i\} \in \mathcal{I}_2$

Define $J' = J \triangle P = J \cup \{e_1, \ldots, e_{m+1}\} \setminus \{f_1, \ldots, f_m\}$.

Given $S = \{x_1, \ldots, x_m\}$ (think: edges) and $J \in \mathcal{I}_1 \cap \mathcal{I}_2$ (think: a matching),[1] define the node set $N = S \cup \{r, s\}$, where r is the source and s is the sink, and the arc set A as follows.

P1. For each x_i such that $x_i \notin J$ and $J \cup \{x_i\} \in \mathcal{I}_2$, add an arc rx_i.

P2. For each x_j such that $x_j \notin J$ and $J \cup \{x_j\} \in \mathcal{I}_1$, add an arc $x_j s$.

P3. For $f, e \in S$, if $f \in J$, $e \notin J$, $J \cup \{e\} \notin \mathcal{I}_2$, and $J \cup \{e\} \setminus \{f\} \in \mathcal{I}_2$, add an arc fe.

P4. For $e, f \in S$, if $e \notin J$, $f \in J$, $J \cup \{e\} \notin \mathcal{I}_1$, and $J \cup \{e\} \setminus \{f\} \in \mathcal{I}_1$, add an arc ef.

[1] In the definition above, e_i and f_j's are "edges", but here the x_i's are "nodes". Just be careful.

Lemma 5.5. *The r, s-alternating paths in this DAG correspond precisely to the "augmenting paths" satisfying the $\star$ conditions.*

Proof. Let P' be a r, s-alternating dipath where $N(P) = \{r, x_1, x_2, x_3, s\}$. This corresponds to an augmenting path $P = \{x_1, x_2, x_3\}$. Align the terms, we see that $x_1 \mapsto e_1, x_2 \mapsto f_1, x_3 \mapsto e_2$. Let us verify that P satisfies all six $\star$s above. By our construction:

- $rx_1 \equiv re_1 \in A(P) \implies e_1 \notin J, J \cup \{e_1\} \in \mathcal{I}_2$, i.e., $\star_1$ and $\star_3$ are satisfied;
- $x_3 s \equiv e_2 s \in A(P) \implies e_2 \notin J, J \cup \{e_2\} \in \mathcal{I}_1$, i.e., $\star_1$ and $\star_4$ are satisfied;
- $x_1 x_2 \equiv e_1 f_1 \in A(P) \implies e_1 \notin J, f_1 \in J, J \cup \{e_1\} \notin \mathcal{I}_1$, and $J \cup \{e_1\} \setminus \{f_1\} \in \mathcal{I}_1$, i.e., $\star_1$, $\star_2$, $\star_5$ are satisfied;
- $x_2 x_3 \equiv f_1 e_2 \in A(P) \implies e_2 \notin J, f_1 \in J, J \cup \{e_2\} \notin \mathcal{I}_2$, and $J \cup \{e_2\} \setminus \{f_1\} \in \mathcal{I}_2$, i.e., $\star_1, \star_2, \star_6$ are satisfied.

It follows that r, s-alternating paths correspond precisely to paths satisfying the $\star$ conditions. In addition, we can simply find the shortest such path to get an inclusion-wise minimal path. □

Lemma 5.6. *If there is no r, s-alternating paths, which implies there are no augmenting paths P satisfying $\star$, then J is a maximum cardinality element of $\mathcal{I}_1 \cap \mathcal{I}_2$.*

Proof. Let $U \subseteq S$ be the set of elements reachable from r via dipaths. If $U = \emptyset$, by P1, J is an inclusion-wise maximal element of $\mathcal{I}_2$, so it is the max cardinality element in $\mathcal{I}_1 \cap \mathcal{I}_2$.

Otherwise, there is some $e \in U \setminus J$. Indeed, by P1, there is an arc re_i for which $e_i \notin J$ and $J \cup \{e_i\} \in \mathcal{I}_2$. Since es is not an arc (or we get an r, s-alternating path), $J \cup \{e\} \notin \mathcal{I}_1$ by P2.

Let C be the $\mathcal{M}_1$-circuit of $J \cup \{e\}$. Since there is no ef with $f \in S \setminus U$ (or f is reachable from r and hence will be contained in U), we must have

$$C \subseteq U \implies C \subseteq (J \cap U) \cup \{e\}..$$

Note that $J \cap U \in \mathcal{I}_1$, so $J \cap U$ is an $\mathcal{M}_1$-basis of U. (Need clean up: (?))

Apply the same argument to show that $J \cap \overline{U}$ is an $\mathcal{M}_2$-basis of J. It follows that

$$|J| = |J \cap U| + |J \cap \overline{U}| = r_1(U) + r_2(\overline{U})$$

and $|J|$ attains the upper bound. □

This leads to a polynomial-time algorithm for solving the matroid intersection problem, assuming checking independence can also be done in polynomial time. The weighted version can also be solved in polynomial time.

LP Approach

Note the problem can also be solved using the following LP:

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & x(A) \leq r_1(A) \quad \forall A \subseteq S \\ & x(A) \leq r_2(A) \quad \forall A \subseteq S \\ & x \geq 0 \end{aligned}$$

5.3 *Matroid Intersection with Three Matroids.*

The matroid intersection problem with three matroids is NP-hard. That is, given matroids $\mathcal{M}_i = (S, \mathcal{I}_i)$ for $i = 1, 2, 3$, finding the largest cardinality element of $\mathcal{I}_1 \cap \mathcal{I}_2 \cap \mathcal{I}_3$ is NP-hard.

Consider the NP-hard directed Hamiltonian path problem: Given a digraph $D = (V, A)$ and r, s distinct vertices in V for which there exists an r, s-dipath in D. Determine if there is an r, s-dipath P in D such that every vertex in $V \setminus \{r, s\}$ is visited exactly once, i.e., $(|\delta^+(v)| = |\delta^-(v)| = 1)$.

By definition, any Hamiltonian path P from r to s satisfies the following conditions:

H1. No arc going into source r; exactly one edge going into every other $v \in V \setminus \{r\}$.
H2. No arc leaving the sink s; exactly one edge leaving every other $v \in V \setminus \{r\}$.
H3. Each vertex $v \in V$ is used exactly once in P.

We can model these three conditions with three matroids $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ as follows:

Let $\mathcal{M}_1 = (E, \mathcal{I}_1)$ be the partition matroid with independent sets being subsets of E with 0 outgoing edge from sink s and at most 1 outgoing edge from every other vertex $v \in V \setminus \{s\}$.

Let $\mathcal{M}_2 = (E, \mathcal{I}_2)$ be the partition matroid with independent sts being subsets of E with 0 edge going into source r and at most 1 edge going into each other vertex $v \in V \setminus \{r\}$.

Let $\mathcal{M}_3 = (E, \mathcal{I}_3)$ be the forest matroid of the underlying undirected graph G of D, i.e., a set $F \subseteq E$ is independent in $\mathcal{M}_3$ iff F contains no cycles.

Consider the following algorithm, which takes in a digraph $D = (V, A)$ and two distinct vertices $r, s \in V$, and outputs **True** if there is an r, s Hamiltonian path and **False** otherwise:

1. Define $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ as above.
2. Let I be the largest cardinality element in $\mathcal{I}_1 \cap \mathcal{I}_2 \cap \mathcal{I}_3$.
3. Return **True** if $|I| = |V| - 1$ and **False** otherwise.

Lemma 5.7. *This algorithm solves the directed Hamiltonian path problem.*

Proof. Let P be a directed Hamiltonian path from r to s. Then P satisfies H1, H2, H3, so it is in $\mathcal{I}_1 \cap \mathcal{I}_2 \cap \mathcal{I}_3$. Moreover, it satisfies the conditions for $\mathcal{I}_1$ and $\mathcal{I}_2$ at equality (always having 1 edge going into each $v \in V \setminus \{r\}$ and always having 1 edge leaving each $v \in V \setminus \{s\}$), so it is a basis for $\mathcal{I}_1$ and $\mathcal{I}_2$. It follows that P is a maximum cardinality set in $\mathcal{I}_1 \cap \mathcal{I}_2 \cap \mathcal{I}_3$ with $|P| = |V| - 1$. So the algorithm returns True.

Now suppose the algorithm returns True, i.e., there exists some $I \in \mathcal{I}_1 \cap \mathcal{I}_2 \cap \mathcal{I}_3$ with $|I| = |V| - 1$. We claim it is a Hamiltonian path. Indeed, since $I \in \mathcal{I}_1$ and $|I| = |V| - 1$, it

must contains exactly one edge going into each $v \in V \setminus \{r\}$. Similarly, it contains exactly one edge leaving each $v \in V \setminus \{s\}$. It cannot contains any cycle because $I \in \mathcal{I}_3$. By definition, I is a Hamiltonian dipath from r to s as required. □

Proposition 5.8. *The matroid intersection problem for 3 matroids is NP-hard.*

Proof. Suppose not. Then we can use the algorithm above to solve the Hamiltonian path problem in polynomial time, which contradicts the fact that Hamiltonian path is NP-hard. The result follows. □

5.4 *Matroid Partitioning.*

Definition 5.9. Let $\mathcal{M}_i = (S, \mathcal{I}_i)$, $1 \le i \le k$ be matroids. We say that $J \subseteq S$ is **partitionable** if $J = J_1 \dot\cup \cdots \dot\cup J_k$ with $J_i \in \mathcal{I}_i$, $1 \le i \le k$.

Theorem 5.10 (Matroid Partitioning; Edmonds, Fulkerson)**.**

$$\max\{|J| : J \text{ is partitionable}\} = \min_{A \subseteq S} \left\{ |\bar{A}| + \sum_{i=1}^{k} r_i(A) \right\}.$$

Proof. Write the ground set as $S = \{e_1, \dots, e_n\}$. Let S^i be a copy of S for each $1 \le i \le k$, i.e.,

$$\forall i = 1, \dots, k : S^i := \{e_1^i, \dots, e_n^i\}.$$

For $J \subseteq \bigcup_{i=1}^k S^i$, let J^0 be the corresponding set of elements in S, i.e.,

$$J^0 := \{e \in S \mid \exists i \in \{1, \dots, k\} : e^i \in J\}.$$

For each i, define $\mathcal{M}_i' := (S^i, \{J \subseteq S^i \mid J^0 \in \mathcal{I}_i\})$, i.e., a copy of the original matroid but over the ground set S^i. Define $N_a := M_1' \oplus \cdots \oplus M_k'$. Define another matroid $N_b = (S', \mathcal{I}_b)$ with

$$\begin{aligned} S' &:= \textstyle\bigcup_{i=1}^k S_i \\ \mathcal{I}_b &:= \{A \subseteq S' \mid \forall e \in S : \ A \text{ has at most one copy of } e\}. \end{aligned}$$

The intuition is that since we can only pick at most one copy of each element e, if we picked the j-th copy, e^j, then that element is from $\mathcal{M}_j$. This leads to the observation that J is independent in both N_a and N_b iff J^0 is partitionable. Moreover, in such case, $|J| = |J^0|$. It follows by the matroid intersection theorem that

$$\max\{|J^0| : J^0 \text{ is partitionable}\} = \min_{B \subseteq S'} \{r_a(B) + r_b(S' \setminus B)\}.$$

We claim that a minimizer B is of the form $\bigcup_{e \in B^0} \{e^1, \dots, e^k\}$. Suppose there is some $e^j \in B$ and let $e^k \in S' \setminus B$. Consider $B' := B \setminus \{e^j\}$. Let D be a $\mathcal{M}_b$-basis of $S' \setminus B$ and notice that $D \subseteq S' \setminus B'$. If $D \cup \{e^j\} \in \mathcal{I}_b$, then $D \cup \{e^k\} \in \mathcal{I}_b$ as well, which contradicts the maximality of D within $S' \setminus B$. It follows that D is also a basis of $S' \setminus B'$ and $r_b(S' \setminus B') = r_b(S' \setminus B)$. Moreover, $r_a(B') \le r_a(B)$, so B' is also a minimizer. With this claim, we get

$$r_a(B) = \sum_{i=1}^{k} r_i(B^0) \quad \text{and} \quad r_b(S' \setminus B) = |S \setminus B^0|,$$

so the result holds. □

Part III

Matchings

6 MATCHING

In this chapter, we will study machines and algorithms for the respective optimization problems. Our plan is as follows.

1. Basic definitions and maximum cardinality matchings.
2. Perfect matching in bipartite graphs: properties and algorithms.
3. Perfect matching in general graphs: properties and algorithms.

Recall the following definitions and properties from CO-342.

Definition 6.1. Given a graph $G = (V, E)$, a subset $M \subseteq E$ is a **matching** if every vertex is incident with at most one edge in M, i.e.,

$$\forall v \in V : |\delta(v) \cap M| \leq 1.$$

Definition 6.2. Given a matching M, a vertex v is said to be M**-covered** if $|\delta(v) \cap M| = 1$ and M**-exposed** otherwise.

Lemma 6.3. *Given $G = (V, E)$ and a matching M, there are $2|M|$ vertices that are M-covered and $|V| - 2|M|$ vertices that are M-exposed.*

Definition 6.4. A matching M is **perfect** if there are no M-exposed vertices.

Definition 6.5. The size of the maximum matching in G is denoted $\nu(G)$.

Definition 6.6. We say $u \in V$ is **essential** if u is M-covered in *every* maximum matching M. Otherwise, we say $u \in V$ is **inessential**.

Lemma 6.7. *M is a perfect matching iff $\nu(G) = |V(G)|/2$.*

Definition 6.8. Given $G = (V, E)$ and a matching M, a path $P = (v_1, \ldots, v_k)$ is called M**-alternating** if $\forall i = 2, \ldots, k-1 : v_{i-1}v_i \in M \iff v_iv_{i+1} \notin M$.

Definition 6.9. An M-alternating path is M**-augmenting** if v_1, v_k are exposed.

6.1 Maximum Matchings.

Symmetric Difference

Definition 6.10. Given $F_1, F_2 \subseteq E$, the **symmetric difference** between F_1 and F_2, denoted as $F_1 \triangle F_2$, is defined as

$$F_1 \triangle F_2 := \{e \in E : \ e \text{ is in exactly one of } F_1, F_2\}.$$

Alternating Paths

Theorem 6.11 (Berge)**.** *Let M be a matching of $G = (V, E)$. Then M is a maximum matching iff there does not exist an M-augmenting path.*

Proof. If P is an M-augmenting path, then $M' := M \triangle (E(P))$ is a strictly larger matching. Now suppose M' is a matching of G with $|M'| > |M|$. Consider $G' = (V, M \triangle M')$. Since each v can incident to at most one edge from each of M and M', we have $|\delta_{G'}(v)| \leq 2$ for all $v \in V$. Thus, G' is a disjoint union of paths and cycles where all of them are alternating wrt both M and M'. Since $|M'| > |M|$, at least one of the components of G' has more edges from M'. Since M and M' would contribute an equal number of edges if this component were a cycle, we conclude that this component must be a path; let's call it P. Moreover, the first and last edge of P must come from M', so P is our desired M-augmenting path. □

Alternating Trees

Recall we could run BFS to check whether there exists a u, v-path in a graph. We can use the same idea to check whether there exists an *M-alternating path* from a *M-exposed vertex u* to an *M-exposed vertex v* as follows: Start at an M-exposed vertex r, compute a BFS tree where odd-distanced vertices were "discovered" through non-matching edges while even-distanced vertices were "discovered" through matching edges.

1. If we "discovered" an M-covered vertex, then we can extend the tree by two vertices.
2. Otherwise, if we "discovered" an M-exposed vertex v, we can augment our matching using the newly found r, v-path and initialize our tree from another exposed vertex.

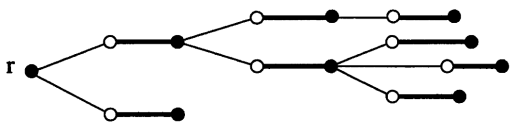


Figure 6.1: An alternating tree.

This idea almost works.

6.2 *Perfect Matchings in Bipartite Graphs.*

Hall's Theorem

For $X \subseteq V$. The neighbour set $N(X)$ is given by $\{v \in V \setminus X : \exists u \in X, uv \in E\}$.

Definition 6.12 (Hall)**.** Let $G = (V, E)$ be bipartite with bipartition $V = A \dot\cup B$. There exists a matching covering A iff $|N(X)| \geq |X|$ for all $X \subseteq A$.

Proof. ($\neg \Longleftarrow \neg$): This is a necessary condition: If there exists $X \subseteq A$ such that $|X| > |N(X)|$, then no matching can cover all vertices in X.

($\Longleftarrow$): Prove by induction on $|A|$. If $|A| \leq 1$, the result holds trivially. Suppose that for every non-trivial subset $X \subseteq A$, $|N(X)| > |X|$. Pick $uv \in E$ with $u \in A, v \in B$ and consider $G' := G - \{u, v\}$. This is a bipartition $A' := A - u$ and $B' := B - v$. For all $X \subseteq A'$, $|N_{G'}(X)| \geq |N_G(X)| - 1 \geq |X|$, so Hall's conditions holds for A' in G'. By induction, there is a matching M' covering A'. It follows that $M' + uv$ covers A.

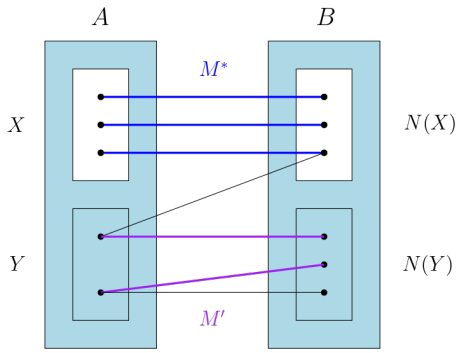


Figure 6.2: Hall's theorem, Case 2 below. (G' is highlighted.)

Otherwise, there is some non-trivial $X \subseteq A$ such that $|N(X)| = |X|$. By induction, there is a matching M^* in $G[X \cup N(X)]$ covering X, since Hall's condition holds for the subgraph. Consider $G' := G[(A \setminus X) \cup (B \setminus N(X))]$. We wish to argue that Hall's condition still holds. Indeed, for any $Y \subseteq A \setminus X$,

$$\begin{aligned} |N_{G'}(Y)| &= |N_G(Y) \setminus N_G(X)| \\ &= |N_G(X \cup Y)| - |N_G(X)| \\ &= |N_G(X \cup Y)| - |X| \\ &\geq |X \cup Y| - |X| = |Y|. \end{aligned}$$

Thus, there exists a matching M' in G' covering $A \setminus X$. Combining M^* and M', we found a matching in G covering A. □

It is easy to see that if the bipartition of a graph satisfies Hall's conditions and have the same size, then the graph has a perfect matching.

Corollary 6.13. *Let $G = (V, E)$ be bipartite with bipartition $V = A \dot\cup B$. Then G has a perfect matching iff $|A| = |B|$ and $|X| \leq |N(X)|$ for all $X \subseteq A$.*

Proof. Omitted. □

Algorithm for Finding a Perfect Matching in Bipartite Graphs

Algorithm 6: Algorithm for Finding a Perfect Matching in Bipartite Graphs

input: $G = (V, E)$ bipartite
output: A perfect matching M of G or None if G does not have a perfect matching
main:

Initialize matching $M \leftarrow \emptyset$ and let $r \in V$
Initialize $T \leftarrow (V(T) = \{r\}, E(T) = \emptyset)$
$A \leftarrow \emptyset, B \leftarrow \{r\}$, the odd-distanced and even-distanced vertices, respectively
while $\exists vw \in E : v \in B(T), w \notin V(T)$ **do**
    if w is M-covered **then** ▷ Case 1 below.
        Use vw to extend T
    else ▷ w is M-exposed; Case 2 below.
        Use vw to augment M
        if $\exists M$-exposed vertex $r' \in V$ **then** ▷ Restate the algorithm
            $T \leftarrow (V(T) = \{r'\}, E(T) = \emptyset)$
            $A \leftarrow \emptyset, B \leftarrow \{r'\}$
        else
            return M ▷ No M-exposed vertex left $\implies$ M is a PM.
return None

Edges in M are marked red:

Figure 6.3: Case 1. Extend the tree by 2 edges.

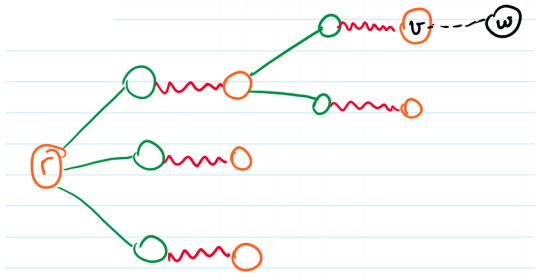


Figure 6.4: Case 2. rw is an M-augmenting path!

Claim. *If the algorithm reaches the last line, then G has no perfect matching.*

Proof. Let T, B, A be the state of the algorithm just before termination. By definition, A, B contain the odd-distanced and even-distanced vertices (from r) in G. We have the following observations:

1. $N(B(T)) = A(T)$: Every neighbour w of $v \in B(T)$ was not present in T when it was added to T. Thus, all of them have an odd distance from r and thus in $A(T)$.
2. $|B(T)| > |A(T)|$: There is exactly one more element in $B(T)$ than $A(T)$, namely the root r of the tree.

Since (A, B) is a bipartition of G but $|A| < |B| = |N(A)|$, by Corollary 6.13, G does not have a perfect matching. □

Claim. *The algorithm runs in polynomial time.*

Proof. We can augment the matching at most $n/2$ times (that's the size of a maximum matching of a graph with n vertices), with each augmentation requiring linear time to compute the alternating tree. □

Example: Algorithm for Finding a Perfect Matching in Bipartite Graphs

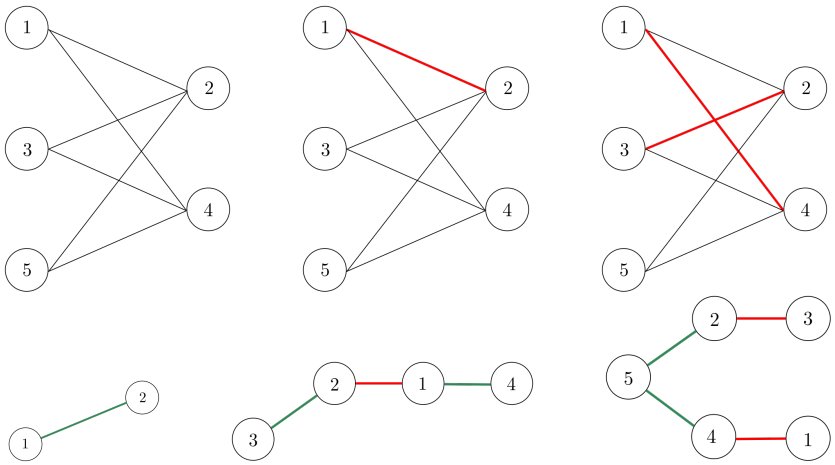


Figure 6.5: Top: Graph and matching M (in red). Bottom: M-alternating tree.

