CO 353: Computational Discrete Optimization

Teach by: Kanstantsin Pashkovich

Jiucheng Zang

January 6th, 2026

Notes compiled for CO 353.

Contents

CHAPTER 1

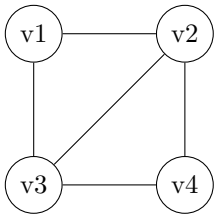
Graph Theory & Optimization

Definition 1.1. An (undirected) graph G is a pair (V, E), V is a non-empty set, E is a set of unordered pairs of elements from V.

- V is called the set of vertices/nodes.
- E is the set of edges.
- For an edge $e = \{u, v\} \in E$, $u, v \in V$.
  - e is said to be incident to u (or v).
  - $u\&v$ are said to be adjacent (to each other).
  - u (or v) is said to be an end of e.
- For $e = uv \in E$, $e' = u'v' \in E$, e and e' are called parallel (edges) if $\{u, v\} = \{u', v'\}$.

  (Strictly speaking, E would be a multiset and/or we would label the two edges differently.)

Example



$$V = \{v1, v2, v3, v4\}, \quad E = \{\{v1, v2\}, \{v2, v4\}, \{v4, v3\}, \{v3, v1\}, \{v2, v3\}\}$$

Definition 1.2. Given a graph $G = (V, E)$, $u, v \in V$, $u \neq v$, a uv-path is a sequence of vertices / nodes $w_1, w_2, \ldots, w_k$, where $w_1 = u, w_k = v$, and $\{w_i, w_{i+1}\} \in E$ for all $1 \leq i < k$.

(Strictly speaking, this is a uv-walk, in graph theory parlance.)

Example

In the graph on the top, v_1, v_3, v_2 is a v_1v_2-path.

Definition 1.3. Given graph $G = (V, E)$, a cycle is a sequence of (with k at least 3) nodes $w_1, w_2, \ldots, w_{k+1} = w_1$, where $\{w_i, w_{i+1}\} \in E$ for all $1 \leq i \leq k$, and the nodes $w_1, w_2, \ldots, w_k$ are all distinct. (in a simple graph)

Example

v_2, v_3, v_4, v_2 is a cycle in the above graph.

Definition 1.4. We say a graph $G = (V, E)$ is connected if for every $u, v \in V, u \neq v$, there is a uv-path in G.

Definition 1.5. A graph with edge lengths has a real number $l_e \geq 0$ associated with each edge $e \in E$.

Then, the length of a uv-path $P : u = w_1, w_2, \ldots, w_k = v$ is

$$l(P) = \sum_{i=1}^{k-1} l(w_i, w_{i+1}).$$

Note: $l(w_i, w_{i+1})$ in short is $l_{w_i, w_{i+1}}$

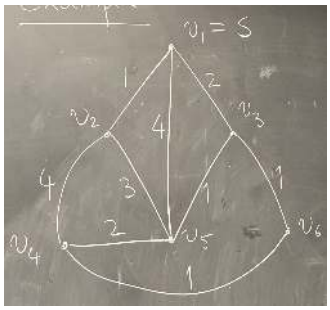
Definition 1.6. The (shortest path) distance between u and v is denoted by

$$d(u, v) := \min l(P) \quad \text{P is a } uv\text{-path}$$

Definition 1.7. Suppose there is a distinguished start node s in G. Then $d(s, u)$ is abbreviated as $d(u)$.

Example

Consider the graph below with edge lengths as shown.



s, v_5, v_4 in an $s - v_4$ path with length $l(p) = 4 + 2 = 6$.

However, $d(v_4) = 4$, given by $s = v_1, v_3, v_4$.

1.1 Shortest Path Problem

Definition 1.8. A directed graph $G = (V, E)$ consists of a set of nodes V, and a set E of ordered pair uv, where $u, v \in V$.

(In graph theory terminology, directed edges are called arcs.)

Directed Paths (walks) and cycles are defined analogously to the undirected case.

Example

A directed $u - v$ path is a sequence $u = w_1, w_2, \ldots, w_k = v$ where $u \neq v$, $w_i \in V$, and $w_i w_{i+1} \in E$ is a directed edge.

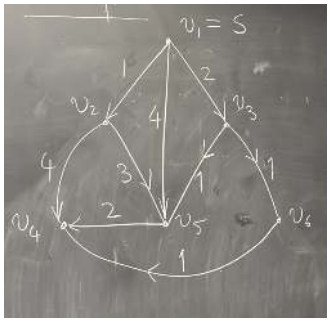
Edges may have length of (directed) paths are defined analogously, as is the distance $d(u)$ of a node u from a start vertex s.