Start with the bipartite graph on the top left. Pick $r = v_1$. Discover v_2 where $v_2 \notin T$ and is M-exposed. Augmenting the matching with v_1v_2. Now pick $r = v_3$. Discover v_2 where $v_2 \notin T$ and is M-covered. Discover an M-augmenting path v_3, v_2, v_1, v_4. Augment M with this path. Now pick $r = v_5$. Discover v_2 and v_4 and extend the tree. Observe there is no other neighbours of v_3 and v_1 that are M-exposed. Stop and return this matching. ◊

6.3 *The Tutte-Berge Formula.*

Vertex Covers

Definition 6.14. A set of vertices $U \subseteq V$ is a **vertex cover** of $G = (V, E)$ if every edge in E has at least one ends in U, i.e., $\forall e \in E : |e \cap U| \geq 1$.

The size of a minimum vertex cover of G is denoted $\tau(G)$.

Lemma 6.15. *For any graph G, $\nu(G) \leq \tau(G)$.*

Theorem 6.16 (Konig)**.** *If G is bipartite, then $\nu(G) = \tau(G)$.*

Odd Components

Motivation. ***Our goal is to find an upper bound for the size of any matching M of G.***

Let $A \subseteq V$ and $H_1, \ldots, H_k$ be the connected components with an odd number of vertices in $G - A$. We call them **odd components** and denote the number of odd components in $G - A$ by $k = \text{odd}(G - A)$.

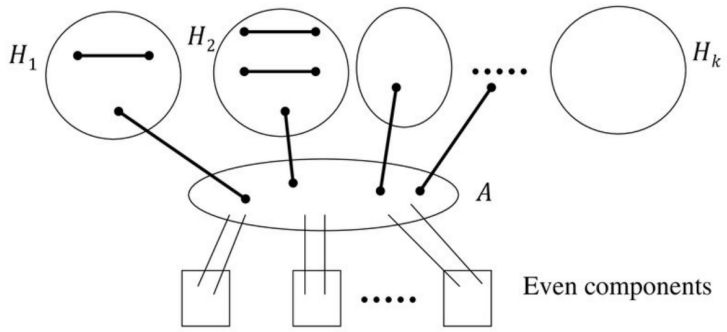


Figure 6.6: Odd components $H_1, \ldots, H_k$ of $G - A$, $A \subseteq V$, and even components of $G - A$.

Claim. *For any matching M of G, the number of M-exposed vertices is $\geq k - |A|$.*

Proof. If H_i has no M-exposed vertex, then there exists at least one edge in M from H_i to A, because an odd component cannot have a perfect matching. The same logic applies to all odd components. However, there can be at most $|A|$ such edges (between vertices in A and one of the H_i's), because we only have $|A|$ vertices in A to use. Therefore, there will always exist at least $k - |A|$ vertices that are M-exposed for all matchings M of G. □

Recall a matching M has $|V| - 2|M|$ vertices that are M-exposed. Thus, $|V| - 2|M| \geq k - |A|$, which implies $|M| \leq \frac{1}{2}(|V| - k + |A|)$. Since $\nu(G)$ is an upper bound of $|M|$ and the above expression must holds for every $A \subseteq V$, we get

$$\nu(G) \leq \frac{1}{2}(|V| - \text{odd}(G - A) + |A|) \qquad \forall A \subseteq V.$$

How tight is this bound? Observe if A is any vertex cover, then $\text{odd}(G - A) = |V| - |A|$ as deleting A from G removes every edge, which gives

$$\frac{1}{2}(|V| - \text{odd}(G - A) + |A|) = \frac{1}{2}(|V| - |V| + |A| + |A|) = |A| \geq \tau(G).$$

Therefore, this bound is *at least* as good as the vertex cover bound $\nu(G) \leq \tau(G)$.

Tutte's Matching Theorem

Theorem 6.17 (Tutte-Berge)**.** *Let $G = (V, E)$. Then*

$$\nu(G) = \frac{1}{2} \min_{A \subseteq V} \{|V| - \text{odd}(G - A) + |A|\}.$$

The Tutte-Berge Formula above tells us that a matching is maximum when it reaches the given size. We will give a formal proof later. For now, we prove an easier result, which can be viewed as a corollary of the Tutte-Berge Formula.

Theorem 6.18 (Tutte)**.** *G has a prefect matching $\Leftrightarrow \forall A \subseteq V : \text{odd}(G - A) \leq |A|$.*

Intuition. Each odd component "consumes" a vertex in A, so if $G - A$ has more than $|A|$ odd components, there cannot be a perfect matching.

Proof. Suppose $\text{odd}(G) > 0$. Then G has no perfect matching, so LHS is false; the RHS is also violated by setting $A = \varnothing$, so we have $\neg$ LHS $\iff$ $\neg$ RHS for the case $\text{odd}(G) > 0$.

Now suppose $\text{odd}(G) = 0$. By the Tutte-Berge formula, G has a perfect matching iff

$$\begin{aligned}
&\iff \nu(G) = \frac{n}{2} && \text{Size of perfect matchings} \\
&\iff n = \min_{A \subseteq V} \{n - \text{odd}(G - A) + |A|\} && \text{Tutte-Berge formula} \\
&\iff \min_{A \subseteq V} \{|A| - \text{odd}(G - A)\} = 0. && \text{Take } n \text{ out as it is a constant}
\end{aligned}$$

The minimum 0 is obtained by setting $A = \varnothing$ as $|A| - \text{odd}(G - A) = 0 - 0 = 0$, provided that $\text{odd}(G) = 0$. Therefore, the minimum evaluates to 0 iff $\forall A \subseteq V : \text{odd}(G - A) \leq |A|$, i.e., G has a p.m. $\iff \min_{A \subseteq V}\{|A| - \text{odd}(G - A)\} = 0 \iff \forall A \subseteq V : \text{odd}(G - A) \leq |A|$. This proves LHS $\iff$ RHS for the case $\text{odd}(G) = 0$. □

Odd Cycles

Let C be an odd cycle. Define $G' := G/C$ given by[1]

$$V(G') := (V(G) \setminus V(C)) \cup \{\mathscr{C}\} \quad \text{(vertex } \mathscr{C} \text{ represents the contracted cycle)}$$
$$E(G') := \{e \in E(G) : e \cap C = \emptyset\} \cup \{v\mathscr{C} : \exists uv \in E(G) \text{ with } u \in V(C), v \notin V(C)\}.$$

Let C be an odd cycle and $G' := G/C$. Observe a matching in G' can be extended to a matching in G with some exposed vertices.

Proposition 6.19. *Let $G = (V, E)$, C be an odd cycle, and $G' := G/C$. Let M' be a matching in G'. Then there exists a matching M of G such that the number of M-exposed vertices in G is equal to the number of M' exposed vertices in G'.*

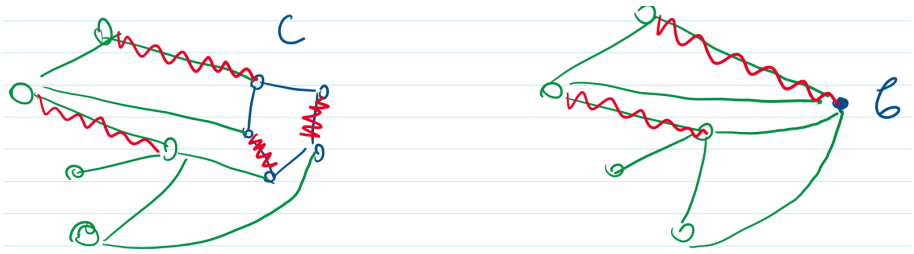


Figure 6.7: "Proof by example".

In particular, we add $(|C| - 1)/2$ new edges to M' to obtain M. It follows that

$$\nu(G) \geq \nu(G') + \frac{|C| - 1}{2}.$$

Unfortunately, equality does not always hold. In the example below, we have $\nu(G') = 1, (|C| - 1)/2 = 1$, but $\nu(G) = 3$:

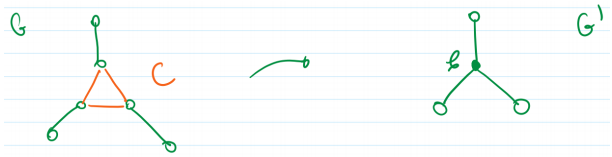


Figure 6.8: Equality does not always hold.

Definition 6.20. We say an odd cycle C is **tight** if

$$\nu(G) = \nu(G') + \frac{|C| - 1}{2}.$$

[1] From now on, we allow graphs to have parallel edges (but not loops) because we are contracting stuff.

Proving Tutte-Berge Formula

Theorem 6.21 (Tutte-Berge). *Let $G = (V, E)$. Then*

$$\nu(G) = \frac{1}{2} \min_{A \subseteq V} \{|V| - \text{odd}(G - A) + |A|\}.$$

Proof. ***Our goal is to produce a matching M and a Berge witness $A \subseteq V$ with exactly $\text{odd}(G - A) - |A|$ M-exposed vertices.*** Since the Tutte-Berge formula trivially provides an upper bound, to show the equality holds, it suffices to show there is a some (M, A) attaining this upper bound. We do induction on $m = |E|$. The case $m = 0$ is trivial. Now choose some $uv \in E$ arbitrarily.

Case 1. v is essential. Suppose that one of the endpoints is essential, say v. Define $G' := G - v$. Since v is essential, removing v necessarily decreases the size of every matching, so $\nu(G') < \nu(G)$. By induction, there is a maximum matching M' in G' and $A' \subseteq V - v$ with

$$|M'| = \frac{1}{2}((n-1) - \text{odd}(G' - A') + |A'|).$$

Let M be a matching of G with $|M| = \nu(G)$. Choose $e \in \delta(v) \cap M$. (Note there must be such an edge since v is essential.) Then $\bar{M} := M - e$ is a matching in G'. It follows that

$$|\bar{M}| = |M| - 1 \leq |M'|.$$

But $|M'| \leq |M| - 1 = |M|$ since v is essential. Thus, $|M'| = |M| - 1$. Now define $A := A' + v$. Notice that $\text{odd}(G - A) = \text{odd}(G' - A')$ by definition. It follows that

$$|M| - 1 = \frac{1}{2}((n-1) - \text{odd}(G' - A') + |A'|)$$
$$|M| = \frac{1}{2}(n - \text{odd}(G - A) + |A|) - 1 + 1 = \frac{1}{2}(n - \text{odd}(G - A) + |A|).$$

Case 2. u, v are inessential. By Lemma 6.22, we can pick a tight odd cycle C containing uv and where $\mathscr{C}$ is inessential in $G' := G/C$. Then there exists a matching M' of G', such that

$$\forall A' \subseteq V(G') : |M|' = \frac{1}{2}\left(|V(G')| - \text{odd}(G' \setminus A') + |A'|\right).$$

By Lemma 6.23, $\mathscr{C} \notin A'$. We claim that any component of $G' \setminus A'$ containing $\mathscr{C}$ will be a component of $G \setminus A'$ of the same parity. To see this, observe we remove $\mathscr{C}$ and add an odd number of vertices (from the odd cycle C) when we reconstruct $G \setminus A'$ from $G' \setminus A'$. Thus, even cycles stay even and odd cycles stay odd, so $\text{odd}(G' \setminus A') = \text{odd}(G \setminus A')$. By Proposition 6.19, we can extend M' to M with $\text{odd}(G' \setminus A') - |A'| = \text{odd}(G \setminus A') - |A'|$ M-exposed vertices as desired. □

Lemma 6.22. *Let $uv \in E$. If u, v are inessential, then there is a tight odd cycle C containing the edge uv such that $\mathcal{C}$ is inessential in $G' := G/C$.*

Proof. Let M_u, M_v be two maximum matchings exposing u and v, respectively. Notice:

- $uv \notin M_u \cup M_v$ as $uv \notin M_u$ and $uv \notin M_v$;
- M_u covers v, otherwise we could add uv to M_u. Similarly, M_v covers u.

Thus, both u and v have exactly one neighbour in $M_u \triangle M_v$, i.e., $\deg_\triangle(u) = \deg_\triangle(v) = 1$.

Let $F := M_u \triangle M_v$. We claim that (V, F) is a vertex disjoint union of M_u, M_v-alternating paths/cycles. Indeed, each vertex in (V, F) has degree at most two, so it is a vertex disjoint union of paths and cycles. Moreover, adjacent edges must come from different matchings, so it is M_u-alternating as well as M_v-alternating.

Since $\deg(u)_\triangle = 1$, there exists an alternating path P starting in u. Notice:

- The first edge of P is from M_v because u is exposed in M_u.
- The last edge of P is from M_u as otherwise this becomes an M_u-augmenting path.

Let z be the other end of P, which must be M_v exposed. If $z \neq v$, since $vu \notin M_v$, $vu + P$ is an M_v-augmenting path, contradiction. Therefore, this path P must be a u, v-path.

Consider cycle $C := uv + P$. Since P is alternating but not augmenting, P has an even number of edges, so C is an odd cycle.

Denote P by $u = v_1, v_2, \ldots, v_{n-1}, v_n = v$, $e_i = v_i v_{i+1}$, $i \in \{1, \ldots, n-1\}$. Note that the odd-indexed edges are in M_v and the even-indexed edges are in M_u.

We claim that $(\delta(C) \cap M_u) = \emptyset$. Suppose not, so there exists $wq \in M_u$ such that $w \in C$ and $q \notin C$. If $w = u$, then u is not M_u-exposed, contradiction. Now suppose $w = v_i$ for some $i \in \{1, \ldots, n\}$. If i is even, $e_i = v_i v_{i+1} \in M_u \cap C$; if i is odd, $e_{i-1} = v_{i-1} v_i \in M_u \cap C$. Thus, v_i is covered by an edge in $M_u \cap C$, contradiction as well. It follows that $\delta(C) \cap M_u = \emptyset$.

Next, we claim that $M_u \setminus C$ is a maximum cardinality matching in G/C. Let's first show it's a matching. Since $M_u \setminus C$ is a subset of a matching, we only need to show that $M_u \setminus C \subseteq E(G/C)$, which in turn implies $M_u \setminus C$ is a matching in G/C. Suppose not, so there is some $e \in M_u \setminus C$ such that $e \notin E(G/C)$. If $e \in \delta(C)$, i.e., $e = qw$ where $q \in M_u \cap C$ and $w \in M_u \setminus C$. But this contradicts $\delta(C) \cap M_u = \emptyset$. Now suppose $e = qw \notin C$ where $q, w \in V(C)$. But every vertex in $C \setminus \{u\}$ is already covered by an M_u edge in C, and u is exposed by construction. Thus, this case is also impossible.

Next,

$$|M_u \cap C| = \frac{|C|-1}{2} \implies |M_u \setminus C| = |M_u| - \frac{|C|-1}{2}.$$

Since C is an odd cycle,

$$\nu(G) \geq \nu(G/C) + \frac{|C|-1}{2}.$$

But we also have

$$\nu(G) = |M_u| = |M_u \setminus C| + \frac{|C|-1}{2},$$

which implies

$$|M_u \setminus C| + \frac{|C|-1}{2} \geq \nu(G/C) + \frac{|C|-1}{2} \implies |M_u \setminus C| \geq \nu(G/C) \implies |M_u \setminus C| = \nu(G/C),$$

so $M_u \setminus C$ is indeed a maximum cardinality matching. Substitute this into the line above, we have

$$\nu(G) = \nu(G/C) + \frac{|C|-1}{2},$$

which implies C is tight. We have found our desired circle C. □

Lemma 6.23. *Let M be a matching and a set of vertices $A \subseteq V$ that satisfies the Tutte-Berge formula*

$$|M| = \frac{1}{2}(|V| - \text{odd}(G \setminus A) + |A|).$$

Then all vertices in A are essential.

Proof. Let $v \in A$. Define $A' := A \setminus \{v\}$, $V' := V \setminus \{v\}$, and $G' := G \setminus \{v\}$. Note the components of $G \setminus A$ are the same as those of $G' \setminus A'$, so $\text{odd}(G \setminus A) = \text{odd}(G' \setminus A')$. Observe

$$\nu(G') \leq \frac{1}{2}\left(|V'| - \text{odd}(G' \setminus A') + |A'|\right) = \frac{1}{2}\left(|V| - 1 - \text{odd}(G \setminus A) + |A| - 1\right) = |M| - 1.$$

Since removing v from G necessarily decreases the size of any maximum matching of G, every maximum matching must have an edge incident to v, so v is essential. □

6.4 *Perfect Matching in General Graph: The Blossom Algorithm.*

Recall our algorithm for finding perfect matchings in bipartite graphs:

1. Given a matching M, pick an M-exposed vertex r.
2. Construct an M-alternating tree rooted at r and extend it if possible.
3. Upon finding an M-augmenting path starting at r, augment M with the path, and restart the algorithm with another M-exposed vertex.

When the algorithm terminates and there were still M-exposed vertices in the graph, we conclude that the bipartite graph cannot have a perfect matching. What happens if we apply this algorithm on a general (not necessarily bipartite) graph?

Frustrated Tree

Intuitively, a *frustrated tree* is an alternating tree where we can no longer make progress. That is, the current matching M is not perfect and we can't extend the tree or augment the matching anymore.

Definition 6.24. We say an M-alternating tree T is **frustrated** if for every edge $uv \in E$ such that $u \in B(T)$, the other endpoint satisfies $v \in A(T)$.

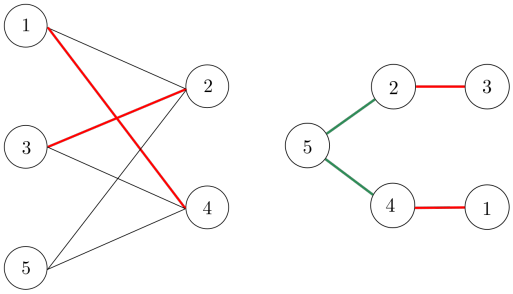


Figure 6.9: Left: Graph and matching (in red). Right: A frustrated tree.

The tree returned by the bipartite graph perfect matching algorithm when no perfect matching exists is an example of a frustrated tree. It is easy to derive a characterization using the notion of frustrated alternating trees.

Proposition 6.25. *If T is frustrated, then G has no perfect matching.*

As a remark, since we start with $B(T) = \{r\}$ and $A(T) = \emptyset$, at all times during the algorithm we have $|B(T)| - |A(T)| = 1$, so $|B(T)| > |A(T)|$.

Proof. Since all neighbours of vertices in $B(T)$ are in $A(T)$, $G \setminus A(T)$ has $\geq |B(T)|$ odd components (each of size 1). Then $|\text{odd}(G \setminus A(T))| \geq |B(T)| > |A(T)|$. It follows from Tutte's matching theorem that G has no perfect matching. □

Blossom

Let $u, v \in B(T)$ such that $uv \in E$. Then $T + uv$ has a unique odd cycle C (called a **Blossom**). Shrink the blossom and let $G' = G/C$. We call the new vertex a **pseudonode**.

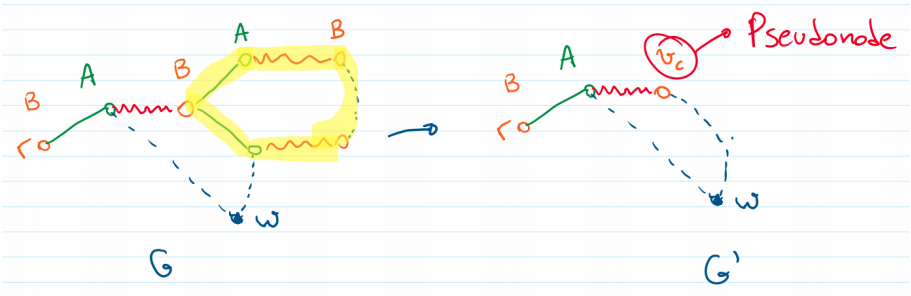


Figure 6.10: Blossom (highlighted in yellow) and pseudonode (circled in red).

Note that

- Edges in $M \setminus E(C)$ form a matching M' in G'.
- Shrunken tree T' is M'-alternating in G'.
- Pseudonode is in the set $B(T')$ for the tree T'.

Thus, we can keep shrinking blossoms and forgetting about the blossoms shrunken so far. Note that when an M'-augmenting path is found, one can augment M' using this path, then reconstruct M and augment it as well. Note one may need to shrink multiple times.

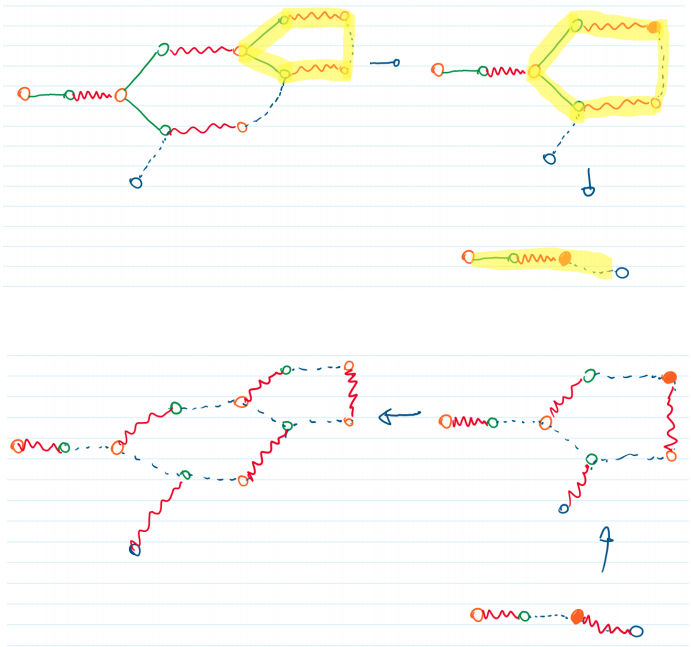


Figure 6.11: Top: shrink, shrink, discover an augmenting path. Bottom: augment, reconstruct, reconstruct.

Derived Graph

We say the graph obtained after shrinking (sequentially) blossoms is a **derived graph**. We use $S(v)$ to denote the set of vertices that have been shrunk into $v \in V(G')$.

Theorem 6.26. *Formally, for all $v \in V(G')$,*

$$S(v) = \begin{cases} v & v \in V(G) \\ \bigcup_{w \in \mathscr{C}} S(w) & v = v_{\mathscr{C}} \text{ for some blossom } \mathscr{C} \end{cases}$$

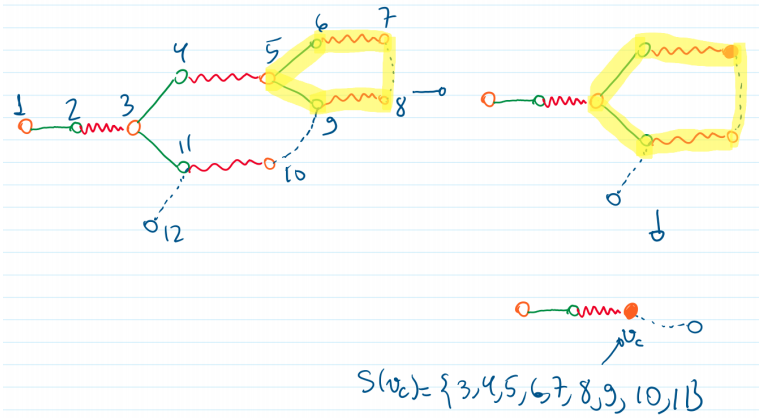


Figure 6.12: Example of $S(v)$.

Notice $|S(v)|$ is odd for all $v \in V(G')$, as it's either a single node or the union of odd cycles.

Proposition 6.27. *Let G' be a derived graph from G, M' a matching of G', T' an M'-alternating frustrated tree of G' with all pseudonodes in $B(T')$. Then G has no perfect matching.*

Proof. As before (note every vertex in $B(T')$ becomes an isolated vertex in $G' - A(T')$),

$$\operatorname{odd}(G' - A(T')) \geq |B(T')| > |A(T')|,$$

because every neighbour of a vertex in $B(T')$ resides in $A(T')$. Upon un-contraction of the blossoms of $B(T')$, each odd component of $G' - A(T')$ remains an odd component of $G - A(T')$ (the same parity argument applies). Thus,

$$\operatorname{odd}(G - A(T')) \geq |B(T')| > |A(T')|.$$

The result follows from Tutte's Matching Theorem. □

The following proposition shows we can recursively shrink blossoms and get desired results. (Combine this with Figure 6.11 and the previous proposition.) Proof is skipped.

Proposition 6.28. *Let G' be a derived graph from G, M' a matching of G', T' an M'-alternating tree, $uv \in E(G')$ with $u, v \in B(T')$, C' a unique cycle (Blossom) in $T' + uv$. Then $M'' := M' \setminus E(C')$ is a matching for $G'' := G' \setminus C'$ and $T'' := (V(T') \setminus V(C') \cup \{v_{C'}\}, E(T') \setminus E(C'))$ is an M''-alternating tree in G'' with $v_{C'} \in B(T'')$.*

Blossom Algorithm for Perfect Matching

Algorithm 7: Blossom Algorithm for Perfect Matching

input: A graph $G = (V, E)$ and a matching M of G.
output: A perfect matching M' of G, or a certificate proving no such matching exists
main:

Set $M' \leftarrow M$ and $G' \leftarrow G$.
Choose an M'-exposed node r of G' and put $T = (\{r\}, \emptyset)$
while there exists $vw \in E'$ with $v \in B(T)$ and $w \notin A(T)$ **do**
    case $w \notin V(T)$ and w is M'-exposed ▷ Case 1: Found augment path
        Use vw to augment M'
        Extend M' to a matching M of G
        Replace M' by M and G' by G
        if there is no M'-exposed node in G' **then**
            return the perfect matching M'
        else
            Replace T by $(\{r'\}, \emptyset)$ where r' is M'-exposed
    case $w \notin V(T)$ and w is M'-covered ▷ Case 2: Extend the tree
        Use vw to extend T
    case $w \in B(T)$ ▷ Case 3: Shrink a blossom and repeat
        Use vw to shrink a blossom $T_{v,w} + vw$ and update M' and T
return G', M', T' and stop. G has no perfect matching

Theorem 6.29. *Blossom algorithm does $O(n)$ augmentations, $O(n^2)$ shrinks, $O(n^2)$ tree extensions, and correctly determines if G has a perfect matching.*

Proof. The correctness follows from our previous propositions about frustrated trees, as that is exactly what our algorithm returns if no perfect matching is found. For running time, note that each augmentation increases $|M'|$ by 1, so there are at most $O(n)$ augmentations. Between two augmentation steps, shrinking reduces the size of G' by at least 2 vertices, which implies $O(n)$ shrinks. Thus, there are in total $O(n^2)$ shrinks. Finally, each edge is added to the tree at most once, so there are $O(m) = O(n^2)$ tree extensions. □

Blossom Algorithm for Maximum Cardinality Matching

We can adapt this algorithm to compute a maximum cardinality matching.

Algorithm 8: Blossom Algorithm for Maximum Cardinality Matching

input: A graph $G = (V, E)$ and a matching M of G.
output: A perfect matching M' of G, or a certificate proving no such matching exists
main:

1: Set $M' \leftarrow M, G' \leftarrow G, \mathcal{T} = \varnothing$ ▷ $\mathcal{T}$ tracks all frustrated trees
2: Choose an M'-exposed node r of G' and put $T = (\{r\}, \varnothing)$
3: **while** there exists $vw \in E'$ with $v \in B(T)$ and $w \notin A(T)$ **do**
4: **case** $w \notin V(T)$ and w is M'-exposed
5: Use vw to augment M'
6: Extend M' to a matching M of G
7: Replace M' by M and G' by G
8: **if** there is no M'-exposed node in G' **then**
9: **return** the perfect matching M'
10: **else**
11: Replace T by $(\{r'\}, \varnothing)$ where r' is M'-exposed
12: **case** $w \notin V(T)$ and w is M'-covered
13: Use vw to extend T
14: **case** $w \in B(T)$
15: Use vw to shrink and update M' and T
16: $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}, G' \leftarrow G \setminus V(T), M' \leftarrow M \setminus E(T)$
17: Goto line 2 if there exists any M'-exposed node r of G'.

In other words, we use $\mathcal{T}$ to keep track all frustrated trees, and on Lines 16 and 17 we remove the tree from the graph and reinitiate our algorithm.

Proposition 6.30. *The algorithm above produces a maximum matching of G.*

Proof. Let $\mathcal{T} := \{T_1, \ldots, T_k\}$ and M be the final matching. For each T_i, there is only one M-exposed vertex in T_i, namely its root r_i. Thus, there are k M-exposed vertices. Put $A := \bigcup_{i=1}^{k} A(T_i)$. Each vertex in $B(T_i)$ is an odd component of $G - A$. It follows that

$$\operatorname{odd}(G - A) \geq \sum_{i=1}^{k} |B(T_i)| \geq \sum_{i=1}^{k} (1 + |A(T_i)|) = |A| + k.$$

However,

$$|M| = \frac{n-k}{2} \geq \frac{1}{2}(n - \operatorname{odd}(G - A) + |A|).$$

The reverse inequality holds for any matching. Thus by Tutte-Berge we are done. □

6.5 *Gallai-Edmonds Decomposition.*

Definition 6.31. Let $G = (V, E)$,

- B be the set of inessential vertices,
- $C := \{v \in V \setminus B : v \in N_G(B)\}$, i.e., vertices adjacent to some $v \in B$,
- $D = V \setminus (C \cup B)$, i.e., everything else.

Then (B, C, D) is called the **Gallai-Edmonds partition/decomposition** of G.

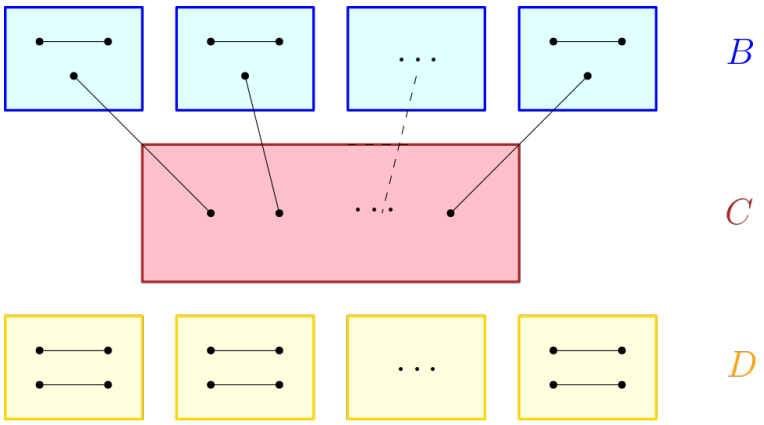


Figure 6.13: Gallai-Edmonds decomposition.

Intuitively,

- $G[B]$ only has odd components;
- C is a minimizer of the Tutte-Berge formula;
- In a maximum matching M,
  - $M \cap E(D)$ is a perfect matching of $G[D]$;
  - there exist some edges between C and B;
  - any M-exposed vertices are in B.

We will spend the rest of this section proving these claims.

Proposition 6.32. *Let $T_1, \ldots, T_k$ be the frustrated trees found in the Blossom algorithms. Then*

$$C = \bigcup_{i=1}^{k} A(T_i), \quad B = \bigcup_{i=1}^{k} \left(\bigcup_{v \in B(T_i)} S(v) \right), \quad D := V \setminus (B \cup C).$$

Note: This proposition implies that:

- All components of $G[B]$ are odd because they correspond to odd cycles.
- C is a minimizer of the Tutte-Berge formula (from the Blossom algorithm).
- Gallai-Edmonds decomposition can be computed in polynomial time.
- $G[D]$ only has even components: all vertices in D are M-covered and are consumed within the component.
- No $v \in D$ is adjacent to a vertex of B while every $v \in C$ is matched/adjacent to a vertex in B.

Proof. (Sketch) We saw all vertices in C are essential. This is because the Blossom algorithm finds a minimizer of the Tutte-Berge formula, of which all vertices are essential.

For all vertices in B, there is an even M-alternating path from an M-exposed vertex r (a root of one of the frustrated trees) to it. Take such an even path P and observe that $M' := -M \triangle E(P)$ is a matching with $|M'| = |M|$ exposing v. It follows that v is inessential.

Finally, we know that $G[D]$ only has even components. Let $v \in D$. We claim that $\nu(G - v) < \nu(G)$. Indeed, we have already shown that A is a minimizer to the Tutte-Berge formula. Therefore, there are exactly k exposed vertices in a maximum matching. But there are at least k exposed vertices of B, so no vertex of D can be exposed in a maximum matching. Thus, $G[D]$ contains a perfect matching as required. □

7.1 Minimum Weight Perfect Matching.

The problem of minimum weight perfect matching is given as below.

Problem. *Given $G = (V, E)$, $c \in \mathbb{R}^E$, find a perfect matching M of G minimizing*

$$c(M) = \sum_{e \in M} c_e.$$

If a perfect matching does not exist, return a certificate of infeasibility.

Recall the degree of each vertex is exactly 1 in a perfect matching. Consider the IP below.

$$\begin{array}{rll} \min & \sum_{e \in E} c_e x_e & \\ s.t. & x(\delta(v)) = 1 & \forall v \in V \\ & x \geq 0, x \in \mathbb{Z}^E & \end{array}$$

We throw away the $x \in Z^E$ constraint and obtain the following primal-dual LP pair.

$$\begin{array}{rll} (P_M) := \min & \sum_{e \in E} c_e x_e & \\ s.t. & x(\delta(v)) = 1 & \forall v \in V \\ & x \geq 0 & \\ (D_M) := \max & \sum_{v \in V} y_v & \\ s.t. & y_u + y_v \leq c_{uv} & \forall uv \in E \end{array}$$

An immediate observation is that if z_{PM} be an optimal solution of P_M, then $z_{PM} \leq c(M)$ for all perfect matchings M. But can we hope to solve the problem of minimum weight perfect matching by solving P_M? The answer is no. Consider the following example. Any perfect matching must include at least one of the horizontal edges of cost 1, yet the optimal value to P_M is 0. (Show)

As a remark, this is what happens in most cases: if you have some combinatorial optimization problem and you write down what you think as the most natural IP formulation, chances are, the optimal solution to its LP relaxation is not integral, i.e., it will not correspond to a combinatorial object you are looking for. The MST problem is a very special case where you could drop the integrality constraint and obtain a solution directly.

7.2 Minimum Weight Perfect Matching in Bipartite Graphs.

In bipartite graphs, the LP approach described in the previous section works.

Theorem 7.1 (Birkhoff). *Let $G = (V, E)$ be a bipartite graph, $c \in \mathbb{R}^E$. Then G has a perfect matching iff P_M is feasible. Moreover, if P_M is feasible, let M^* be the min cost perfect matching. Then we have $z_{PM} = c(M^*)$.*

The forward direction of the iff is trivial: if G has a perfect matching, then P_M is feasible. The backward direction will be shown in A3. We will do an algorithmic proof for the second statement.

Deduce Algorithm from CS Conditions

Recall the primal-dual LP pair:

$$\begin{aligned}(P_M) := \min \quad & \textstyle\sum_{e\in E} c_e x_e \\ s.t. \quad & x(\delta(v)) = 1 \quad \forall v \in V \\ & x \geq 0\end{aligned}$$

$$\begin{aligned}(D_M) := \max \quad & \textstyle\sum_{v\in V} y_v \\ s.t. \quad & y_u + y_v \leq c_{uv} \quad \forall uv \in E\end{aligned}$$

Assuming G is bipartite, we want to construct a matching M that corresponds to an optimal solution to P_M. As before, we use CS conditions. Let $\bar{y}$ be feasible for D_M. Define

$$E^= := \{uv \in E : \bar{y}_u + \bar{y}_v = c_{uv}\},$$

i.e., $E^=$ contains the edges whose dual constraints are satisfied at equality. Recall the CS condition requires either a dual constraint is tight, or the corresponding primal variable is 0. Thus, for any edge we want to include in our matching (i.e., $x_{uv} = 1$), then the corresponding dual constraint must be tight. In other words, $E^=$ contains the edges we could use in our matching. As a remark, the other CS condition requires either a primal constraint is tight or the corresponding dual variable is 0. But every constraint must be satisfied at equality for any feasible solution,so we don't need to worry about this one.

If $G^= := (V, E^=)$ has a perfect matching M, then $x^M, \bar{y}$ satisfy the CS conditions, which in turn shows that M is a minimum weight perfect matching. Otherwise, we want to update $\bar{y}$. This will be the basic flow of our algorithm. As a remark, this relies on the perfect matching identification algorithm we covered in the last chapter.

Updating Potentials

Recall after running the perfect matching algorithm on $G^=$; two cases could happen. If we found a perfect matching M, since $M \subseteq E^=$, this is the desired min-weight perfect matching in G. If we found a frustrated tree, we want to update $E^=$ by updating y, where

- $\bar{y}$ is still feasible for D_M: maintain a feasible dual solution.
- The current $M \subseteq E^=_{\text{new}}$: keep our matching valid.
- The current $E(T) \subseteq E^=_{\text{new}}$: keep the alternating tree valid.
- At least one edge $uv \in E \setminus E^=_{\text{old}}$ satisfies $u \in B(T), v \notin V(T)$ is in $E^=_{\text{new}}$: bringing in new candidates for our matching.

Recall edges not in $E^=$ satisfy $\bar{y}_u + \bar{y}_v < c_{uv}$. To keep y a feasible dual solution and to bring more edges into our candidate set $E^=$, we define

$$\varepsilon := \min\{c_{uv} - \bar{y}_u - \bar{y}_v \mid \exists uv \in E, u \in B(T), v \notin V(T)\}$$

and update $\bar{y}_u$ with

- $\forall u \in B(T) : \bar{y}_u \leftarrow \bar{y}_u + \varepsilon$;
- $\forall u \in A(T) : \bar{y}_u \leftarrow \bar{y}_u - \varepsilon$;
- $\forall u \notin V(T) : \bar{y}_u$ remains unchanged.

This ensures that $\bar{y}$ is still feasible for the dual constraint

$$\forall uv \in E : y_u + y_v \leq c_{uv}.$$

Next, $M \subseteq E^=$ for the new $E^=$ by construction; $E(T) \subseteq E^=$ since we increased/decreased the vertex potentials based on parity of distance from the root. And if there are any vertices $u \notin V(T)$, our update ensures that at least one M-exposed vertex is found and we can augment our matching.

Example

Here is an example of running the algorithm for one step. Note the number above each vertex u is the potential $\bar{y}_u$ and the number above each edge uv is the cost c_{uv}. We use colors to denote the edges within different groups.

Figure 7.1: Current situation.

Suppose we ran the perfect matching algorithm on $G^=$ and saw ourselves in the situation described in the figure. Observe there are two edges not in $E^=$ that goes from $u \in B(T)$ to $v \notin V(T)$, namely the two orange edges. Since the lower edge has a weight of 2 and a sum of potentials 1 and the upper edge has a weight of 5 and a sum of potentials 3, we want to set $\varepsilon = \min\{5-3, 2-1\} = 1$. We then increase all potentials of vertices in $B(T)$ by $\varepsilon = 1$ and decrease all potentials of vertices in $A(T)$ by $\varepsilon = 1$, which results in the graph below:

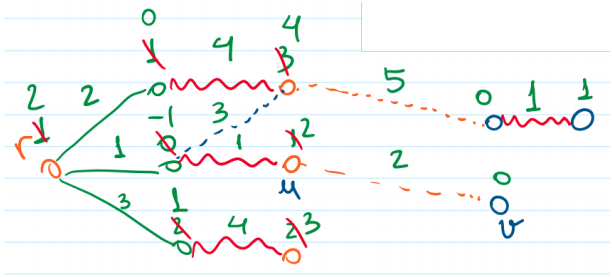


Figure 7.2: Updating the potentials.

Now observe the path from r to v is an M-augmenting path.

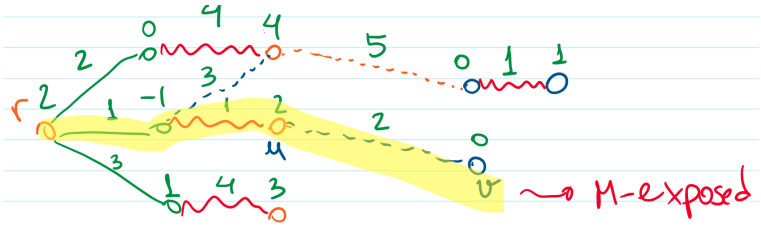


Figure 7.3: Detecting an M-augmenting path.

Augment our matching with this path. Since this is a perfect matching of $G^=$, by our previous remark about CS conditions, it must be a minimum weight perfect matching of G. We are done.

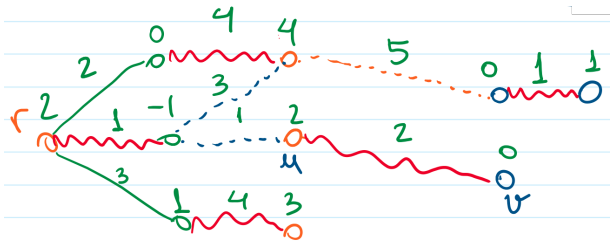


Figure 7.4: Found a perfect matching. Done.

It's time for the actual algorithm.

Algorithm 9: Algorithm for Finding a Min Weight Perfect Matching in Bipartite Graphs

input: : $G = (V, E)$ bipartite
output: : A perfect matching M of G or None if G does not have a perfect matching
main:

Let y be a feasible dual solution and M be a matching of $G^=$
if M is a perfect matching of G **then**
    return M
Initialize $T \leftarrow (V(T) = \{r\}, E(T) = \emptyset)$, where r is an M-exposed vertex in G
$A \leftarrow \emptyset, B \leftarrow \{r\}$, the odd-distanced and even-distanced vertices, respectively
while True **do**

    # Find a perfect matching in our bipartite graph
    while $\exists vw \in E : v \in B(T), w \notin V(T)$ **do**
        if w is M-covered **then**
            Use vw to extend T, $A \leftarrow A \cup \{w\}, B \leftarrow B \cup \{w\}$
        else ▷ w is M-exposed
            Use vw to augment M
            if $\exists M$-exposed vertex $r' \in V$ **then**
                $T \leftarrow (\{r'\}, \emptyset)$ where r' is M-exposed, $A \leftarrow \emptyset, B \leftarrow \emptyset$
                continue
            else ▷ No M-exposed left vertex in G, we are done.
                return the perfect matching M

    # Update dual solution
    if every $vw \in E$ with $v \in B(T)$ has $w \in A(T)$ **then** ▷ Frustrated tree!
        Stop, G has no perfect matching
    else
        Let $\varepsilon = \min\{\bar{c}_{vw} : v \in B(T), w \notin V(T)\}$
        Replace y_v by $y_v + \varepsilon$ for $v \in B(T)$, $y_v - \varepsilon$ for $v \in A(T)$

To see correctness, observe $M \subseteq E^=$ at all time, and we have the following two stop points:

- Found a perfect matching M in $G^=$: $M \subseteq E^=$ so by CS conditions we are good.
- Found a frustrated tree in G: no perfect matching exists, we are doomed.

Observe that the inner while loop is the bipartite matching algorithm and terminates in polynomial time. On the other hand, if the outer loop does not terminate, then it is guaranteed we have an M-augmenting path in $G^=$ in the next iteration. Thus, the outer loop runs for at most n iterations. To summarize, the algorithm correctly finds a minimum weight perfect matching in a bipartite graph with polynomial time.

7.3 *Minimum Weight Perfect Matching in General Graphs.*

A New Constraint

Observe any perfect matching must use at least one edge in $\delta(S)$:

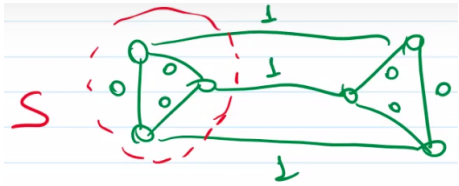


Figure 7.5: Any perfect matching must use at least one edge in $\delta(S)$.

Let us add some constraints to the LP that represent this observation:

$$\forall S \subseteq V \text{ where } |S| \text{ is odd and } |S| \geq 3 : x(\delta(S)) \geq 1$$

Let us denote these S by $\mathcal{O}$, the "odd sets". This leads to a new formulation:

$$\begin{array}{lll|lll} (P'_M) := \min & \sum_{e\in E} c_e x_e & & (D'_M) := \max & \sum_{v\in V} y_v + \sum_{S\in\mathcal{O}} y_S & \\ s.t. & x(\delta(v)) = 1 & \forall v \in V & s.t. & y_u + y_v + \sum_{S\in\mathcal{O}:uv\in\delta(S)} y_S \leq c_{uv} & \forall uv \in E \\ & x(\delta(S)) \geq 1 & \forall S \in \mathcal{O} & & y_S \geq 0 & \forall S \in \mathcal{O} \\ & x \geq 0 & & & & \end{array}$$

With this constraint added, the set of feasible solutions of the IP remains the same, but the set of feasible solutions of the LP P'_M is different. In particular, the counterexample we provided before is no longer a feasible solution to P'_M as it violates the new constraint!

Algorithm Implied by CS Conditions

Similar what we did before, we will try to compute a perfect matching which satisfy CS conditions. We will only use edges uv with $\bar{c}_{uv} = 0$ in our matching:

$$\bar{c}_{uv} := c_{uv} - \sum_{S\in\mathcal{O}:uv\in\delta(S)} y_S - y_u - y_v$$

The CS conditions requires that:

- $\forall v \in V : x(\delta(v)) = 1$: satisfied by any perfect matching, so we are good.
- $\forall S \in \mathcal{O} : x(\delta(S)) = 1$ or $y_S = 0$: need some work.
- $\forall uv \in E : x_{uv} = 0$ or $\bar{c}_{uv} = 0$: need some work.

We will construct a perfect matching M such that

- ⋆ $M \subseteq E^= := \{e \in E : \bar{c}_e = 0\}$
- ⋆⋆ $|M \cap \delta(S)| = 1$ for all $S \in \mathcal{O}$ when $y_S > 0$.

Note that $E^=$ is defined based on implicitly the graph G and a feasible dual solution $\bar{y}$.