Definition 1.9. Given a directed graph $G = (V, E)$ with edge lengths $l_e, e \in E$, $l_e \geq 0$, and a distinguished start node s, find the (shortest paths and) distance from s to all other vertices in the graph.

For simplicity, we assume there is a path from s to u for all $u \in V, u \neq s$.

Example

Consider the directed graph below with edge lengths as shown.



$$d(s) = 0, d(v_2) = 1, d(v_3) = 2, d(v_4) = 3, d(v_5) = 4, d(v_6) = 3$$

1.1.1 Dijkstra's Algorithm

1. Suppose uv is an edge, P_1 is a shortest path from s to v. Then $l(P_1) \leq l(P_2) + l_{uv}$

2. Suppose P is the shortest path from s to v, and uv is the last edge on P. Then P_1 is a shortest path from s to u.

$$d(v) = \underbrace{l(P_1)}_{=d(u)} + l_{uv}$$

Moreover, for every $v \neq s$, there is such a shortest path P form s (by assumption), and therefore a vertex u s.t. $d(v) = d(u) + l_{uv}$.

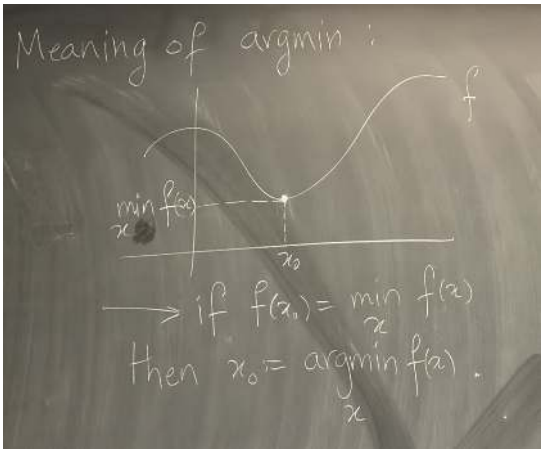
Idea: Maintain the set A of "explored" vertices, i.e. vertices with known the shortest path distances from s, and labels.

$$v \in V \setminus A (i.e., v \notin A, \text{ where } d'(v) \text{ are upper estimates on the distance of } v \text{ from } s)$$

Algorithm 1: Dijkstra's Algorithm

Input: Directed graph $G = (V, E)$, start vertex $s \in V$, edge lengths $l_e \geq 0$ for all $e \in E$
Output: Distances $d(v)$ from s to every $v \in V$
$A \leftarrow \{s\}$; $d(s) \leftarrow 0$; $d(v) \leftarrow \infty$ for all $v \in V \setminus \{s\}$;
while $A \neq V$ **do**
  $d'(v) \leftarrow \min\{d'(v), \min_{u \in A, uv \in E} d'(u) + l_{uv}\}$ for all $v \in V \setminus A$;
  $w \leftarrow \arg\min_{v \in V \setminus A} d'(v)$; `// vertex attaining the minimum`
  $A \leftarrow A \cup \{w\}$; $d(w) \leftarrow d'(w)$;

Definition 1.10.



(Note: Can pick arbitrary such x_0 if not unique.)

Example

For the example above, we can run the Dijkstra's Algorithm

(a) $A \leftarrow \{s\}$, $d(s) \leftarrow 0$, $d(v) \leftarrow \infty$ for all $v \in V \setminus \{s\}$

(b) Iteration 1:

- $d'(v_2) = \min\{\infty, \min_{u \in A, uv \in E} d(s) + l_{sv}\} = \min\{\infty, 0+1\} = 1$
- $d'(v_3) = \min\{\infty, \min_{u \in A, uv \in E} d(s) + l_{sv}\} = \min\{\infty, 0+2\} = 2$
- $d'(v_4) \leftarrow \infty, d'(v_5) \leftarrow 4, d'(v_6) \leftarrow \infty$

$w \leftarrow v_2$, $d(v_2) = d'(v_2) = 1$, $A \leftarrow A \cup \{v_2\}$

(c) Iteration 2:

- $d'(v_3) = \min\{2, 0+2\} = 2$
- $d'(v_4) = \min\{\infty, d(v_2) + l_{v_2v_4}\} = \min\{\infty, 1+4\} = 5$
- $d'(v_5) = \min\{4, 0+4, 1+3\} = 4$
- $d'(v_6) = \infty$

$w \leftarrow v_3$, $d(v_3) = d'(v_3) = 2$, $A \leftarrow A \cup \{v_3\}$

1.1.2 Correctness of the Dijkstra's Algorithm

We would like to show

a) The algorithm terminates

b) The output $d(v)$ is (the shortest path) distance of v from s, for all $v \in V$

Proof

a) The size of A is 1 at the beginning, and it increases by 1 in every iteration, so after $(n-1)$ iterations where $n = |V|$, $A = V$ and the algorithm terminated.

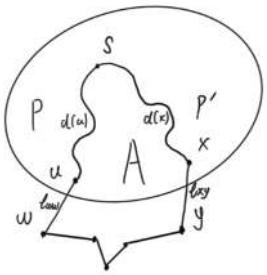
b) We show by infuction that after every iteration, the value $d(v), v \in A$ is correct.

Iteration 0: $d(s) = 0$ is correct. Assume that the statement holds for iteration $i \geq 0$.

Iteration $i+1$: Suppose we add w to A, we will show that $d(w)(= d'(w)$ after i+1 iterations) is the distance of w from s.

Since (by assumption) there is an s-w path, $d'(w) = d(u) + l_{uw}$ for some $u \in A$. Suppose

$d(w) = d'(w) = d(u) + l_{uw}$ is not the shortest path distance. Suppose P' is a shortest $s - w$ path $l(P') < d(w) = d(u) + l_{uw}$.



The path P' starts in A and ends outside A, so there is a first vertex y on F' that is outside A. Suppose x is the vertex on P' just before y. In iteration $i + 1$, we have the estimate $d'(y) \leq d(x) + l_{xy} \leq l(P') < d(w)$.

This means we would have added y to A instead of w a contradiction.

Therefore, $d(w) = d'(w)$ is indeed the shortest path distance from s to w.

By induction, the statement holds after every iteration.

□

1.1.3 Runtime of Dijkstra's Algorithm

Note

Runtime related notation check here.

It is written in terms of input length; which is the number of bits required to describe the input.

Example

for an integer $n > 1$ the number of bits required to write it in bits(binary presentation) is

$$\approx \log_2(n)$$

Input length for a graph:

Suppose $|V| = n$, $|E| = m$.

$$
\begin{aligned}
&\approx \log_2(n) + \log_2(m) \\
&\qquad + \underbrace{m(2 \cdot \log_2(n))}_{\text{edges}} \\
&\qquad + \underbrace{\sum_{e \in E} \log_2(l_e)}_{\text{edge lengths}} \\
&\qquad = O\left(m \cdot \log(n) + \sum_e \log(l_e)\right)
\end{aligned}
$$

Runtime:

$$
\begin{aligned}
&= 3n \text{ (step 1)} \\
&+ n \text{ iterations} \\
&\quad \cdot (O(m) \text{ (update estimates)} + O(n) \text{ (find min)}) \\
&= O(n^2 + nm) \\
&= O(n^2) \text{ (since } m \leq n^2 \text{ for directed graph)}
\end{aligned}
$$

Note

This can be optimized to $O(nm)$

1.2 Runtime Related Notation

Order notation

Suppose $f : \mathbb{R}_+ \to \mathbb{R}_+$ (or $f : \mathbb{Z}_+ \to \mathbb{R}_+$)

Definition 1.11. We say $f : \mathbb{R}_+ \to \mathbb{R}_+$ is of order $g : \mathbb{R}_+ \to \mathbb{R}_+$, written as $f(n) = O(g(n))$ if there exist constants $c > 0, n_0 \geq 0$ such that

$$f(x) \leq cg(x) \quad \forall x \geq n_0$$

Example

- $n = O(3n)$, $n_0 = 1, c = 3$

  $3n = O(n)$, $n_0 = 1, c = 1/3$

- $f(x) = 2x + 100$, $f(n) = O(3n)$

  $n_0 = 100, c = 1$, $2n + 100 \leq 3n$ for all $n \geq 100$

- $f(x) = 2x + 100$, $g(x) = x^2$

  Show: $f(x) = O(g(x))$
- $\sqrt(n) = O(n)$ Exercise
- In general, for $\alpha < \beta$; $\alpha, \beta > 0$,

  $n^\alpha = O(n^\beta)$

  Example: $n^{2/3} = O(n^{3/4})$
- $\log_2(n) = O(n)$, $\log_2(n) = O(\log_3(n))$, $\log_\alpha(n) = \frac{\log_\beta(n)}{\log_\beta(\alpha)}$ for all $\alpha, \beta > 1$
- $2^n = O(3^n)$, but $3^n \neq O(2^n)$ Verify
- $f(x) = O(1)$ iff $f(x)$ is bounded by a constant.