Why the previous algorithm for bipartite graph fails here

Let's start with what we did for bipartite graphs:

1. Start with a feasible dual solution $\bar{y}$ such that $\bar{y}_S = 0$ for all $S \in \mathcal{O}$.
2. If found a perfect matching M in $G^=$, we are done.
3. Otherwise, look at the frustrated tree T in $G^=$ and update $\bar{y}$ so that more vertices can be added to T.

$$\varepsilon := \{\min \bar{c}_{uv} : u \in B(T), v \notin V(T)\}, \qquad \bar{y}_u := \begin{cases} \bar{y}_u + \varepsilon & u \in B(T) \\ \bar{y}_u - \varepsilon & u \in A(T) \\ \bar{y}_u & u \notin V(T) \end{cases}$$

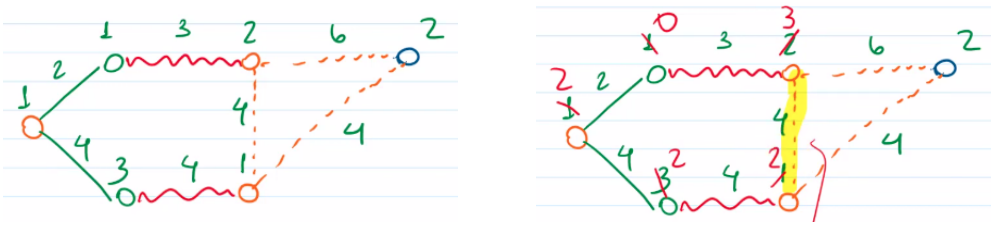


Figure 7.6: Applying the MWPM algorithm for bipartite graph to a non-bipartite graph.

Given the left graph, we see that $\varepsilon = \min\{4-3, 6-4, 4-3\} = 1$. Update $\bar{y}$ using the above rule. Note that the highlighted vertical edge has a weight of 4 but the potentials at its ends sum up to $3+2=5$, which means $\bar{c}_{uv} = -1$, violating the dual constraint!

To address this issue, we define ε_2, which imposes an upperbound on how much we can increase the potentials at both ends of an edge without violating the dual constraint.

$$\begin{aligned} \varepsilon_1 &:= \min\{\bar{c}_{uv} : u \in B(T), v \notin V(T)\} \\ \varepsilon_2 &:= \min\{\bar{c}_{uv}/2 : u \in B(T), v \in V(T)\} \\ \varepsilon &:= \min\{\varepsilon_1, \varepsilon_2\} \end{aligned}$$

With this modification, $\varepsilon = \min\{1, 1/2\} = 1/2$, so we update the potentials as below. Note that the vertical edge now is part of $E^=$!

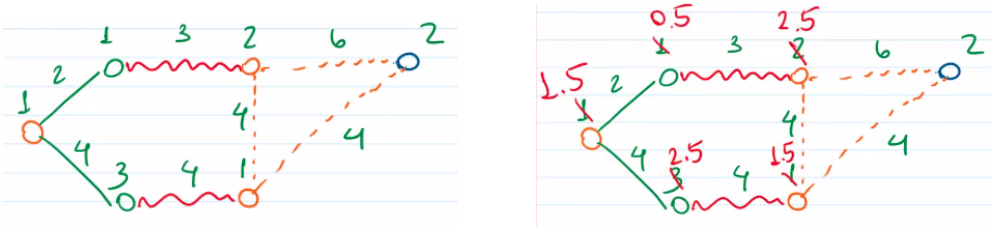


Figure 7.7: Fixing the problem.

Observe the left 5 edges make up a blossom in $G^=$. Therefore, if we find a perfect matching in $G^=/C$, then it can be extended to a perfect matching M of $G^=$ satisfying $\star\star$. We will see more details about this later.

Shrinking blossoms

But how exactly do we shrink the blossom? Previously when we were working with unweighted perfect matchings, all we care about is whether we could find a perfect matching in the shrunken graph; we don't care which vertex the pseudo-vertex v_C really represents. But now since edges could have different weights and we are looking for the one with minimum weight, we do need to care about this.

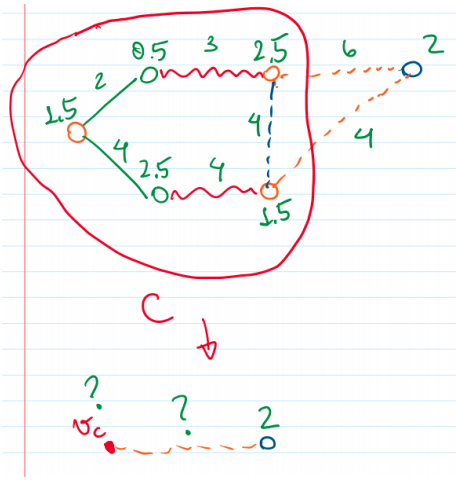


Figure 7.8: The resulting graph (edge) could either be the edge of weight 4 or the one of weight 6.

What are my potential and weight for this graph here? First, since v_C represents a blossom, the value of y_{v_C} should just be

$$y_{v_C} := y_{S(v_C)},$$

the total potential of the set of vertices in the shrunken blossom. Since we started with $\bar{y}_S = 0$ for all $S \in \mathcal{O}$, we can just initialize this to 0. Not too bad.

The bigger problem is the edge weight. In the unweighted case, we don't differentiate between the edges, but now since they have different weights, we want to keep both of them in the shrunken graph. Thus, we introduce parallel edges. Note this implies that we will be working with a multi-graph.

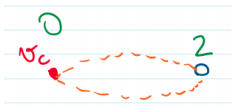


Figure 7.9: Introducing parallel edges to retain information on both edges.

Another observation is that we never work with the edge weight directly in the algorithm. Instead, we work with the CS conditions, i.e., the c_e is used to calculate $\bar{c}_{uv}$, which tells us how much more we can increase y_v or y_u and still be feasible for the dual LP. Recall

$$\bar{c}_{uv} := c_{uv} - y_u - y_v - \sum_{S \in \mathcal{O}: uv \in \delta(S)} y_S \geq 0.$$

Consider the edge uv in the original graph. We want to reflect the fact that we won't be modifying y_u after shrinking, and since $\bar{c}_{uv}$ must remain positive, the upper bound of $y_v + \sum_{S \in \mathcal{O}: uv \in \delta(S)}$ becomes $6 - 2.5 = 3.5$. In general, we update the edge weight to be

$$c'_{v_C v} := c_{uv} - y_u.$$

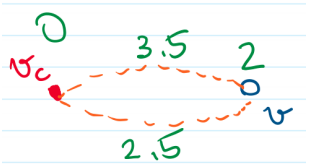


Figure 7.10: Assigning new costs to the edges.

This concludes the shrinking part: we know how to shrink, and we know the potentials and weights for the resulting graph.

Continue the Perfect Matching Algorithm

Continue the perfect matching algorithm, find v_C as the exposed vertex, cannot extend it because it's in orange, so we increase y_{v_C} here to 0.5. The bottom edge is now in $E^=$.

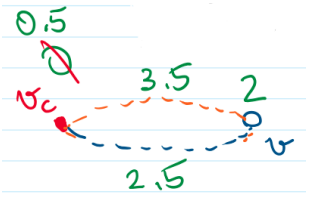


Figure 7.11: Assigning new costs to the edges.

Build a perfect matching from this. As a remark, $y_{v_C} = 0.5$ means the blossom has a potential of 0.5.

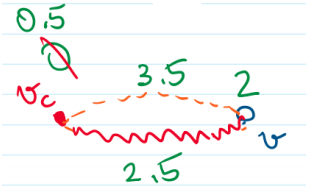


Figure 7.12: Find a perfect matching in the resulting graph.

Reconstruct the perfect matching.

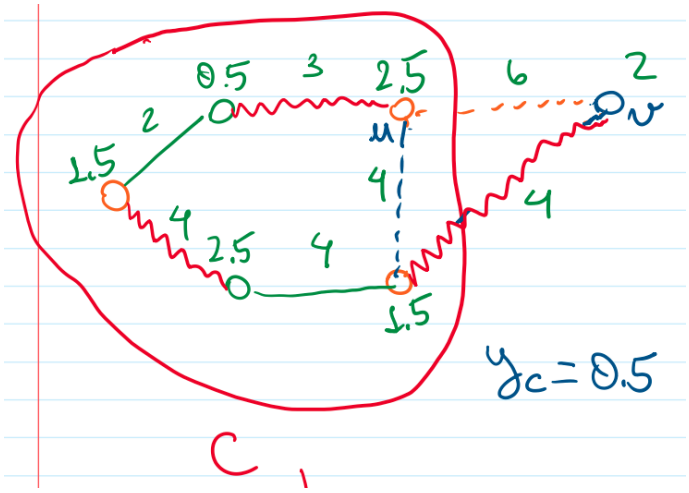


Figure 7.13: Reconstruct the perfect matching.

Observe this matching satisfies $\star$ and $\star\star$. First, all edges in the matching are in my $E^=$ (the special edge here is the right one: the total potential is $2 + 1.5 + 0.5 = 4$ where the 0.5 comes from the blossom), and we use exactly one edge in the right triangle which is the only $S \in \mathcal{O}$ such that $y_S > 0$.

Time to formalize this.

Algorithm for Shrinking

We say G', c' is **derived** from G, c by shrinking a blossom C if G', c' are defined as:

- $V(G') := (V(G) \setminus C) \cup \{v_C\}$;
- $\forall uv \in E$:
  - If $u, v \notin V(C)$, add uv to $E(G')$, and let $c'_{uv} = c_{uv}$.
  - If $u \in V(C), v \notin V(C)$, add $v_C v$ to $E(G')$, with cost $c'_{v_C v} := c_{uv} - y_u$.

Algorithm for Finding Perfect Matchings

Let's start with what we did for bipartite graphs:

1. Start with a feasible dual solution $\bar{y}$ such that $\bar{y}_S = 0$ for all $S \in \mathcal{O}$.
2. If found a perfect matching M in $G^=$, we are done.
3. Otherwise, look at the frustrated tree T in $G^=$ and update $\bar{y}$ so that more vertices can be added to T. If update allows to find a blossom C with $E(C) \subseteq E^=$, shrink it, and set $y_{v_C} = 0$.

The following proposition formalizes the idea.

Proposition 7.2. *Let $\bar{y}$ be feasible for $D'_M(G)$ with $\bar{y}_S = 0$ for all $S \in \mathcal{O}(G)$. Let G', c' be derived from G, c by shrinking blossom C with $E(C) \subseteq E^=(G, \bar{y})$. Let M' be a perfect matching of G' and y' feasible for $D'_M(G')$, where M', y' satisfy $\star, \star\star$, and $y'_{v_C} \geq 0$.[a] Extend M' to a perfect matching $\hat{M}$ of G and define $\hat{y}$ as*

- *$\hat{y}_v = \bar{y}_v$ for all $v \in V(C)$;*[b]
- *$\hat{y}_v = y'_v$, for all $v \in V(G') \setminus v_C$;*[c]
- *$\hat{y}_{S(v_C)} = y'_{v_C}$.*[d]
- *$\hat{y}_{S(D)} = y'_D$, for all $D \in \mathcal{O}(G')$;*
- *$\hat{y}_S = 0$, for all other $S \in \mathcal{O}$;*

Then $\hat{y}$ is a feasible solution for $D'_M(G)$ and $\hat{M}, \hat{y}$ satisfy both $\star$ and $\star\star$.

[a] Note the $y'_{v_C} \geq 0$ is an extra condition here. The y variables didn't have sign constraints in the dual LP. But we need this constraint here to make the proposition work. A deeper reason is that y_{v_C} corresponds to a blossom, so when we un-shrink the blossom, this is in fact one of the y_S variables which has to be non-negative. See the bottom of this page for more info.

[b] The potentials for vertices in C remain the same.

[c] y'_v here is the potential of v in the shrunken graph. When we reconstruct the perfect matching, we just keep it this way for all $v \in V(G') \setminus v_C$.

[d] The potential of the blossom (more precisely, the vertex set of the blossom) is equal to the potential of the pseudo-vertex in the shrunken graph.

Proof. Omitted. □

Another Bound on Potential

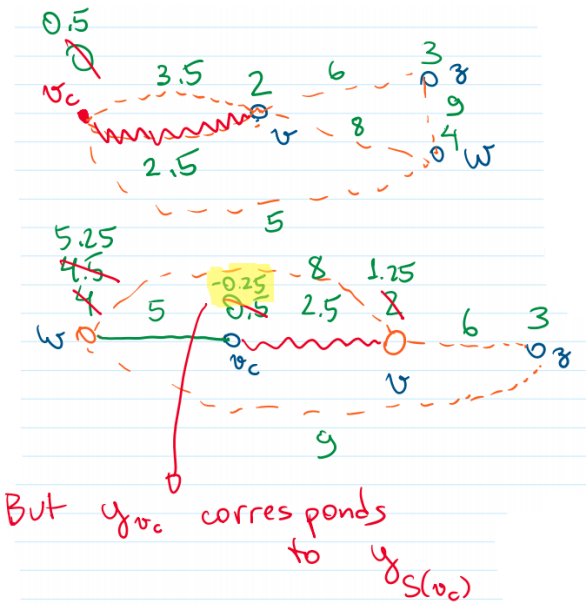


Figure 7.14: New bound ε_3.

In the current iteration of the algorithm, we get $\varepsilon_1 = 1$ and $\varepsilon_2 = 0.75$, so we pick $\varepsilon = 0.75$ and update the potentials. Note that y_{v_C} corresponds to $y_{S(v_C)}$, i.e., it corresponds to a blossom instead of a variable. For y to be a feasible dual solution, y_S must be non-negative for all $S \in \mathcal{O}$. Our update here makes y_{v_C} negative, violating this constraint. (Remark: Observe we require $y'_{v_C} \geq 0$ in the proposition above. Here we revealed the reason.) To address this, we introduce ε_3 given by

$$\varepsilon_3 := \min\{\bar{y}_u : u \in A(T) \text{ and } u \text{ is a pseudonode}\}.$$

With this change, $\varepsilon = \varepsilon_3 = 0$.

Un-shrink blossoms when stuck

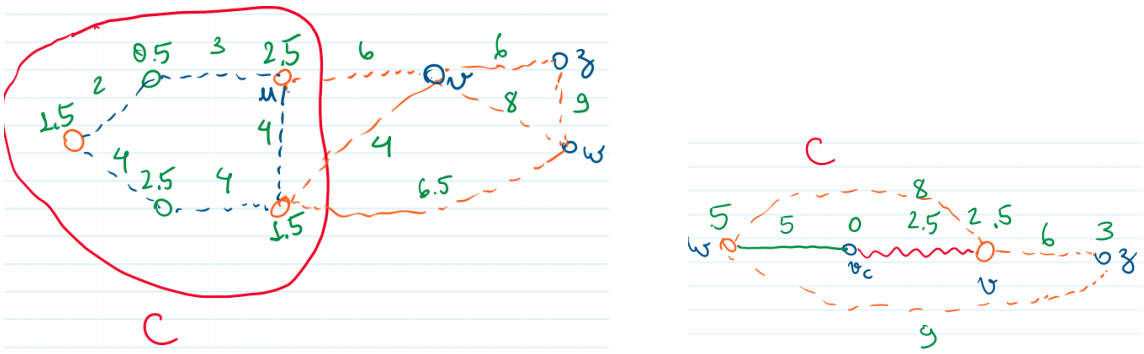


Figure 7.15: Got stuck. Time to expand C. Left: original graph. Right: current derived graph.

Now we are stuck. Since $y_{v_C} = 0$, $\varepsilon = \varepsilon_3 = 0$ from now on, so we can no longer make progress in this derived graph. Let's try to expand the blossom C. Recall we don't want to expand the blossom every time we found an augmenting path because we don't want to keep track of all $S \in \mathcal{O}$ with $y_S > 0$. But now since $y_{v_C} = 0$, expanding C back wouldn't cause any problems.

A formal description of expanding the blossom is as follows. Suppose we are given a matching M' consisting of equality edges of a derived graph G', an M'-alternating tree T consists of equality edges, and an odd pseudonode v_C of G' such that $y_{v_C} = 0$.

Let f, g be the edges of T incident with v, let C be the circuit that was shrunk to form v_C, let u, w be the ends of f, g in $V(C)$, and let P be the even-length path in C joining u to w. Replace G' by the graph obtained by expanding C. Replace M' by the matching obtained by extending M' to a matching of G'. Replace T by the tree having edge-set $E(T) \cup E(P)$. For each edge st with $s \in V(C)$ and $t \notin V(C)$, replace c'_{st} by $c'_{st} + y_s$.[1] The expanded graph is shown below.

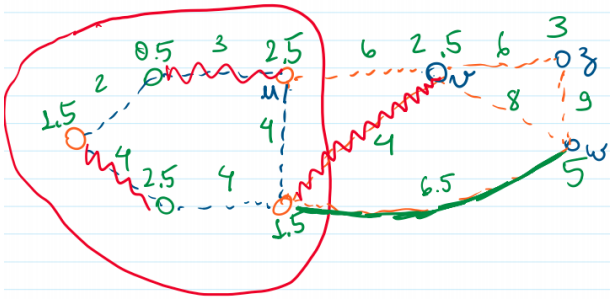


Figure 7.16: Expand the blossom.

[1] This last step corresponds to "restoring" the cost of the edges. See Figure 7.11 for more info on how we arrived at c'_{st} in the first place.

Algorithm

Algorithm 10: Algorithm for Finding a Min Weight Perfect Matching in General Graphs

input: : $G = (V, E)$
output: : A perfect matching M of G or None if G does not have a perfect matching
main:

Let y be a feasible dual solution, M' be a matching of $G^=$, and $G' \leftarrow G$.
if M' is a perfect matching of G **then**
    return M'. Done.
Initialize $T \leftarrow (V(T) = \{r\}, E(T) = \emptyset)$, where r is an M-exposed vertex in G
$A \leftarrow \emptyset, B \leftarrow \{r\}$, the odd-distanced and even-distanced vertices, respectively
while True **do**

// Augmenting path found. Augment M' and continue working in the derived graph.
    case $\exists e \in E^=$ whose ends in G' are $v \in B(T)$ and an M' exposed node $w \notin V(T)$
        Use vw to augment M'
        if there is no M'-exposed node in G' **then**
            Extend M' to a perfect matching M of G and stop
        else
            Replace T by $(\{r'\}, \emptyset)$ where r is M'-exposed

// Extend the tree. Same as before.
    case $\exists e \in E^=$ whose ends in G' are $v \in B(T)$ and an M'-covered node $w \notin V(T)$
        Use vm to extend T

// Found edge $e \in E^=$ that creates a blossom. Shrink and work in the derived graph.
    case $\exists e \in E^=$ whose ends in G' are $v, w \in B(T)$
        Use vw to shrink and update M', T and c'

// Stuck. Expand the blossom and work in the expanded graph.
    case $\exists v_C \in A(T)$ pseudonode with $y_{v_C} = 0$
        Expand v_C and update M', T and c'

// Check if we've reached a frustrated tree. If not, update potentials and continue.
    case None of the above
        if every $e \in E$ incident in G' with $v \in B(T)$ has its other end in $A(T)$ and $A(T)$ contains no pseudonode **then**
            Stop, T is a frustrated tree and $A(T)$ has no perfect matching
        else Change y and continue our algorithm

Correctness can be shown by arguing each step preserves $M \subseteq E^{=}$ (and use CS conditions). Omitted. To show polynomial time complexity, we bound the number of steps while keeping the same matching. The key remark is that shrink and unshrink may lead to infinite loop (as they may undo each other). To address this, observe every unshrink is done on a pseudonode in $A(T)$. Every time we shrink, the newly-created pseudonode lands in $B(T)$. The only way for a pseudonode to go from $B(T)$ to $A(T)$ is that we modified matching. Therefore, we will not be stuck in an infinite loop.

One more thing to note: Every time we do a change in y, the next step will always be either: shrink, unshrink, extend the tree, or augment the matching.

This concludes the minimum weight perfect matching algorithm.

7.4 Maximum Weight Matching.

Maximum Weight Perfect Matching

Suppose we are given a minimum weight perfect matching algorithm as a black box. Reducing the **maximum weight perfect matching problem** to the minimum weight perfect matching problem is trivial: just multiple the weights by -1.

Maximum Weight Matching

A more interesting problem is the **maximum weight matching** problem, where the goal is to find a matching M (not necessarily perfect) maximizing $c(M)$. There are direct algorithms to solve this, but we can also reduce this to maximum weight perfect matching.

Let $G = (V, E)$ and $c \in \mathbb{R}^E$. Let G' be a copy of G with exactly the same edge costs.

Let $\bar{G}$ be a graph with vertices $V(G) \cup V(G')$, edges $E(G) \cup E(G') \cup \{vv' : v \in V(G)\}$ where these new edges have costs 0. Note that $\bar{G}$ always has a perfect matching (by just using the new edges).

Proposition 7.3. *Let $\bar{M}$ be a maximum weight perfect matching in $\bar{G}$. Then $M = \bar{M} \cap E(G)$ is a maximum weight matching in G.*

Proof. It is clear that M is a matching of G. Let M^* be a maximum weight matching of G. We can copy M^* in G' and take the edges vv' for M^*-exposed vertices v in G to construct a perfect matching in $\bar{G}$. This matching has a weight of $2 \cdot c(M^*)$.

Since the edges in $\delta(V(G))$ all have weight 0, the edges within $E(G)$ and $E(G')$ contribute to all the weights. In particular, we must have $c(M) \geq c(\bar{M} \cap E(G'))$ or else $\bar{M}$ was not maximum (take a copy of $\bar{M} \cap E(G')$ in G instead).

It follows that $2 \cdot c(M) \geq c(\bar{M}) \geq 2 \cdot c(M^*) \implies c(M) \geq c(M^*)$. □

LP Formulation for Maximum Weight Matching

$$
\begin{aligned}
(P_M) := \max \quad & \textstyle\sum_{e \in E} c_e x_e & \\
s.t. \quad & x(\delta(v)) = 1 & \forall v \in V \\
& x(E(S)) \leq |S| - 1 & \forall S \in \mathcal{O} \\
& x \geq 0
\end{aligned}
$$

8 T-JOINS

This chapter is closely related to weighted matchings. Make sure to read that chapter first.

8.1 Motivation.

Euler Tour and Postman Tour

Given a connected graph $G = (V, E)$ (potentially having parallel edges), an **Euler tour** is a closed walk visiting every edge *exactly* once. Note a connected graph G has a Euler tour iff every vertex of G has an even degree.

A **postman tour** is a closed walk traversing every edge at least once. Thus, every Euler tour is a postman tour.

Problem. *Find a minimum cost postman tour wrt* $c \in \mathbb{R}^E_+$.

It is clear that if G has an Euler tour T, then T must be an optimal solution to this problem. Thus, the interesting case is when G is not **Eulerian**. Let $T \subseteq V$ be the set of vertices with odd degrees. By the hand-shaking lemma, each graph contains an even number of odd vertices, i.e., $|T|$ is even or $|T| \bmod 2 \equiv 0$.

Consider a postman tour and say it visits each edge $1 + x(e)$ times, where $x(e) \in \mathbb{Z}_{\geq 0}$. Then, it is easy to see that the multigraph induced by placing $1 + x_e$ copies of e is in fact Eulerian. Conversely, if $x \in \mathbb{Z}^E_{\geq 0}$ and the multigraph is Eulerian, then it induces a postman tour of cost $\sum_{e \in E} c_e(1 + x_e)$. In the following example, $x \in \mathbb{Z}^E_{\geq 0}$ is labeled red.[1]

Figure 8.1: G and G^x.