The notation generalizes to function of multiple arguments:

Suppose: $f : \mathbb{R}_+^\times \mathbb{R}_+ \to \mathbb{R}_+$,$g : \mathbb{R}_+^\times \mathbb{R}_+ \to \mathbb{R}_+$

Definition 1.12. We say $f(n, m) = O(g(n, m))$ if there exist constants $c > 0, n_0 \geq 0, m_0 \geq 0$ such that

$$f(n, m) \leq cg(n, m) \quad \forall n \geq n_0, m \geq m_0$$

Note

An algorithm is efficient if its running time is upper bounded by a polynomial function in input size.

Reason	Algo1	Algo2
Running Time	$4 \cdot n$	2^n

Let's consider a computer that does 16 operations per 1 minute

To run Algo1 within 1 min, the input size n should be at most 4 (since $4 \cdot n \leq 16$).

To run Algo2 within 1 min, the input size n should be at most 4 (since $2^n \leq 16$).

Let's consider a stronger computer with 32 operations per 1 minute

Now for Algo1 and Algo2 to run within 1 minutes the input size ≤ 8 and ≤ 5 respectively.

Last time we saw that the running time of Dijkstra's Algorithm is $O(\underbrace{m}_{\#\text{ edges}} \cdot \underbrace{n}_{\#\text{ vertices}})$

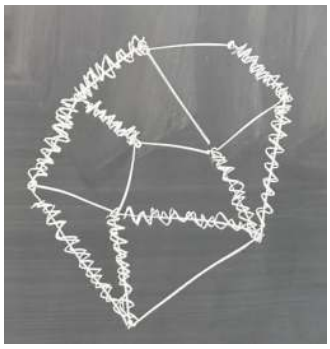
Using Fibonacci Heaps allows for running time $O(m + n \log n)$

1.3 Minimum Spanning Tree

Definition 1.13. A **tree** is a connected acyclic graph.

Definition 1.14. Given a graph $G = (V, E)$, a spanning tree of G such that $F \subseteq E$ and T is a tree.

Example



1.3.1 Minimum Cost Spanning Tree Problem (MST)

Definition 1.15. Given a graph $G = (V, E)$ with costs c_e for each edge $e \in E$, find a spanning tree $T = (V, F)$ such that

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

is minimized.

Proposition

Fundamental Theorem about Tree (FTT)

Let $T = (V, F)$ be a graph, the following statements are equivalent:

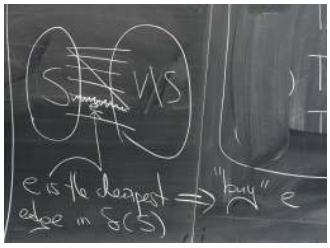
- T is a tree, i.e. T is connected and acyclic.
- T is connected and $|F| = |V| - 1$.
- T is acyclic and $|F| = |V| - 1$.

Definition 1.16. Given a graph $G = (V, E)$, and $S \subset V$, the **cut** $\delta(S)$ is defined as

$$\delta(S) = \{uv \in E : u \in S, v \notin S\}.$$

Theorem 1.1 Cut property

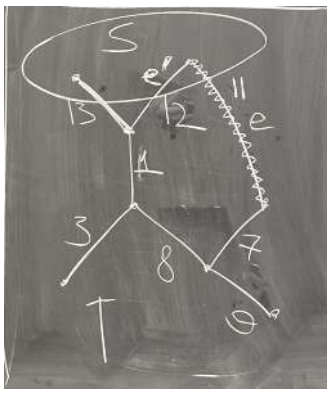
Let's assume that all edges costs c_e are distinct. Let $S \subset V$ be such that $S \neq \emptyset$ and $S \neq V$. Let $e^* = \arg\min_{f \in \delta(S)} c_f$ be the edge with minimum cost across the cut $\delta(S)$. Then every minimum cost spanning tree contains the edge e^*.



Proof

Let's prove by contradiction. Assume that there exists a minimum cost spanning tree $T = (V, F)$ that does not contain the edge e^*.

Consider the graph $(V, F \cup \{e^*\})$ by FTT the graph has a cycle C.



Let us show that $|C \cap \delta(S)| \geq 2$. Note that $|C \cap \delta(S)|$ is even since every time we enter S we must leave S again. Since $e^* \in C \cap \delta(S)$, we have $|C \cap \delta(S)| \geq 2$.

So there is an edge $e' \in C \cap \delta(S)$ such that $e' \neq e^*$.