This motivates the following LP formulation.

[1] Note this x does not give us a postman tour because G^x does not have a Euler tour as the left vertex has an odd degree. A feasible x is given by $x_e = 1$ for the middle edges and 0 otherwise.

Linear Program Formulation

Let $x \in \mathbb{Z}^E_{\geq 0}$. Let G^x be obtained by making $1 + x_e$ copies of e (all with cost c_e). The idea is to find x such that G^x has an Euler tour. Equivalently, we want every vertex v in G^x to have an even degree (the condition of having a Euler tour), i.e.,

$$\forall v \in V : \sum_{e \in \delta(v)} (1 + x_e) \equiv 0 \bmod 2.$$

This is equivalent to saying $x(\delta(v)) \equiv |\delta(v)| \bmod 2$ as

$$\sum_{e \in \delta(v)} (1 + x_e) = \sum_{e \in \delta(v)} 1 + \sum_{e \in \delta(v)} x_e = |\delta(v)| + x(\delta(v)).$$

Observe that if $x_e \geq 2$, then reducing x_e by 2 maintains feasibility. Combined with the assumption that $c_e \geq 0$, we may assume $x_e \in \{0, 1\}$ for all $e \in E$. Thus, we get:

$$\begin{array}{lll} \min & \sum_{e \in E} c_e x_e & \\ s.t. & x(\delta(v)) \equiv |\delta(v)| \bmod 2 & \forall v \in V \\ & x_e \in \{0, 1\} & \forall e \in E \end{array}$$

To be more precise, we are looking for some set of edges $J \subseteq E$ such that

$$|J \cap \delta(v)| \equiv |\delta(v)| \bmod 2.$$

Such sets J (the ones that are feasible for this problem) are called **postman sets**.

Remark (More intuition)**.** *The basic idea is to add some parallel edges so that each $v \in V$ has an even degree in G^x. So how many edges can we add for each v? Since we want v to end up with an even degree in G^x, if v is an odd vertex in G, we would add an odd number of edges incident to v; if v is an even vertex in G, we would add an even number of edges incident to v.*

Recall x_e counts the number of additional copies of e in G^x, which corresponds to how many extra times our walk x uses e. Thus, when $|\delta(v)|$ is odd and we add an odd number of copies for each $e \in |\delta(v)|$, the total number of extra copies of edges in $|\delta(v)|$ must also be odd, i.e., $x(\delta(v)) \equiv |\delta(v)| \bmod 2$. The same goes for the vertices of even degrees.

Also, if $c \geq 0$, then we don't want to add any parallel edges incident to v if v is an even degree in G, and we want to add exactly one parallel edge incident to v if v is an odd vertex in G; adding more edges yields sub-optimal solutions. We therefore have $x_e \in \{0, 1\}$.

8.2 T-Joins.

We can solve the min-cost postman tour problem by finding an optimal T-join.

Definition 8.1. Let $T \subseteq V$ with $|T|$ even. A set of edges $J \subseteq E$ is a **T-join** if

$$\forall v \in V : |J \cap \delta(v)| \equiv |T \cap \{v\}| \bmod 2.$$

In other words, $J \subseteq E$ is a T-join when T contains the odd vertices in (V, J).

Remark (Remark on the condition)**.** *First consider the case that*

$$|T \cap \{v\}| \bmod 2 \equiv 1 \equiv |J \cap \delta(v)| \bmod 2.$$

This means that $v \in T$ and v is an odd vertex in (V, J). Now consider

$$|T \cap \{v\}| \bmod 2 \equiv 0 \equiv |J \cap \delta(v)| \bmod 2.$$

Since $|T \cap \{v\}| \leq 1$, we have $|T \cap \{v\}| = 0$. Thus, when v has an even degree in (V, J), it is not in T. It follows that T is the set of vertices with odd degree in (V, J).

Remark (Some more intuition)**.** *Let J be an optimal solution to the min-cost postman tour problem on graph G. By our previous analysis, we know that each edge in G is repeated 0 or 1 time in G^x depending on the parity of (the degree of) v. The graph (V, J) consists of only these additional edges.*

If v is an odd vertex in G, since it becomes an even vertex in G^x, we must have added in total an odd amount of edges to $|\delta(v)|$, i.e., $|J \cap \delta(v)| \bmod 2 \equiv 1$. If v is an even vertex in G, since it remains even in G^x, we must have added in total an even amount of edges to $|\delta(v)|$, i.e., $|J \cap \delta(v)| \bmod 2 \equiv 0$. By definition of a T-join, v has an odd degree in (V, J) iff $v \in T$ and that J is a T-join.

8.3 Minimum-Cost T-Join.

The Minimum Cost T-Join Problem

Problem. *Given $c \in \mathbb{R}^E$, $G = (V, E)$, $T \subseteq V$ with $|T|$ even, find a T-join of G minimizing $c(J) := \sum_{e \in J} c_e$.*[a]

[a] We assume G is connected and $c \geq 0$ in the postman problem. Neither is required for the generic min-cost T-join problem.

Besides postman sets, T-joins are also useful in the following settings.

Remark (Even Sets/Negative Cycles). *Set $T = \emptyset$, i.e., we want every vertex in G to have an even degree. Then a T-join is exactly an **even set**, i.e., a set $A \subseteq E$ such that every vertex in (V, A) has an even degree. A set is even iff it can be decomposed into edge-sets of edge-disjoint circuits. The problem is trivial (pick $\emptyset$) if $c \geq 0$.*[a] *It follows that we can find a negative cycles or determine that none exists by solving an optimal T-join problem.*

[a] Because of the decomposability property mentioned above, $\emptyset$ is optimal iff G has no negative cycles.

Remark (Shortest Paths). *Set $T = \{r, s\}$, i.e., only r and s can have odd degree. The solution to this problem will be the sum of an r, s-path plus a bunch of cycles. In particular, when $c \geq 0$, the optimal does not contain any cycle, so it is a (shortest) r, s-path.*

Min-Cost T-Join with Non-Negative Weights

Let us first focus on the case where $c \geq 0$. In this case, there is always an optimal T-join that is minimal (does not contain any T-join as a strict subset, easy argument with $c \geq 0$). So what do minimal T-joins look like? Intuitively, it is a union of disjoint paths.

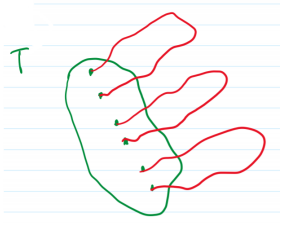


Figure 8.2: Example: A minimal T-join.

We prove the following result first.

Proposition 8.2. *Let J' be a T'-join of G. Then J is a T-join of G iff $J \triangle J'$ is a $(T \triangle T')$-join of G.*

Proof. $\Longrightarrow$: Let $\overline{J} = J \triangle J'$ and $v \in V$. If $v \in T$ and $v \notin T'$, then $v \in (T \triangle T')$, $|J \cap \delta(v)|$ is odd, and $|J' \cap \delta(v)|$ is even. Then $|J \cap \delta(v)| + |J' \cap \delta(v)|$ is odd (parity argument). But $J \triangle J'$ removes an even number of edges (removing the edges in both T and T', i.e., the double-counted ones), which implies $|\overline{J} \cap \delta(v)|$ is odd.

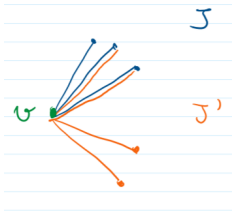


Figure 8.3: The top three edges are double-counted.

The exact same argument works for all other cases. It follows that $|\overline{J} \cap \delta(v)|$ is odd $\iff$ $v \in T \triangle T'$. Note this is precisely the definition of a $(T \triangle T')$-join and thus we are done.

$\Longleftarrow$: We utilize the associativity and commutativity of the symmetric difference operator. Specifically, let us apply the $\Longrightarrow$ argument with $J' := J', T' := T, J := J \triangle J'$, and $T := T \triangle T'$. Then $J' \triangle J \triangle J' = J$ is a $T \triangle T \triangle T' = T$-join. □

Proposition 8.3. *J is a minimal T-join iff it is the union of the edges of $|T|/2$ edge-disjoint paths, joining pairs of vertices in T (all distinct).*

Proof. $\Longleftarrow$ is trivial: we only have this many vertices to use.

$\Longrightarrow$: It suffices to show that J contains such edge set.

Let $u \in T$ and K be the connected component of (V, J) containing u. There is necessarily some $v \in T \setminus u$ such that $v \in K$ since vertices in T have odd degree in (V, J).

Let P be a u, v-path in (V, J). Consider $J' := J \setminus E(P)$. If we show J' is a T'-join where $T' := T \setminus \{u, v\}$, then by induction (same problem, smaller graph) we are done.

Observe $J' \triangle J = E(P)$ is a $\{u, v\}$-join as the only odd degree vertices of $(V, E(P))$ are the endpoints u, v. Thus an application of the next proposition yields the result. □

Solving Min-Cost T-Joins with Non-Negative Weights

Proposition 8.4. *Suppose $c \geq 0$. Then there exists a min-cost T-join that is the union of $|T|/2$ edge-disjoint shortest paths joining vertices of T in pairs (all distinct).*

Proof. Let J be a minimal min-cost T-join. Let P be a u,v-path with $E(P) \subseteq J$; $u,v \in T$. Suppose P is not the shortest, i.e., there exists a u,v-path P' with $c(E(P')) < c(E(P))$. By previous remarks/examples, $E(P)$ and $E(P')$ are $\{u,v\}$-joins. Then $J' := J \triangle E(P) \triangle E(P')$ is a $T \triangle \delta\{u,v\} \triangle \delta\{u,v\} = T$-join with cost

$$\begin{aligned} c(J') &= c(J \setminus E(P)) + c(E(P')) - 2c((J \setminus E(P)) \cap E(P')) \\ &\leq c(J) + c(E(P')) - c(E(P)) < c(J). \end{aligned}$$

Contradiction. □

Lemma 8.5. *Let G' be the complete graph with $V(G)$ and $d(u,v)$ be the cost of a shortest u,v-path. A min-cost T-join when $c \geq 0$ can be found by computing a minimum weight perfect matching in G' with weights $c_{uv} = d(u,v)$.*

Proof. Let M be the minimum weight perfect matching in G'. Let $\{u_i, v_i\}_{i=1}^{|T|/2}$ be the edges in M. Put P_i as the shortest path in G for $1 \leq i \leq |T|/2$. Then $E(P_1) \triangle \cdots \triangle E(P_{|T|/2})$ is a T-join of cost $\leq \sum_{i=1}^{|T|/2} d(u_i, v_i)$. By the previous proposition, any minimal T-join corresponds to a matching in G' and has cost at least that sum. Optimality follows. □

Min-Cost T-Join for Arbitrary Costs

Let $N = \{e \in E : c_e < 0\}$. Let $T' := \{v \in V : v \text{ has odd degree in } (V,N)\}$. Then N is a T'-join. By previous results, we know J is a T-join iff $J \triangle N$ is a $(T \triangle T')$-join. Observe[2]

$$\begin{aligned} c(J) &= c(J \setminus N) + c(J \cap N) \\ &= \Big[c(J \setminus N) - c(N \setminus J)\Big] + \Big[c(N \setminus J) + c(J \cap N)\Big] \\ &= \sum_{e \in J \triangle N} |c_e| + c(N) \end{aligned}$$

Since $c(N)$ is a constant, in order to minimize $c(J)$, it suffices to find a min cost $(T \triangle T')$-join wrt costs $c' := |c| \in \mathbb{R}^E$. This motivates the following algorithm:

1. Find a min-cost $(T \triangle T')$-join J^* wrt $c' := |c| \in \mathbb{R}^E$.
2. Output the min-cost T-join $J^* \triangle N$.

[2] Every edge in $N \setminus J$ has a negative cost, so $c(J \setminus N) - c(N \setminus J)$ is essentially summing up the absolute values of the costs of edges in $J \triangle N$.

8.4 LP Formulations for Min-Cost T-Joins.

In this section, we solve the min-cost T-join problem with linear programmming.

T-Odd Sets and T-Cuts

Definition 8.6. A set $S \subseteq V$ is **T-odd** if $|S \cap T|$ is odd. If S is T-odd, then $\delta(S)$ is called a **T-cut**.

To see how this definition is related to T-joins, let $S \subseteq V$ be T-odd and J be a T-join. By definition of a T-join, every $v \in (S \cap T)$ has an odd degree in (V, J). If $J \cap \delta(S) = \emptyset$, then the subgraph of (V, J) induced by S (that is, the graph obtained by removing vertices in $V \setminus S$ and any edge using these vertices) has an odd number of odd degree vertices, which is impossible. Thus, any T-join must cross this T-cut at least once, i.e.,

$$|J \cap \delta(S)| \geq 1.$$

Let $\mathcal{O}$ be the set of T-odd sets. We can formulate an LP based on the observation above:

$$\begin{aligned} (P) := \min \quad & x^T c \\ s.t. \quad & x(\delta(s)) \geq 1 \quad \forall S \in \mathcal{O} \\ & x \geq 0. \end{aligned}$$

Proof of Correctness for (P)

Theorem 8.7. *Let $G = (V, E)$, $T \subseteq V$ with even cardinality, and $c \in \mathbb{R}^E_{\geq 0}$. Then the cost of a min-cost T-join wrt c is equal to the optimal value of (P).*

Proof. Let J^* be a optimal T-join. We have argued that the algorithm gives us a feasible solution to the LP, so $c(J^*) \geq z^*$ where z^* is the optimal value of the LP.

Consider the primal-dual pair below:

$$\begin{aligned} (P) := \min \quad & \textstyle\sum_{e \in E} c_e x_e \\ s.t. \quad & x(\delta(S)) \geq 1 \quad \forall S \in \mathcal{O} \\ & x \geq 0 \\ (D) := \max \quad & \textstyle\sum_{S \in \mathcal{O}} \alpha_S \\ s.t. \quad & \textstyle\sum_{S \in \mathcal{O} : e \in \delta(S)} \alpha_S \leq c_e \quad \forall e \in E \\ & \alpha \geq 0 \end{aligned}$$

Case 1: $T = V$. (Note this implies that $|V|$ is even as we cannot have an odd number of odd vertices in a graph.) Let $G' = (T = V, E')$ be the complete graph used to get a perfect matching with costs d that solves the min-cost T-join problem. As before, define [3]

$$\mathcal{O}_M := \{A \subseteq V : |A| \geq 3, |A| \text{ odd}\}.$$

Here's the primal-dual pair for the minimum weight perfect matching:

$$\begin{aligned}
(P_M) := \min \quad & \textstyle\sum_{uv \in E'} d(u,v) w_{uv} & \\
s.t. \quad & w(\delta(v)) = 1 & \forall v \in V \\
& w(\delta(A)) \geq 1 & \forall A \in \mathcal{O}_M \\
& w \geq 0 &
\end{aligned}$$

$$\begin{aligned}
(D_M) := \max \quad & \textstyle\sum_{v \in V} \beta_v + \sum_{A \in \mathcal{O}_M} \gamma_A & \\
s.t. \quad & \beta_u + \beta_v + \textstyle\sum_{A \in \mathcal{O}_M : uv \in \delta(A)} \gamma_A \leq d(u,v) & \forall u, v \in E' \\
& \gamma_A \geq 0 &
\end{aligned}$$

We've shown that $c(J^*) = OPT_{P_M} = OPT_{D_M}$ (Lemma 8.5). Let us transform solutions to (D_M) to solutions to (D). The idea is as follows. Note that $E \subseteq E'$ and $d(u,v) \leq c_{uv}$ for all $uv \in E$. From an optimal solution to (D_M), we can build a solution to (D) of the same cost and vice versa.

The only caveat is that some of the β variables can be negative. However, it can be shown that they are in fact non-negative. We omit the proof.

Case 2: General T. Let $G = (V, E)$, $T \subseteq V$, $|T|$ even, $c \in \mathbb{R}^E_{\geq 0}$. We define $\hat{G}$ to have vertex set $\hat{V} := V \cup \{\hat{v} : v \in V \setminus T\}$ and edge set $\hat{E} := E \cup \{v\hat{v} : v \in V \setminus T\}$, where $c(v\hat{v}) = 0$ for all $v \in V \setminus T$. Let $\hat{T} = \hat{V}$ and $\hat{J}$ be a min-cost $\hat{T}$-join of $\hat{G}$. We claim that we can construct a min-cost T-join J from $\hat{J}$.

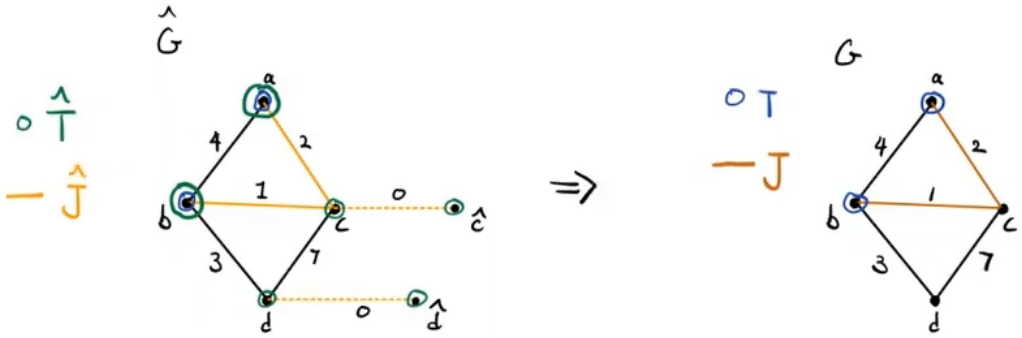


Figure 8.4: Construction of $\hat{G}$.

[3] Check the chapter on weighted matchings if you find this unfamiliar.

For every $\hat{v}$ that is newly added, there is exactly one edge $v\hat{v}$ incident to $\hat{v}$ in $\hat{G}$. Since $\hat{J}$ is a min-cost $\hat{T}$-join of $\hat{G}$, it is a union of $|\hat{T}|/2$ edge-disjoint paths joining pairs of vertices in $\hat{T} = \hat{V}$. Every vertex in $\hat{V}$ is being joined by a path in $\hat{J}$ to another vertex in $\hat{V}$. In particular, $v\hat{v}$ must belong to the path in $\hat{J}$ with $\hat{v}$ as an endpoint. Thus, $v\hat{v} \in \hat{J}$ for every $v \in V \setminus T$.

We define $J := \hat{J} \setminus \{v\hat{v} : v \in V \setminus T\}$. Clearly, $J \subseteq E$. For every $u \in T$, no edge incident to u are removed from $\hat{J}$. Thus, $|\delta_G(u) \cap J| = |\delta_{\hat{G}}(u) \cap \hat{J}|$, which is odd. For every $v \in V \setminus T$, exactly one edge $v\hat{v}$ is incident to v is removed from $\hat{J}$. Thus, $|\delta_G(v) \cap J| = |\delta_{\hat{G}}(v) \cap \hat{J}| - 1$, which is even. Hence, J is a T-join. Since only edges of 0 costs are removed from $\hat{J}$, we have $c(J) = c(\hat{J})$.

Now we have a T-join, let us prove it's minimal. Suppose J is not a min-cost T-join. Let J' be a T-join with $c(J') < c(J)$. Let $\hat{J}' := J' \cup \{v\hat{v} : v \in V \setminus T\}$. For every $u \in T$, no edge incident to u are added to $\hat{J}'$, so $|\delta_{\hat{G}}(u) \cap \hat{J}'| = |\delta_G(u) \cap J'|$, which is odd. For every $v \in V \setminus T$, $v\hat{v}$ is the only edge added to $\hat{J}'$ incident to v or $\hat{v}$, so $|\delta_{\hat{G}}(v) \cap \hat{J}'| = \delta_G(v) \cap J'| + 1$, which is odd. We also have $|\delta_{\hat{G}}(\hat{v}) \cap \hat{J}'| = 1$, which is odd. It follows that $\hat{J}'$ is a $\hat{T}$-join. However, we only added edges of 0 costs to $\hat{J}'$, we get

$$c(\hat{J}') = c(J') < c(J) = c(\hat{J}),$$

which contradicts the fact that $\hat{J}$ is a min-cost $\hat{T}$-join. It follows that J must be a min-cost T-join.

This concludes the proof. □

Part IV

Flows

9 Flows

9.1 Max-Flow Min-Cut.

Maximum Flow Problem

Definition 9.1. Given a digraph $D = (V, A)$ and $r, s \in V$ (often referred to as the **source** and the **sink**), we say $x \in \mathbb{R}^A$ is an r, s**-flow** if the following holds (often referred to as the **flow conservation constraint**):

$$\forall v \in V \setminus \{r, s\} : x(\delta^-(v)) - x(\delta^+(v)) = 0,$$

where

- $\delta^-(S) := \{uv \in A : u \notin S, v \in S\}$ is the set of arcs entering S, and
- $\delta^+(S) := \{uv \in A : u \in S, v \notin S\}$ is the set of arcs leaving S.

We call their difference $f_x(v) := x(\delta^-(v)) - x(\delta^+(v))$ the **net flow into** v.

Definition 9.2. Given **lower** and **upper bounds** (or **capacities**) $\ell \leq u \in \mathbb{R}^A$, an r, s-flow x is **feasible** if

$$\forall a \in A : \ell_a \leq x_a \leq u_a.$$

The **value** of a feasible r, s-flow x is given by $f_x(s)$, the *total flow into the sink*.

For simplicity, assume the lower bound of the flow is 0 and there is no arc entering the source or leaving the sink, i.e., $\ell = \vec{0}$ and $\delta^-(r) = \delta^+(s) = \varnothing$.

Problem.

- ***Maximum Flow:*** *Given $D = (V, A)$, source and sink $r, s \in V$, and $\ell, u \in \mathbb{R}^A$, find a feasible r, s-flow $x \in \mathbb{R}^A$ of maximum value.*
- ***Maximum Integral Flow:*** *Given $D = (V, A)$, source and sink $r, s \in V$, and $\ell, u \in \mathbb{R}^A$, find a feasible r, s-flow $x \in \mathbb{Z}^A$ of maximum value.*

Basic Results

Definition 9.3. For $R \subseteq V$, we call the set of arcs leaving R, $\delta^+(R)$, a (directed) **cut**. Moreover, if $r \in R$ and $s \notin R$, we say that $\delta^+(R)$ is an r, s**-cut**.

Proposition 9.4. *If x is a feasible r,s-flow and $\delta^+(R)$ is an r,s-cut, then*

$$x(\delta^+(R)) - x(\delta^-(R)) = f_x(s).$$

Proof. Since x is feasible, $f_x(v) = x(\delta^+(v)) - x(\delta^-(v)) = 0$ for all $V \setminus \{r,s\}$. Then

$$\begin{aligned} x(\delta^+(R)) - x(\delta^-(R)) &= \sum_{v \in R} \left[x(\delta^+(v)) - x(\delta^-(v))\right] \\ &= f_x(s) + \sum_{v \in R \setminus \{s\}} \left[x(\delta^+(v)) - x(\delta^-(v))\right] \\ &= f_x(s). \end{aligned}$$

□

Corollary 9.5. *If x is a feasible r,s-flow and $\delta^+(R)$ is any r,s-cut, then*

$$f_x(s) \leq u(\delta^+(R)).$$

In words, the total in-flow of s cannot exceed the out-flow of any r,s-cut.

Proof. Observe $f_x(s) = x(\delta^+(R)) - x(\delta^-(R)) \leq x(\delta^+(R)) \leq u(\delta^+(R))$. □

Augmenting Paths

Suppose P is a r,s-path in D that uses some arcs in "forward" direction, some in "backward" direction (i.e., P is not a real dipath).

Definition 9.6. We say that P is **x-incrementing** if for each edge $a \in E(P)$,

- $x_a < u_a$ if a appears in forward direction, and
- $x_a > 0$ if a appears in backward direction.

P is **x-augmenting** if it is an x-incrementing r,s-path.

Below is an x-augmenting path. Observe it satisfies the condition such that *all forward arcs are not at capacity* and *all backward arcs have a positive flow*.

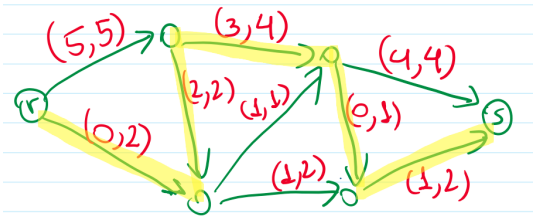


Figure 9.1: The highlighted path is an x-augmenting path. Arcs are labelled with (flow, capacity)

The key observation is that, ***if there exists an x-augmenting path, then our current flow x is NOT a maximum flow***, because we can ***increase x_a by ε for all forward arcs*** and ***decrease x_a by ε for all backward arcs***, which increases $f_x(s)$ by ε. For example, we can "push" one extra unit of flow from r to s via this path:

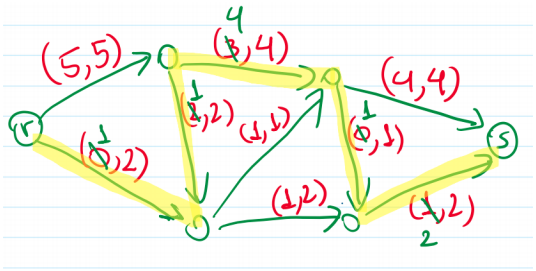


Figure 9.2: Augmenting the flow.

Max-Flow Min-Cut

Theorem 9.7 (Max-Flow Min-Cut)**.** *If there exists a max flow, then*

$$\max\{f_x(s) : x \text{ is a feasible } r,s\text{-flow}\} = \min\{u(\delta^+(R)) : \delta^+(R) \text{ is an } r,s\text{-cut}\}.$$

In words, there exists a cut whose out-flow (cut value) equals the max flow into s.

Proof. Let x be a max r,s-flow and R be the set of vertices reachable by an x-incrementing path starting at r. Note that $r \in R$ and $s \notin R$, so $\delta^+(R)$ is an r,s-cut. For all $vw \in \delta^+(R)$, we must have $x_{vw} = u_{vw}$, as otherwise we can push flow to w which means $w \in R$. Likewise, any $vw \in \delta^-(R)$ must have $x_{vw} = 0$ or we could decrease the flow on rw which makes $v \in R$. It follows that $f_x(s) = x(\delta^+(R)) - x(\delta^-(R)) = u(\delta^+(R)) - 0 = u(\delta^+(R))$. □

Corollary 9.8. *A feasible r,s-flow x is maximum iff there is no x-augmenting path.*

Proof. Easy. Omitted.

Theorem 9.9. *If $u \in \mathbb{Z}_+^A$ and there is a maximum flow, then there is an integral maximum flow.*

Proof. First remark that the set of integral flows are non-empty and the flow values of integral flows are bounded from above. Thus, a maximum integral flow exists. Let x be a maximum integral flow but not a maximum flow. Then there is a x-augmenting r,s-path. But the residual capacities are integral and thus at least 1. This contradicts the assumption that x was a maximum integral flow. □

9.2 *Maximum Flow Algorithms: Ford-Fulkerson and Edmonds-Karp.*

Ford-Fulkerson

The idea is to construct an auxiliary graph $D_x = (V, A_x)$, where

- $vw \in A_x$ if $vw \in A$ and $x_{vw} < u_{vw}$, and
- $vw \in A_x$ if $wv \in A$ and $x_{wv} > 0$.

This reduces the problem of finding an maximum flow in D to finding r, s-dipaths in D_x.

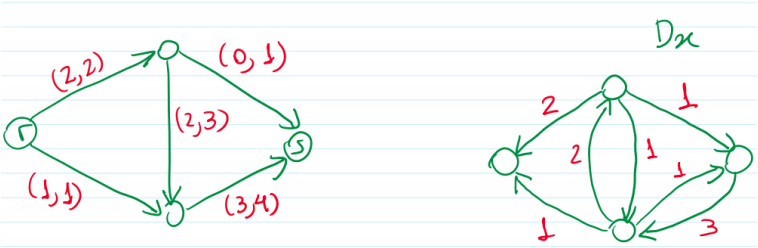


Figure 9.3: Digraph D with flow x (left) and the auxiliary graph D_x (right).

Consider the middle arc of D. It has a capacity of 3 and we are using 2 units. Thus, we have the option to decrease at most 2 units of flow and to send at most 1 extra unit of flow along this arc. These two options correspond to the two middle arcs in D_x.

Observe ***there exists an x-augmenting path iff there exists an r, s-dipath in D_x.*** Moreover, if P is such a path, then $f_x(s)$ can be increased by the smallest **residual capacity** in P. An augmenting path can be found in $O(m) = O(|A|)$ time. But how many times do we need to augment a flow before reaching a maximum one? Recall from CO-351, if we don't choose our strategy carefully, the following graph could require $2M$ iterations.

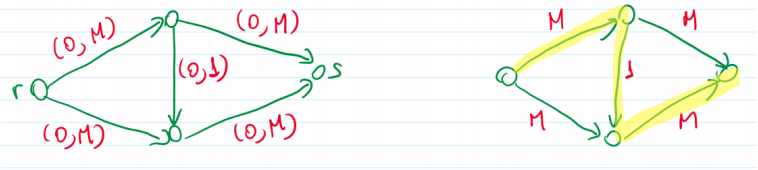


Figure 9.4: Naive way of augmenting the flow.

Remark. *Will FF always terminate? If c is integral, then at each step we increase our flow by at least 1, so it terminates by induction. If c is rational, we can multiply c by the GCD of all denominators and make it integral. If c is irrational, then it is possible that it does not terminate. The runtime for FF is* ***pseudo-polynomial****, i.e., it is a polynomial in the numeric values of your input.*

Edmonds-Karp

Edmonds-Karp is an implementation of Ford-Fulkerson which has a guaranteed $O(|V||E^2|)$ time complexity. The idea is simple: always choose the shortest x-augmenting path in D_x.

Theorem 9.10 (Edmonds-Karp)**.** *If P is chosen to be the shortest (wrt to number of arcs) r,s-dipath in the auxiliary graph D_x, then there are at most nm augmentations.*

Proof. Let $d_x(v,w)$ be the length of shortest v,w-path in D_x, $P = v_0,\dots,v_k$ be the shortest r,s-path in D_x, and x' be the feasible r,s-flow obtained after augmenting x with P. We break the proofs into the following two lemmas. First, the length of the shortest r,v-path and the length of the shortest v,s-path cannot go down after an augmentation.

Claim. *We have* $\forall v \in V : d_{x'}(r,v) \geq d_x(r,v),\ d_{x'}(v,s) \geq d_x(v,s)$. [a]

[a] If no u,v-path exists in D_x, we set $d_x(u,v) = \infty$.

Proof (Claim). Suppose $\exists v$ such that $d_{x'}(r,v) < d_x(r,v)$. Choose such v with $d_{x'}(r,v)$ smallest possible (i.e., every w with a shorter distance from r satisfies $d_{x'}(r,w) \geq d_x(r,w)$). Let P' be the r,v-path of $D_{x'}$ with length $d_{x'}(r,v)$. This path has at least 1 arc as otherwise $v = r$ and $d_{x'}(r,r) = d_x(r,r) = 0$. Let w be the vertex immediately before v in P'. Then

$$\begin{aligned} d_x(r,v) &> d_{x'}(r,v) && \text{assumption} \\ &= d_{x'}(r,w) + 1 && \text{definition of } w \\ &\geq d_x(r,w) + 1 && \text{choice of } v \end{aligned}$$

Suppose wv exists in the previous residual graph D_x. Then $d_x(r,v) \leq d_x(r,w) + 1$ because we went from r to w and then w to v. This contradicts what we just proved. Therefore, we must have $wv \notin A(D_x)$. But $wv \in P'$, so $wv \in A(D_{x'})$, which implies wv or vw is an arc in P (shortest r,s-path in D_x) as these are the arcs with flows changed by the augmentation.

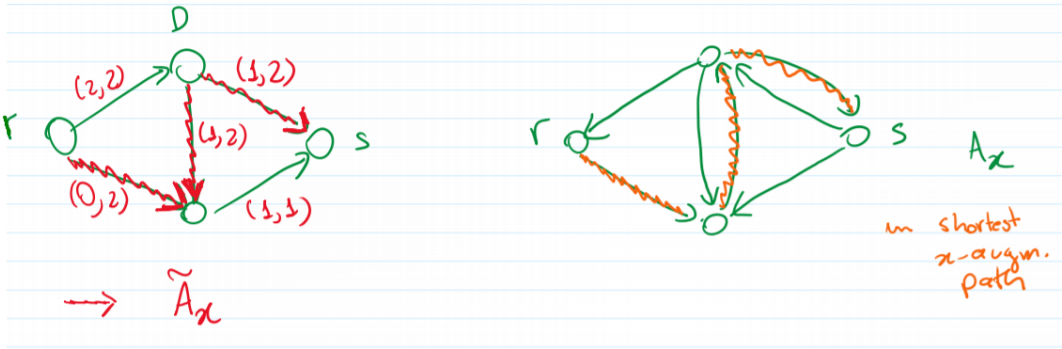
Since $E(P) \subseteq A(D_x)$ and $wv \notin A(D_x)$, we have $vw \in E(P)$, so $v = v_{i-1}$ and $w = v_i$ for some $i = 1,\dots,k$. Combined with $d_x(r,v) \geq d_x(r,w) + 1$, we have $d_x(r,v_{i-1}) \geq d_x(r,v_i) + 1$. But this cannot happen because $d_x(r,v_i)$ is the length of the shortest r,v_i-path that uses v_{i-1} as an intermediate node. In particular, since P visits v then w, we have $d_x(r,w) = d_x(r,w) - 1$.

The second part can be shown with an analogous proof. ∎

The claim above shows that the algorithm works in $\leq n-1$ stages (in each stage $d_x(r,s)$ remains constant). We now show that in each stage we do a limited number of operations.

Claim. *Define* $\tilde{A}_x := \{wv \in A : \ wv \text{ or } vw \text{ is in a shortest } x\text{-augmentation path}\}$. *If* $d_{x'}(r,s) = d_x(r,s)$, *then* $\tilde{A}_{x'} \subsetneq \tilde{A}_x$.

In words, if the distance from r to s didn't change, then the set of arcs $\tilde{A}_{x'}$ is a strict subset of $\tilde{A}_x$. Consider the set of red arcs below. They correspond to the orange x-augmenting path on the right. Thus, $\tilde{A}_x$ consists of these three arcs. We will show that the number of arcs in this set keeps decreasing as we augment the flow x.



Proof (Claim). Let $k = d_x(r,s)$ and fix $vw \in \tilde{A}_x$. If vw is in the shortest r,s-path in $D_{x'}$, then

$$d_{x'}(r,v) = i-1, \ d_{x'}(w,s) = k-i \implies d_{x'}(r,v) + d_{x'}(w,s) = k-1.$$

By the previous claim, $d_x(r,v) + d_x(w,s) \leq k-1$ because d cannot decrease after an augmentation. Suppose to the contrary that $vw \notin \tilde{A}_x$, which means neither vw nor wv were in the shortest x-augmenting path, so the flow on vw remains the same, i.e, $x_{vw} = x'_{vw}$. This implies $vw \in A_x$. But then there exists an r,s-path of length k, contradiction. Thus, $vw \in \tilde{A}_x$ and $\tilde{A}_{x'} \subseteq \tilde{A}_x$. A similar argument shows that wv is in a shortest r,s-path in $D_{x'}$.

We now argue for strict containment. Let P be the path used to change x to x'. We know there exists $vw \in A$ such that $vw \in P \wedge x'_{vw} = u_{vw}$ or $wv \in P \wedge x'_{uv} = 0$ (the two possible modifications of the flow). We show this particular arc vw is no longer in $\tilde{A}_{x'}$.

Focus on Case 1 first. Since vw is on the shortest path, we have $d_x(r,v) = i-1$, $d_x(w,s) = k-i$, $vw \in \tilde{A}_x$, and $vw \notin D_{x'}$. An x'-augmenting path cannot use wv, so if $vw \in \tilde{A}_{x'}$, there exists an x'-augmenting path using wv. But then

$$d_{x'}(r,w) + d_{x'}(v,s) \geq d_x(r,w) + d_x(v,s) = (i-1+1) + (k-i+1) = k+1$$

(the new r,w-path use v in the middle). But the length of the shortest r,s-dipath in $D_{x'}$ using wv is not a shortest r,s-path. Thus, $vw \notin \tilde{A}_{x'}$. The second case is similar. ■

Going back to the main proof. The second claim shows that each stage has $\leq m$ iterations. Therefore, the total run time is bounded by $O(n \cdot m)$ which is polynomial in input size. □

9.3 *Applications of Flows/Cuts.*

Bipartite Matching

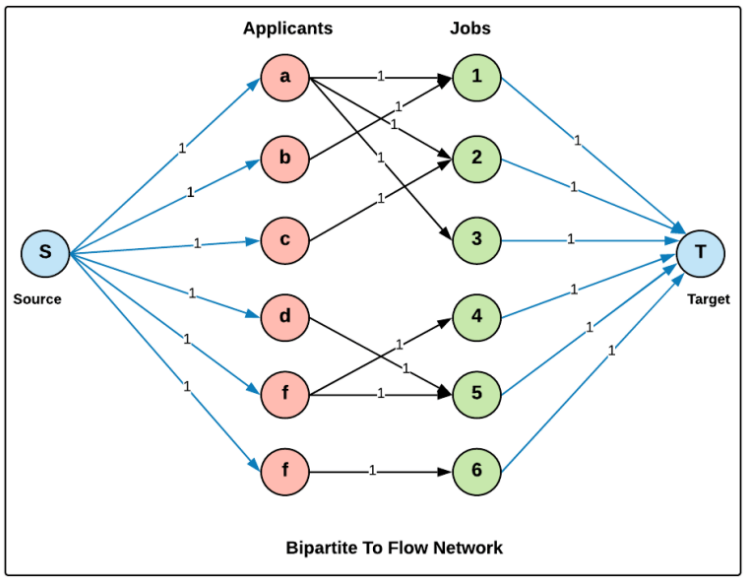


Figure 9.5: Reducing bipartite matching to maximum flow.

- Direct all edges from one bipartition to the other.
- Add two auxiliary nodes as source and sink.
- Add arcs ra for each $a \in A$ an arcs bs for each $b \in B$.
- Set capacity for all arcs equal to 1.

The max-flow algorithm (FF/EK) returns an integral maximum flow, which corresponds to a maximum cardinality matching. Note it's not hard to show Konig's theorem ($\tau(G) = \nu(G)$ when G is bipartite) with this set up.

Flow Feasibility

Given $D = (V, A)$, $u \in \mathbb{R}_+^A$ and $b \in \mathbb{R}^V$ such that $b(V) = 0$.

Problem. *Determine if there exists a flow $x \in \mathbb{R}^A$ such that*

- $\forall a \in A : 0 \leq x_a \leq u_a$,
- $\forall v \in V : f_x(v) = b_v$.

Using a similar intuition from CO-351 transshipment problem, let us treat nodes v with $b_v < 0$ as "supply nodes" and treat nodes v with $b_v > 0$ as "demand nodes". Our goal is to "ship" all units of flows from the "demand nodes" to "supply nodes" such that all supplies are exhausted, all demands are satisfied, and no flow capacity constraint is violated.

Consider the following reduction. Create new vertices r, s. For each v where $b_v < 0$, add arc rv with capacity $u_{rv} = -b_v$. For each v where $b_v > 0$, add arc vs with capacity $u_{rv} = b_v$. Intuitively, we added an additional source and sink for the entire network. In particular, the total flow into all demand nodes is equal to the total in-flow of auxiliary sink s. Since $b(V) = 0$, i.e., total supply equals total demand, if we manage to satisfy all demands and no capacity constraint is broken, then we've found a feasible flow. In other words, a feasible flow exists iff the max flow into s in the auxiliary graph attains

$$\sum_{v:b_v>0} b_v.$$

Observe for any $S \subseteq V$, $S \cup \{r\}$ is an r, s-cut, so the max flow attains $\sum_{v:b_v>0} b_v$ iff

$$\forall S \subseteq V : u(\delta^+(S \cup \{r\})) \geq \sum_{v:b_v>0} b_v.$$

We can rewrite the cut value as

$$u(\delta^+(S \cup \{r\})) = \sum_{v \in S:b_v>0} b_v + \sum_{v \notin S:b_v<0} (-b_v) + u(\delta_D^+(S)).$$

- $\sum_{v \in S:b_v>0} b_v$ is the total capacity for edges going from S into s.
- $\sum_{v \notin S:b_v<0} (-b_v)$ is the total capacity for edges going from r to nodes not in S.
- $u(\delta_D^+(S))$ is cut value of S in the original graph D.

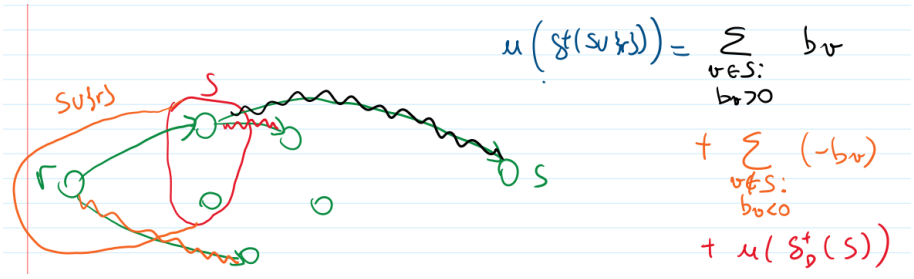


Figure 9.6: Flow feasibility.

Plugging this in, we have for all $S \subseteq V$,

$$u(\delta^+(S \cup \{r\})) = \sum_{v \in S:b_v>0} b_v + \sum_{v \notin S:b_v<0} (-b_v) + u(\delta_D^+(S)) \geq \sum_{v:b_v>0} b_v$$

$$u(\delta_D^+(S)) \geq \sum_{v \notin S:b_v>0} b_v + \sum_{v \notin S:b_v<0} b_v = \sum_{v \notin S} b_v = b(\overline{S})$$

Flipping around S and $\overline{S}$ in the inequality above, we have $u(\delta_D^+(\overline{S})) \geq b(S)$. In conclusion, there exists a feasible flow iff the maximum r, s-flow has value $\sum_{v:b_v>0} b_v$ iff

$$\forall S \subseteq V : b(S) \leq u(\delta_D^+(S)).$$

9.4 *Undirected Minimum Cut: Gomory-Hu Trees.*

Undirected Minimum Cut

> **Problem.** *Given $G = (V, E)$ undirected, $u \in \mathbb{R}^+_{\geq 0}$, find $\emptyset \neq S \subsetneq V$ minimizing the cut $u(\delta(S))$.*

We can brute-force the solution by replacing each undirected edge with a forward and backward arc with the same capacity and computing maximum v, w-flow for all $v, w \in V$. This requires $O(n^2)$ max-flow computations.

Gomory-Hu Trees: Construction

The **Gomory-Hu tree** of an undirected graph with capacities is a weighted tree that represents the min s,t-cuts for all s,t-pairs in the graph. It can be constructed in $|V| - 1$ max-flow computations.

Let $\lambda(G)$ be the weight of a minimum cut and $\lambda(G, v, w)$ be the weight of a minimum v, w-cut, i.e., $v \in S$ and $w \notin S$. Pick $r, s \in V$ arbitrary and compute the min r, s-cut. Let $R, S \subseteq V$ be its "shores", i.e.,

$$\lambda(G, r, s) = u(\delta(R)), \quad S := V \setminus R.$$

We store results in a tree $T = (\{R, S\}, \{RS\})$ where the edge RS has label $\lambda(G, v, w)$. This edge represents (the weight of) a minimum r, s-cut.

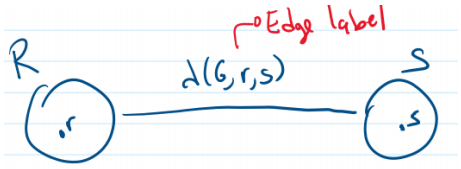


Figure 9.7: Later

In general, if our current Gomory-Hu tree T has vertices corresponding only to singletons, then we are done. Otherwise, suppose there is a tree node $A \in V(T)$ which corresponds to at least 2 vertices in $V(G)$.

Pick $a_1, a_2 \in A$. Let us find the min a_1, a_2-cut in the original graph. The idea is as follows. Suppose we take away the vertex set A. Then we are left with four connected components (in the original graph), namely B, C, D, E. We contract these components of $T \setminus A$ and let G_A be the resulting graph.

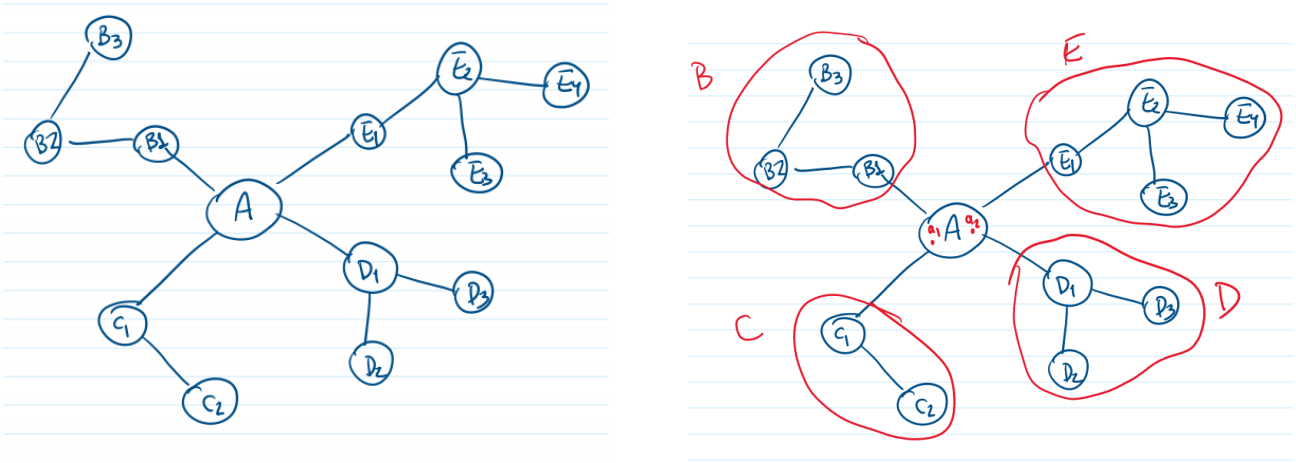


Figure 9.8: Left: current Gomory-Hu tree. Right: $a_1, a_2 \in A$; connected components of $T \setminus A$.

Compute the min a_1, a_2 cut in G_A and name it $\delta(X)$, so that $\lambda(G_A, a_1, a_2) = u(\delta(X))$. Define $A_1 := X \cap A$ and $A_2 := \bar{X} \cap A$. Split A into A_1, A_2 within T, and label the edge $A_1A_2 \in E(T)$ with $\lambda(G_A, a_1, a_2)$.

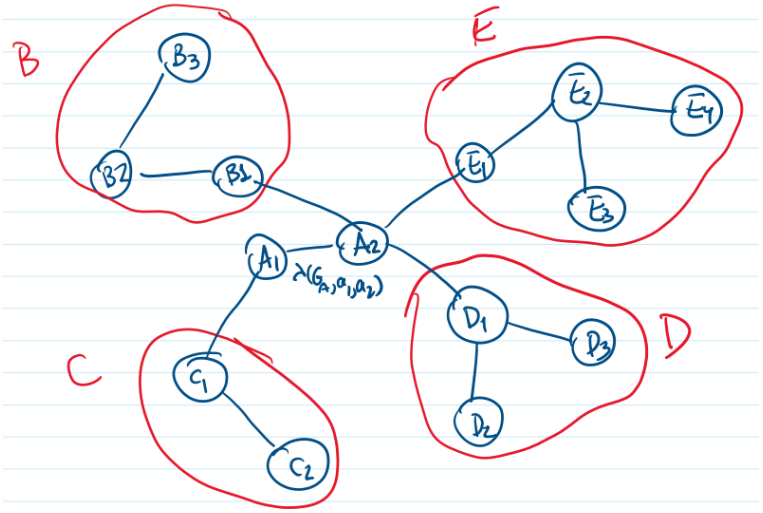


Figure 9.9: Split A into A_1, A_2 within T. Connect the components.

As for each connected components of $T \setminus A$, the corresponding contracted super-vertex belongs either to X or $\bar{X}$ in G_A. Connect each component to either A_1 or A_2 accordingly. For example, we have C_1 connected to A_1, which implies that the super-vertex $v_C \in V(G_A)$ representing C was in X. Similarly, D_1 is connected to A_2, which implies that the super-vertex $v_D \in V(G_A)$ representing D was in $\bar{X}$. Note C_1 and D_1 were originally connected to A, which is the reason why they are the "representatives" of each component.

Gomory-Hu Tree: Example

Let G be given as follows. We start off with a single node that represents the entire $V(G)$.

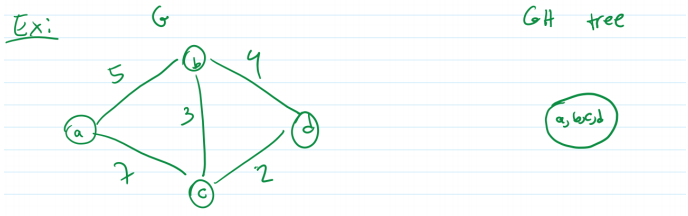


Figure 9.10: Initialize G_H with a single node.

Pick two vertices arbitrarily, say a and b, in the vertex $\{a, b, c, d\} \in V(G_H)$. Since we only have 1 vertex in G_H, there is nothing to contract. The red line represents a min a, b-cut in G with a weight of 10, where a, c are in one shore and b, d are in the other shore.

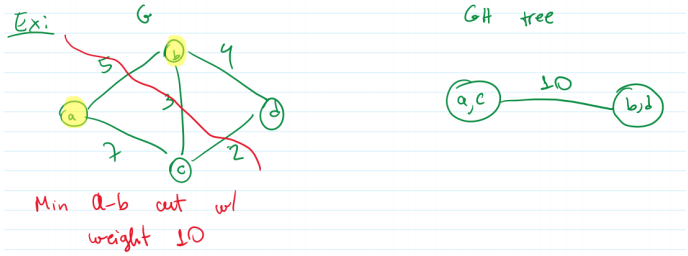


Figure 9.11: Compute a min a, b-cut in G and update the tree.

Observe the vertex $\{a, c\} \in V(G_H)$ corresponds to 2 vertices in G, so we want to split it. Pick $a, c \in \{a, c\}$. If we take away a and c, we are left with a component with vertex set $\{b, d\}$. Contract this component to obtain $G_{\{a,c\}}$. The red line represents a min a, c-cut in G with a weight of 12, where a is in one shore and c, b, d are in the other shore. We split $\{a, c\}$ and connect $\{b, d\}$ to c because they are on the same side.

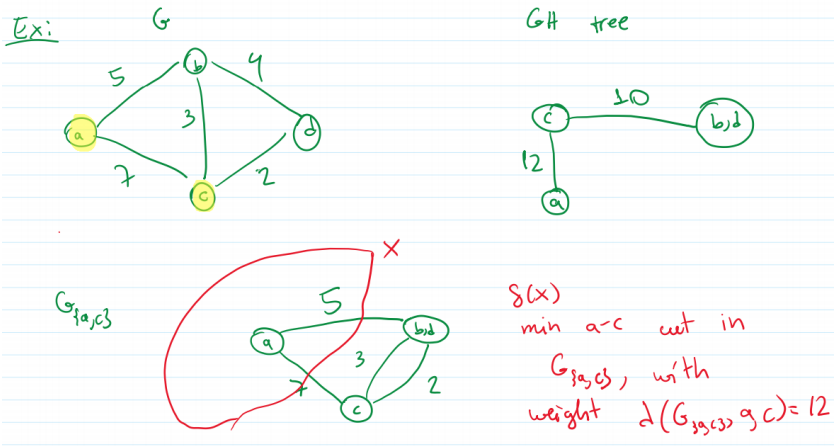


Figure 9.12: Compute a min a, c-cut in G and update the tree.

It remains to split $\{b, d\}$. If we take away b and d, we are left with a connected component with vertex set $\{a, c\}$. Contract this component to obtain $G_{\{b,d\}}$. The red line represents a min b, d-cut in G with a weight of 6, where d is in one shore and b, a, c are in the other shore. We split $\{b, d\}$ and connect $\{b\}$ to $\{c\}$ because they are on the same side.

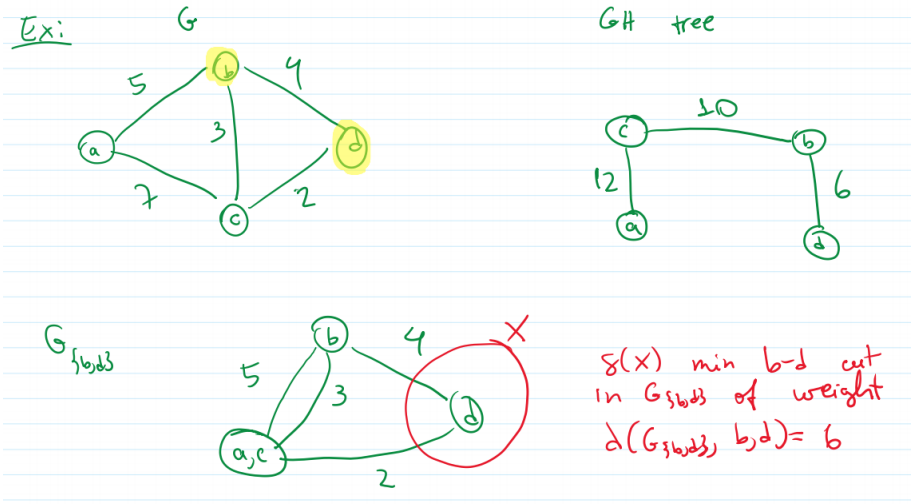


Figure 9.13: Compute a min b, d-cut in G and update the tree.

Since each tree node represents a single vertex in G, we are done.

Gomory-Hu Tree: Reconstruct Solutions

How do we get min cuts from this Gomory-Hu tree T? Suppose we are interested in a min x, y-cut. We just need to look at the min cost edge e^* in $T_{x,y}$, the unique x, y-edge in T.

For example, the min c, d-cut has a weight of 6 (edge bd in T), which corresponds to the cut $\{cd, bd\}$ in G.

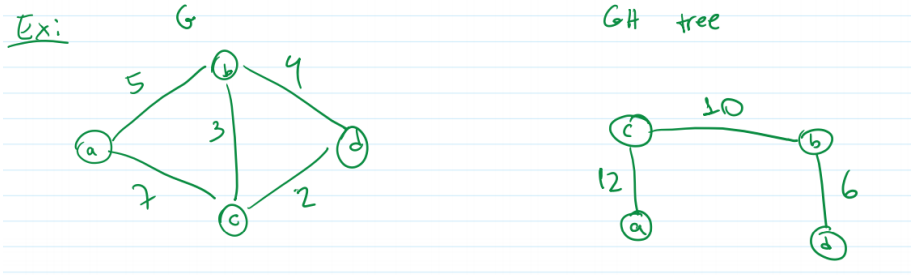


Figure 9.14: Find min cuts from Gomory-Hu trees.

Thus, with $n - 1$ max-flow computations, we get a nice data structure to store all v, w-cuts.

Gomory-Hu Tree: Correctness

Let us first show that the function $u(\delta(A))$ is in fact submodular.

Lemma 9.11. $\forall A, B \subseteq V : u(\delta(A)) + u(\delta(B)) \geq u(\delta(A \cup B)) + u(\delta(A \cap B))$.

Proof. Fix $A, B \subseteq V$. Let us carefully decompose edge sets in consideration.

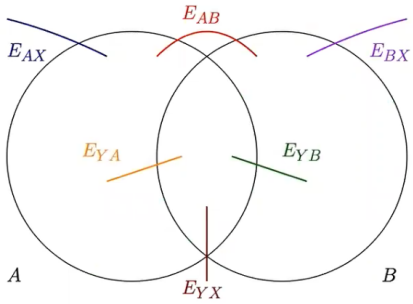


Figure 9.15: Problem setting.

Consider the following disjoint edge-sets.

$$\begin{aligned}
E_{AB} &:= \{ab \in E : a \in A \setminus B, b \in B \setminus A\} \\
E_{AX} &:= \{ax \in E : a \in A \setminus B, x \in V \setminus (A \cup B)\} \\
E_{BX} &:= \{bx \in E : b \in B \setminus A, x \in V \setminus (A \cup B)\} \\
E_{YX} &:= \{yx \in E : y \in A \cap B, x \in V \setminus (A \cup B)\} \\
E_{YB} &:= \{yb \in E : y \in A \cap B, b \in B \setminus A\} \\
E_{YA} &:= \{ya \in E : y \in A \cap B, a \in A \setminus B\}
\end{aligned}$$

Note these sets are pairwise disjoint since $V \setminus (A \cup B)$, $B \setminus A$, $A \setminus B$, and $A \cap B$ are disjoint sets and we consider the six edge sets grouped by endpoints in the disjoint sets. Note that

$$\begin{aligned}
\delta(A) &= E_{AX} \cup E_{YX} \cup E_{YB} \cup E_{AB} \\
\delta(A) &= E_{BX} \cup E_{YX} \cup E_{YA} \cup E_{AB} \\
\delta(A \cup B) &= E_{AX} \cup E_{BX} \cup E_{YX} \\
\delta(A \cap B) &= E_{YA} \cup E_{YB} \cup E_{YX}
\end{aligned}$$

It follows that

$$u(\delta(A)) + u(\delta(B)) - u(\delta(A \cup B)) - u(\delta(A \cap B)) = 2u(E_{AB}) \geq 0$$

as desired. □

Lemma 9.12. *Let $\delta(S)$ be a min r,s-cut and let $v,w \in S$. Then there exists a min v,w-cut $\delta(T)$ such that $T \subseteq S$.*

In words, if I have a min s,t-cut $\delta(S)$ and two vertices v,w on the same side of the shore (S), then I can compute the minimum cut between v,w with the shore containing them.

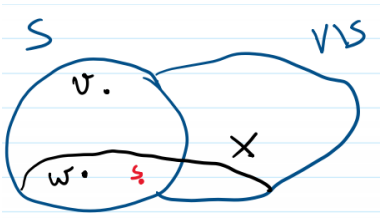


Figure 9.16: Proof setting.

Proof. Let $\delta(X)$ be a minimum v,w-cut. Note that $S \cap X \neq \emptyset$ and $S \cap \bar{X} \neq \emptyset$. WLOG, relabel (switch S with $\bar{S}$ and/or X with $\overline{X}$) if necessary, we may assume that $s \in S \cap X$.

First, suppose that $r \in X$. Since S is a r,s-cut, we have $r \in X \cap \bar{S}$.

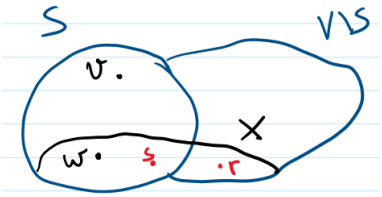


Figure 9.17: Case 1: $r \in X$.

Since $u(\delta(A))$ is submodular, we have

$$u(\delta(S)) + u(\delta(\bar{X})) \geq u(\delta(S \cap \bar{X})) + u(\delta(S \cup \bar{X})).$$

Note that $S \cup \bar{X}$ is a shore of an r,s-cut. Since $\delta(S)$ is a min r,s-cut, we get

$$\begin{aligned} u(\delta(S)) + u(\delta(\bar{X})) &\geq u(\delta(S \cap \bar{X})) + u(\delta(S \cup \bar{X})). \\ u(\delta(S)) + u(\delta(\bar{X})) &\geq u(\delta(S \cap \bar{X})) + u(\delta(S)) \\ u(\delta(\bar{X})) &\geq u(\delta(S \cap \bar{X})) \end{aligned}$$

Since $u(\delta(\bar{X})) = u(\delta(X))$ is a min v,w-cut, $u(\delta(S \cap \bar{X}))$ must also be a min v,w-cut. In particular, it is a subset of S. This case is done.

The second case is analogous. □

Lemma 9.13. *Let $G = (V, E)$, $u \in \mathbb{R}_+^E$, $s, t \in V$, $B \subseteq V$, and $s, t \notin B$. If there exists a min s,t-cut $\delta(X)$ with $X \cap B \neq \emptyset$, then $\lambda(G, s, t) = \lambda(G/B, s, t)$.*

Note this lemma justifies the contraction in the Gomory-Hu tree algorithm: we can safely contract other components without worrying about the minimum cuts being destroyed.

Proof. Since $X \cap B = \emptyset$, $X \subseteq V(G \setminus B)$, contracting B does not impact X, so

$$\lambda(G, s, t) = u(\delta_G(X)) = u(\delta_{G/B}(X)) \geq \lambda(G/B, s, t).$$

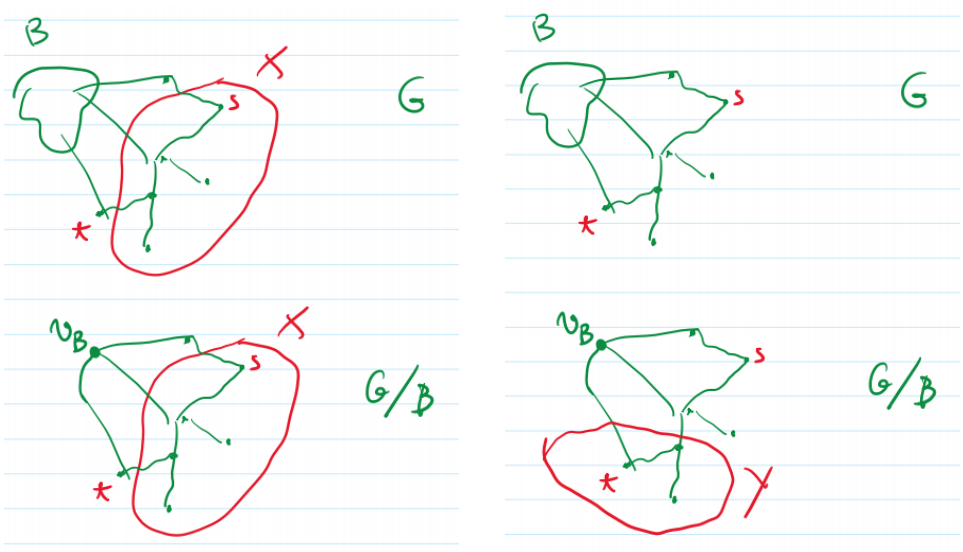


Figure 9.18: Left: X is min s,t-cut. Right: Y is a min s,t-cut in G/B.

Let $Y \subseteq V(G/B)$ be a minimum s,t-cut in G/B with $v_B \notin Y$ (relabel Y and $\bar{Y}$ if necessary). Then Y is also an s,t-cut in G. Thus, $\lambda(G/B, s, t) = u(\delta_{G/B}(X)) = u(\delta_G(X)) \geq \lambda(G, s, t)$. □

Definition 9.14. Suppose T is a GH-tree at any point during the algorithm. Let $f_e, \forall e \in E(T)$ be its labels. Let RS be an edge in T. We say RS has a **representative** if there exists $r \in R$ and $s \in S$ such that

- $\lambda(G, r, s) = f_{RS}$, i.e., the label f_{RS} on edge RS represents the min r, s-cut in the original graph G, and
- the connected component of $T \setminus \{RS\}$ induces the cut of weight $\lambda(G, r, s)$, i.e., the two connected components in $T \setminus \{RS\}$ correspond to the two shores of a min r, s-cut.

Lemma 9.15. *Every edge in $E(T)$ has a representative at all times.*

Proof. We argue by induction. This is clearly true when initially there are no edges in the tree. It is also true when we build the first edge. Now let $x, y \in R$ and X, Y define a cut in G_R with $Y := V(G_R) \setminus X, x \in X, y \in Y, u(\delta_{G_R}(X)) = \lambda(G_R, x, y)$. Note this describes how we want to split R, i.e., how we go from the current GH-tree to the next GH-tree.

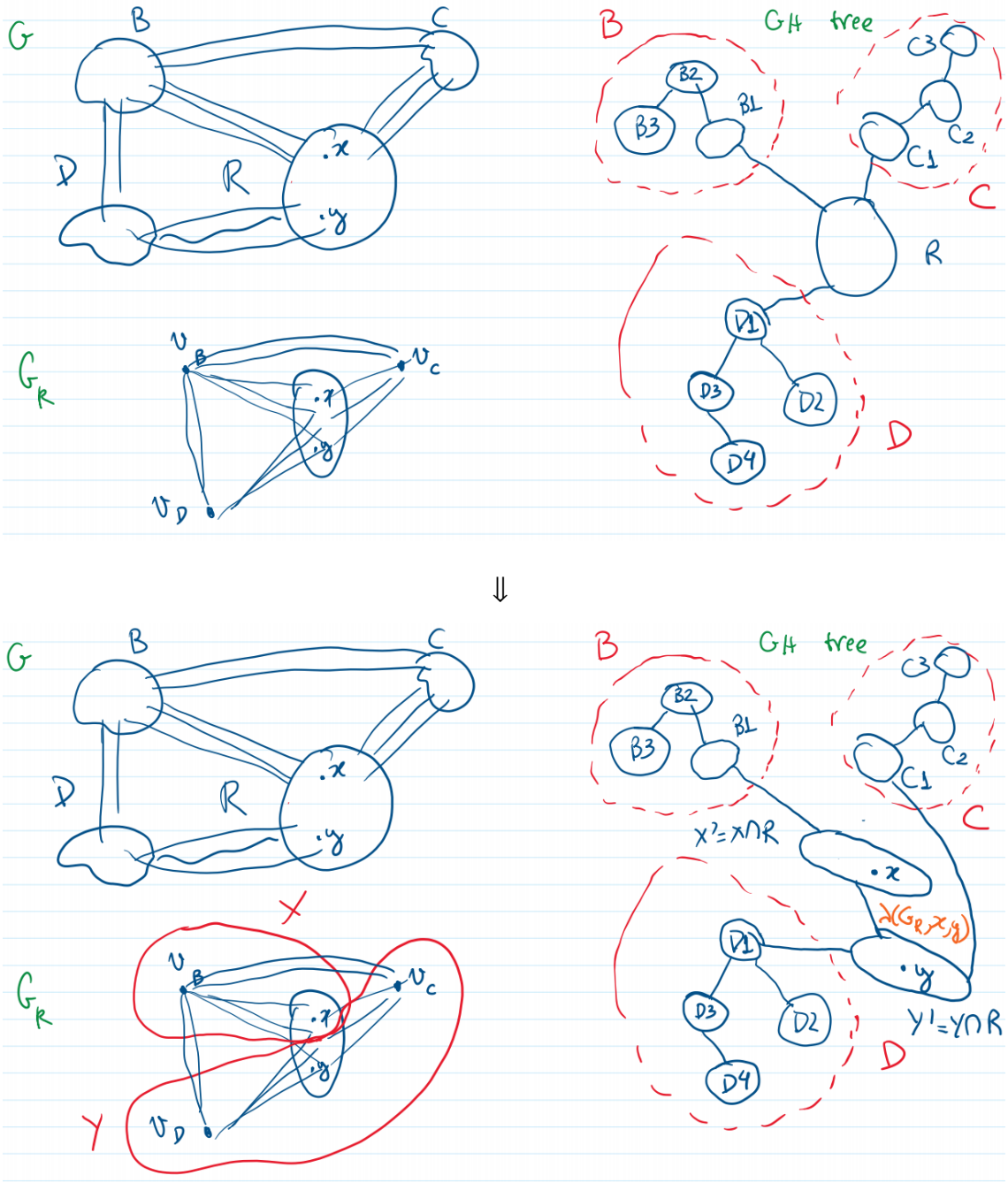


Figure 9.19: G, G_R, and GH-tree before and after the current iteration of the algorithm.

It suffices to show that $\lambda(G_R, x, y) = \lambda(G, x, y)$. as this immediately shows the first part of the definition of being represented; moreover, the definition of how the algorithm redistributes the edges in $\delta_T(R)$ yields the second part of the definition.

Restate the results we've proved before:

- L1 Let $\delta(S)$ be a min r, s-cut and let $v, w \in S$. Then there exists a min v, w-cut $\delta(T)$ such that $T \subseteq S$.
- L2 Let $G = (V, E)$, $u \in \mathbb{R}_+^E$, $s, t \in V$, $B \subseteq V$, and $s, t \notin B$. If there exists a min s, t-cut $\delta(X)$ with $X \cap B \neq \emptyset$, then $\lambda(G, s, t) = \lambda(G/B, s, t)$.

For clarity, let us denote each edge AB as (A, B). Since (B_1, R) had a representative, i.e., there exists $b_1 \in B_1$ and $b_2 \in R$ such that $u(\delta_G(B)) = \lambda(G, b_1, b_2)$. Apply Lemma 1 with $S = \bar{B}$, so there exists a min x, y-cut $\delta_G(U)$ with $U \subseteq \bar{B}$. Apply Lemma 2, so $\lambda(G, x, y) = \lambda(G/B, x, y)$. Thus, it's safe to contract B.

Continue in a similar fashion, since (D_1, R) has a representative, its label was $\lambda(G, d_1, d_2)$ and $u(\delta_G(D)) = \lambda(G, d_1, d_2)$ with some $d_1 \in D_1$ and $d_2 \in R$. Since $D \cap B = \emptyset$, apply Lemma 2 so that $\lambda(G, d_1, d_2) = \lambda(G/B, d_1, d_2)$. So $\delta_{G/B}(D)$ is still a min d_1, d_2-cut in G/B. Apply Lemma 1 to get a min x, y-cut in G/B $\delta_{G/B}(W)$ with $W \subseteq \bar{D}$. Apply Lemma 2, so $\lambda(G, x, y) = \lambda(G/B, x, y) = \lambda((G/B)/D, x, y)$.

In fact, you can continue doing the same argument and eventually arrive at $\lambda(G, x, y) = \lambda(G_R, x, y)$. In other words, the new edge (X', Y') has a representative. Also, every other edge whose vertex set did not change still has a representative. It remains to show the new edges (except (X', Y')) created by the algorithm, e.g., (B_1, X'), (D_1, Y'), and (C_1, Y').

Consider some redistributed edge in $\delta_T(R)$ with label $\lambda(G, b, g)$ such that $b \notin R$ and $g \in R$. WLOG, suppose the edge was distributed with new endpoint X. If $g \in X$, then the edge still has a representative. Now suppose $g \in Y$. We claim in this other case that $\lambda(G, b, g) = \lambda(G, b, x)$, so the redistributed/new edge (B_1, X') still has a representative.

Suppose we have the following lemma (see assignment):

- L3 Let $G = (V, E), u \in \mathbb{R}_+^E$, $p, q, r \in V$. Then $\lambda(G, p, q) \geq \min\{\lambda(G, q, r), \lambda(G, p, r)\}$.

Note this is true iff the smallest two of $\lambda(G, p, q)$, $\lambda(G, q, r)$, and $\lambda(G, p, r)$ are equal. This observation (iff) gives a shorter proof of the lemma:

- $\implies$ RHS is always equal to the smallest, so the result follows.
- $\impliedby$ Let $\lambda(G, p, q)$ be the smallest. Then one of the others must be equal to $\lambda(G, p, q)$.

We want to show that $\lambda(G, b, g) = \lambda(G, b, x)$. Since $b \in B$ and $x \notin B$,

$$u(\delta_G(B)) = \lambda(G, b, g) \geq \lambda(G, b, x).$$

Next, recall we've proven $\lambda(G, x, y) = u(\delta_G(S))$. By Lemma 1, there exists a min b, x-cut $\delta_G(W)$ with $W \subseteq S$. Now let $G' = G/Y'$. Apply Lemma 2, we get $\lambda(G, b, x) = \lambda(G', b, x)$. In words, if we contract Y' from G, this shouldn't impact the minimum (B_1, X') cut. But then this implies that any $v_{Y'}, b$-cut in G' is a b, g-cut in G, so we get

$$\lambda(G', v_{Y'}, b) \geq \lambda(G, b, g).$$

Also, any $x, v_{Y'}$-cut in G' is an x, y-cut in G, so

$$\lambda(G', v_{Y'}, x) \geq \lambda(G, x, y).$$

Moreover, the min x, y-cut in G is a b, g-cut in G, so

$$\lambda(G, x, y) \geq \lambda(G, b, g).$$

Combining these three inequalities and use Lemma 3, we get

$$\begin{aligned}
\lambda(G, b, x) = \lambda(G', b, x) &\geq \min\{\lambda(G', v_{Y'}, b), \lambda(G', v_{Y'}, x)\} && \text{Lemma 3} \\
&\geq \min\{\lambda(G, b, g), \lambda(G, x, y)\} && \text{Inequality 1, 2} \\
&\geq \lambda(G, b, g) && \text{Inequality 3} \\
&\geq \lambda(G, b, x) && \lambda(G, b, g) \geq \lambda(G, b, x).
\end{aligned}$$

Therefore, everything above is equal. Our proof is done. □

We are finally ready to show that the final GH-tree stores all information we need for global min-cuts. In particular, the min-cut value is equal to the smallest label on the unique path in the GH-tree.

Theorem 9.16. *Let T be the final GH-tree. Then for all $r, s \in V$, $\lambda(G, r, s)$ is equal to the smallest label of an edge in $T_{r,s}$. Also, if e^* is such an edge, then $\lambda(G, r, s) = u(\delta(H))$, where H is one of the connected components of $T \setminus e^*$.*

Proof. Let $T_{r,s} = v_0, e_1, v_1, e_2, \ldots, e_k, v_k$ and write f_e to be the labels of edges in T. By the previous lemma, $\lambda(G, v_{i-1}, v_i) = f_{e_i}$ for all $i \in [k]$. We claim that

$$\lambda(G, r, s) \geq \min_{i \in [k]} \lambda(G, v_{i-1}, v_i),$$

which shows both statements: the minimum v_{i-1}, v_i-cut induced by components of $T - e_i$ are also r, s-cuts; each edge e_i has a representative in $\{v_{i-1}\}, \{v_i\}$. We do induction on k. The base case $k = 1$ is trivial. Fix $k \geq 2$. By induction, $\lambda(G, r, v_{k-1}) \geq \min_{i \in [k-1]} \lambda(G, v_{i-1}, v_i)$. Finally, by Lemma 3, $\lambda(G, r, s) \geq \min\{\lambda(G, r, v_{k-1}), \lambda(G, v_{k-1}, s)\} \geq \min_{i \in [k]} \lambda(G, v_{i-1}, v_i)$ as desired. □

This wraps up the chapter on flows and cuts.

Part V

Miscellaneous

10.1 Randomized Algorithms.

Instead of having deterministic algorithms, we can use *randomization* in a clever way to achieve one or more of:

- better complexity,
- better performance guarantees,
- simpler algorithm,
- better practical performance.

Karger's Algorithm for Min-Cut

Problem. *Given $G = (V, E)$, $u \in \mathbb{R}_+^E$, find $\emptyset \neq S \subsetneq V$ minimizing $u(\delta(S))$.*

Assume G is connected.

Algorithm. *While $|V(G)| > 2$:*

- *Choose e with probability $u_e/u(E(G))$.*
- *Update $G \leftarrow G/e$.*

Return the cut separating the 2 vertices.

Observe edges with higher weight have a higher probability being chosen/contracted.

Theorem 10.1. *Karger's algorithm returns a min cut with probability $\geq \frac{2}{n(n-1)}$.*

Proof. Let $A \subseteq E$ be a min cut. The algorithm returns A if none of its edges are contracted. Let G^i be the graph after i edges have been contracted. Note $|V(G^i)| = n - i$. Since A is a min cut and each $\delta_{G^i}(v)$ represents a cut in G, we have

$$\forall v \in V(G^i) : u(A) \leq u(\delta_{G^i}(v)) \implies u(A) \leq \sum_{v \in V(G^i)} \frac{u(\delta_{G^i}(v))}{n-i} = \frac{2u(E(G^i))}{n-i}. \quad (\star)$$

The probability of picking an edge in $E(G^i) \setminus A$ is $1 - \frac{u(A)}{u(E(G^i))}$. Combined with $\star$,

$$1 - u(A)/u(E(G^i)) \geq 1 - \frac{2}{n-i} = \frac{n-i-2}{n-i}.$$

Complete this. □

For more information, see CO-351 notes.

Boosting Probability

The algorithm fails with probability $\leq 1 - \frac{2}{n(n-1)}$, which is about 99% when $n = 100$. This is really bad. To fix this, we can run the algorithm q times (independently) and pick the best result returned. The algorithm fails iff it fails ***all*** q times, which has a probability

$$\leq \left(1 - \frac{2}{n(n-1)}\right)^q \leq \left(1 - \frac{2}{n^2}\right)^g \leq e^{-2q/n^2}$$

where the last inequality follows from $1 + x \leq e^x$. Choose $q = Kn^2$, the probability of failure is $\leq e^{-2K}$. For example, for $K = 5$, the probability of success is > 0.9999.

As a remark, if the probability of chosen each edge is uniform, the it requires an exponential number of boosts.

Take CS-761 if you are interested in randomized algorithms.

10.2 Approximation Algorithms.

We want to find polytime algorithms with quality guarantee for NP-hard problems? In particular, for a maximization problem with an optimal solution exists

$$z^* = \max f(x) \quad s.t. \quad x \in S,$$

an $\alpha(n)$ approximation algorithm is an algorithm that, on inputs of size n, returns in time poly(n) a solution $\bar{x}$ of value $\geq \alpha(n) \cdot z^*$. As a remark,

- For maximization problems, $\alpha(n) \leq 1$ and we need $z^* \geq 0$.
- For minimization problems, $\alpha(n) \geq 1$ and we need $z^* \geq 0$.

Note the greedy algorithm for matroid is an approximation algorithm!

K-Cuts

Problem. *Given $G = (V, E)$, $u \in \mathbb{R}_+^E$, find $A \subseteq E$ minimizing $u(A)$ such that $(V, E \setminus A)$ has $\geq k$ components.*

Algorithm.

- *Compute a Gomory-Hu tree T with edge labels f_e.*
- *Order edges of T such that $f_{e_1} \leq f_{e_2} \leq \cdots \leq f_{e_{n-1}}$.*
- *Remove $e_1, \ldots, e_{k-1}$ gives k connected component of T; name them $V_1, \ldots, V_k$.*

Let $A^* \subseteq E$ be an optimal solution with connected components $V_1^*, \ldots, V_k^*$. Note that

$$\sum_{i=1}^{K} u(\delta(V_i^*)) = 2u(A^*))$$

as we counted every edge in A^* exactly twice.

Theorem 10.2. *the solution we return*

$$u\Big(\overbrace{\{vw \in E \mid v \in V_i, w \in V_j, i \neq j\}}\Big) \leq \left(2 - \frac{2}{k}\right) \cdot u(A^*).$$

Proof. Contract V_i^* into v_i^* in T. Now drop some edges to get a GH tree T'. Consider the following example. Suppose we have $V_1^* = \{1, 6\}, V_2^* = \{3, 4\}, V_3^* = \{2, 5\}$. Start with the Gomory-Hu tree on the right, contract them to obtain the multi-graph in the middle, then drop some edges to get a tree T' in the left.

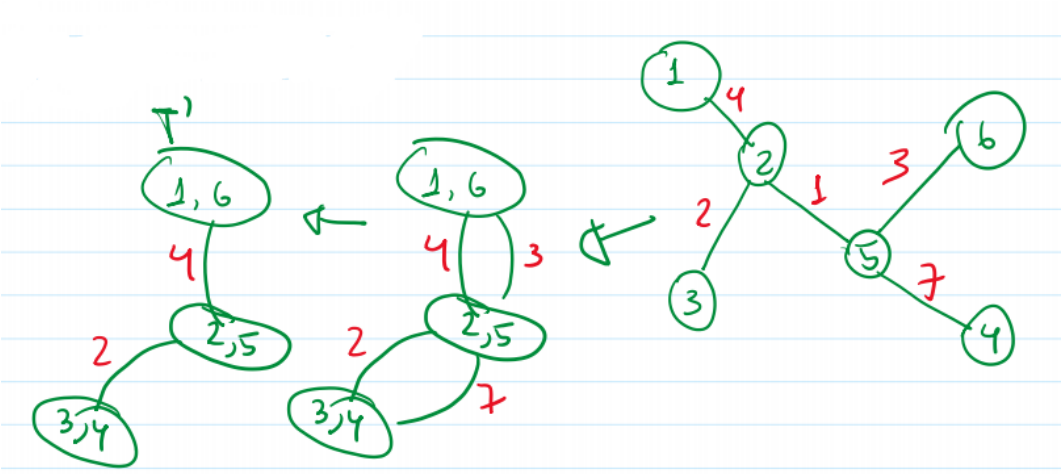


Figure 10.1: Example.

Note T' has K nodes and $K-1$ edges. Each edge $v_i^* v_j^*$ in T' corresponds to an edge ab in T with $a \in V_i^*$ and $b \in V_j^*$. Thus,

$$f_{v_i^* v_j^*} = f_{ab} \leq u(\delta_G(V_j^*))$$

since $\delta_G(V_j^*)$ is an a, b-cut.

Consider T' as a tree rooted at some v_r^* maximizing

$$u(\delta_G(V_r^*)).$$

Apply the inequality above to the child endpoint of every edge. Every node except the root is counted once. Thus,

$$\begin{aligned}
\sum_{e \in E(T')} f_e &\leq \sum_{j=1}^{k} u(\delta_G(V_j^*)) - u(\delta_G(V_r^*)) \\
&\leq \sum_{j=1}^{k} u(\delta_G(V_j^*)) - \sum_{j=1}^{k} \frac{u(\delta_G(V_j^*))}{k} \\
&= \left(1 - \frac{1}{k}\right) 2u(A^*)
\end{aligned}$$

But we picked the cheapest $k-1$ edges in T which is a lower bound for

$$\sum_{e \in E(T')} f_e.$$

This concludes the proof. □

Set Cover

Problem. *Given elements $G = \{1, \dots, m\}$ and a set $\{S_1, \dots, S_n\}$ of subsets of G, each with a cost $c_j \geq 0$, find $\Delta \subseteq \{1, \dots, n\}$ minimizing $c(L)$ such that*

$$\sum_{j \in \Delta} S_j = G.$$

LP formulation:

$$\begin{aligned} (P) := \min \quad & \textstyle\sum_{j=1}^{n} c_j x_j \\ s.t. \quad & \textstyle\sum_{j : i \in S_j} x_j \geq 1 \quad \forall i = 1, \dots, m \\ & x \geq 0 \end{aligned}$$

We may assume that any optimal solution x^* to (P) satisfies $0 \leq x_j^* \leq 1$ for all $j = 1, \dots, n$. Let us view these x_j's as probabilities.

Algorithm.

1. *For each $j \in [n]$, select S_j with probability x_j^* independently.*
2. *Return Δ as the selected S_j's.*

Claim. *The probability i is covered by Δ is at least $1 - 1/e$.*

Proof. By independence, the probability that i is uncovered is at most

$$\prod_{j : i \in S_j} (1 - x_j^*) \leq \prod_{j : i \in S_j} e^{-x_j^*} = \exp\left(- \sum_{j : i \in S_j} x_j^* \right) \leq e^{-1}.$$

as $\sum_{j : i \in S_j} x_j^* \geq 1$. □

Run this algorithm $2 \ln n$ times and output the UNION of all sets that were picked at any given iteration. The probability that any i is not covered is at most

$$e^{-2 \ln n} = \frac{1}{n^2}.$$

By a union bound, the probability that there is an uncovered i is at most $1/n$.

The expected cost of a single run of this algorithm is given by

$$\mathbb{E}[c(\Delta)] = \sum_{j=1}^{n} c_j \Pr(S_j \in \Delta) = \sum_{j=1}^{n} c_j x_j^* = \mathrm{OPT}_P \leq \mathrm{OPT}_{\text{Set Cover}}$$

Hence, the final solution has expected cost $\leq 2 \ln n \cdot \mathrm{OPT}$.

Primal-Dual Approach for Set Cover

Consider the following primal-dual pair:

$$\begin{aligned}(P) := \min \quad & \sum_{j=1}^{n} c_j x_j \\ s.t. \quad & \sum_{j:i\in S_j} x_j \geq 1 \quad \forall i = 1,\dots,m \\ & x \geq 0\end{aligned} \qquad \begin{aligned}(D) := \max \quad & \sum_{i=1}^{m} y_i \\ s.t. \quad & \sum_{i\in S_j} y_i \leq c_j \quad \forall j = 1,\dots,n \\ & y \geq 0\end{aligned}$$

The idea is to find x^*, y^* feasible to (P) and (D) satisfying the CS conditions:

- $\forall j : x_j^* > 0, \sum_{i\in S_j} y_i^* \geq c_j$,
- $\forall i : y_i^* > 0, \sum_{j:i\in S_j} x_j^* \leq 1$.

Note how we are writing the CS conditions. For example, when $x_j^* > 0$, we want the corresponding dual constraint to be tight, i.e., $\sum_{i\in S_j} y_i^* = c_j$. Instead of writing this, we say $\sum_{i\in S_j} y_i^* \geq c_j$, which combined with the constraint ($\leq$) implies the tightness.

We need to relax the CS condition a bit as we are likely not able to find an integer solution. Replace the "1" in the RHS of the primal constraint with f for some $f > 1$. This measures "how far" our solution is from 1. Note with this, we get

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{j=1}^{n} x_j^* \left(\sum_{i\in S_j} y_i^* \right) = \sum_{i=1}^{m} y_i^* \left(\sum_{j:i\in S_j} x_j^* \right) \leq f \sum_{i=1}^{m} y_i^* \cdot 1 = f \cdot \text{OPT}.$$

Consider the following algorithm.

1. Start with $x^* \leftarrow 0, y^* \leftarrow 0$.
2. While $\exists i : \sum_{j:i\in S_j} x_i^* < 1$:
   (a) Pick one such i and raise y_i^* until $\sum_{i\in S_j} y_i^* = c_j$ for some j.
   (b) Let $x_j^* = 1$ for all j where $\sum_{i\in S_j} y_i^* = c_j$.
3. Output x^*.

Note we started with a feasible dual solution. We will maintain a feasible dual solution throughout the algorithm. Some of the primal variables/constraints might be infeasible. We will fix them.

The algorithm always progresses in each iteration since the only way an item is not covered is if no previously chosen set covers it. Hence, there is some unchosen set which covers it and the corresponding set variable can be raised. It is also easy to see its polynomial time complexity.

Claim. *The proposed algorithm is an f-approximation algorithm where f is the maximum number of sets in an element appears.*

Proof. Omitted. □

10.3 Integer Programming.

Minimum Bounded Degree Spanning Tree

Consider the problem of finding a minimum cost spanning tree of G where each vertex in the tree has degree at most k.

Problem (MBDST). *Given $G = (V, E)$ connected, $c \in \mathbb{R}^E$, $k \in \mathbb{Z}_+$, $k \geq 2$, find a spanning tree T of minimum cost such that*

$$\forall v \in V : |\delta_T(v)| \leq k.$$

First, note the problem is NP-hard as it is equivalent to HamPath for $k = 2$. Let $G' = (V, E')$ and let G be the corresponding complete graph. Define $c_e = -1$ if $e \in E'$ and 0 otherwise. Then G' has a Hamiltonian path iff the MBDST of G has cost $-n - 1$.

Let $OPT(k)$ be the value of MBDST with parameter $k \geq 2$. We want to find a spanning tree T with (a slightly relaxed constraint)

$$\forall v \in V : |\delta_T(v)| \leq k + 2$$

with $c(T) \leq OPT(k)$. Assume the problem is feasible.

Remark. *Note this is slightly different from the classic approximation algorithm setting. In a typical approximation algorithm, we work with feasible solutions whose objective values are slightly worse than the optimal. Here, we work with "almost-feasible" solutions whose objective values are no worse than the optimal. In other words, we relax the feasibility condition instead of the optimality condition here.*

Consider the following LP formulation, which is obtained by adding the degree constraint to the MST LP:

$$\begin{aligned} \min \quad & \textstyle\sum_{e \in E} c_e x_e \\ s.t. \quad & x(E(S)) \leq |S| - 1 \quad \forall S \subseteq V \\ & x(E) = n - 1 \\ & x(\delta(v)) \leq k \qquad \forall v \in V \\ & x \geq 0 \end{aligned}$$

An integral optimal solution to this LP is an optimal solution to MBDST. Unfortunately, this LP is exponential in size of input (graph), so we can't solve it directly.

Integer Programming

Let x^* be an optimal solution to the LP above. Define $E^* := \{e \in E : x^*_e > 0\}$, the set of non-zero variables. Intuitively, the positive yet non-integral x variables are saying "I'm not sure whether we want this edge or not" and the zero x variables are saying "we probably don't want this edge". We will show this is actually the correct intuition.

Next, find the desired spanning tree in $G^* = (V, E^*)$.

The road map is as follows.

1. We may assume x^* is an extreme point of the LP.
2. Under this assumption, $|E^*(U)| \leq 2|U| - 1$ for all $U \subseteq V$.
3. By A5Q1, there exists an orientation of the edges E^* such that the in-degree of every vertex is at most 2.
4. Use matroid intersection to find the desired tree T.
5. Argue that $c(T) \leq OPT(k)$.

Note the first two statements use arguments of polyhedral theory, which is covered in CO-452/652, the Integer Programming course. We will focus on the last two steps.

Let A^* be an orientation of E^* such that the digraph $D^* = (V, A^*)$ satisfies

$$\forall v \in V : |\delta^-_{D^*}(v) \leq 2.$$

For any $B \subseteq A^*$, let $E(B)$ be the corresopnding set of (undirected) edges in E^*. Define $\mathcal{M}_1 = (E^*, \mathcal{I}^*)$ where

$$\mathcal{I}^* := \{F \subseteq E^* \mid \forall v \in V : |F \cap E(\delta^+_{D^*}(v))| \leq k\}.$$

Note this is a partition matroid.

Lemma 10.3. *If $F \subseteq \mathcal{I}^*$, then $|F \cap \delta_G(v)| \leq k + 2$.*

Proof. The degree of each vertex is the sum of in and out degrees, which is at most $k + 2$.

Let $\mathcal{M}_2$ be the graphic matroid of F^*. Consider the following algorithm:

1. Define $\bar{c} := -c + M > 0$. Note this implies that T is a min-cost spanning tree wrt c iff T is a max-cost spanning tree wrt $\bar{c}$.
2. Compute the max-weight independent set T in $\mathcal{M}_1$ and $\mathcal{M}_2$.
3. Return T.

Lemma 10.4. *T is a spanning tree.*

Proof. Skipped.

By results in matroid intersection (plus some other polyhedral theory), x^T corresponding to T (not the transpose!) is an optimal solution to the following LP:

$$\begin{aligned} \min \quad & \textstyle\sum_{e \in E} c_e x_e \\ s.t. \quad & x(E(S)) \leq |S| - 1 \quad \forall S \subseteq V \\ & x(E) = n - 1 \\ & x(E(\delta^+_{D^*}(v))) \leq k \quad \forall v \in V \\ & x \geq 0 \end{aligned}$$

But x^* is a feasible solution for this LP. Therefore, we get

$$c(T) \leq c^T x^* \leq OPT(k)$$

as desired.

More on Integer Programming

Consider the integer program

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

Some of our approaches based on solving it by solving its LP relaxation is often not possible. In the Integer Programming course, we study the following questions:

- When can we solve the IP by solving its LP relaxation? (Polyhedral Theory)
- What to do if this is not possible and I still want to solve IP?
  - Approximation algorithms: relax optimality constraint.
  - Integer programming: still want optimal solutions, relax polytime constraint.

Take CO-452/652 if you are interested. This is the end of this course.