Consider $T' = (V, F \cup \{e^*\} \setminus \{e'\})$. By FTT, T' is a spanning tree.
(We removed one edge from the cycle and added another edge, the number of edges is still $|V| - 1$, also T' is connected since all the edges in $F \setminus \{e'\}$ are in T'.)

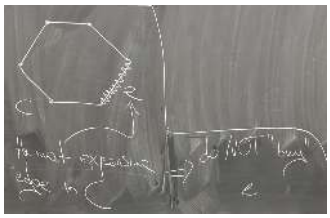
Then $c(T) - c(T') = c_{e'} - c_{e^*} > 0$ since c_{e^*} is the minimum cost edge across the cut $\delta(S)$.

Thus, we got a spanning tree T' with $c(T') < c(T)$ which is a contradiction.

□

Theorem 1.2 Cycle Property

Let us assume that all costs $c_e, e \in E$ are distinct. Let C be a cycle in G and let $e = \arg\max_{f \in C} c_f$. Then NO minimum cost spanning tree contains the edge e.



1.3.2 Prim's Algorithm

Algorithm 2: Prim's Algorithm

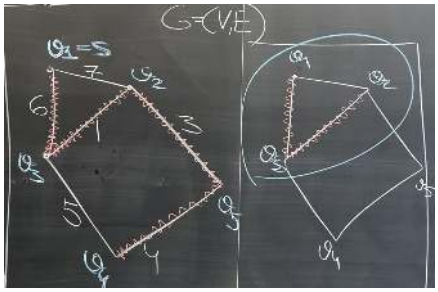
$T \leftarrow \emptyset$;
Select an arbitrary vertex $v_0 \in V$;
$S \leftarrow \{v_0\}$;
while $S \neq V$ **do**
  Select edge $e = (u, v)$ with minimum cost such that $u \in S$ and $v \in V \setminus S$;
  $T \leftarrow T \cup \{e\}$;
  $S \leftarrow S \cup \{v\}$;
return T;

Proof

For simplicity, we assume that all costs are distinct.

Prim's algorithm at termination guarantees that $|T| = |V| - 1$.

At every point during the algorithm. Every vertex in A is connected to s by using only edges in T. So T is connected, and $|T| = |V| - 1$ implies that T is a spanning tree.



By cut property, every edge in T is contained in every minimum cost spanning tree.

□

Remark 1.1

- If all costs are distinct, Minimum Cost Spanning Tree is unique.
- Using Fibonacci heap, the running time of Prim's Algorithm is $O(m + n \log n)$.

1.3.3 Kruskal's Algorithm

Algorithm 3: Kruskal's Algorithm

```
Sort edges such that c(e_1) ≤ c(e_2) ≤ ... ≤ c(e_m);
T ← ∅;
for i = 1 to m do
    if T ∪ {e_i} does not contain a cycle then
        T ← T ∪ {e_i};
return T;
```

Lemma 1.1

Given a connected graph $G = (V, E)$ Kruskal's algorithm correctly compputes a minimum cost spanning tree of G.

Proof

Clearly, at the end of algorithm, T has NO cycles. Let us show that T corresponds to a connected graph.

Recall that u, v are NOT connected in $T = (V, F)$ iff $\exists S \subseteq V, v \in S, u \notin S$ such that $\delta(S) = \emptyset$.

For the contradiction, suppose T is not connected. Thus, there is $S \subseteq V$, $S \neq \emptyset, S \neq V$ but $\delta(S) \cap T = \emptyset$.

Note that G is connected, so $\delta(S) \neq \emptyset$. There exists $e \in \delta(S)$ such that $e \notin T$.

Since e is not in T at the end of the algorithm, $T \cup \{e\}$ contains a cycle C.

Since $|C \cap \delta_G(S)|$ is even and $|C \cap \delta_G(S)| \geq 1$, there is $f \in T$ such that $f \in \delta_T(S)$, contradict to $\delta_T(S) = \emptyset$.

Conclusion: T is a spanning tree.

Let us show that T is a minimum cost spanning tree. (For simplify, we assume all costs are distinct.)

Let us consider a time point when an edge $e = uv$ is about to be added to T.

Thus, there is $S \subseteq V$, $v \in S$, $u \notin S$, $\delta_T(S) = \emptyset$.

By the order in which Kruskal's algorithm inspects edges

$$e = \arg \min_{f \in \delta_G(S)} c(f)$$

By Cut Property, e is contained in every minimum cost spanning tree.

□

Remark 1.2

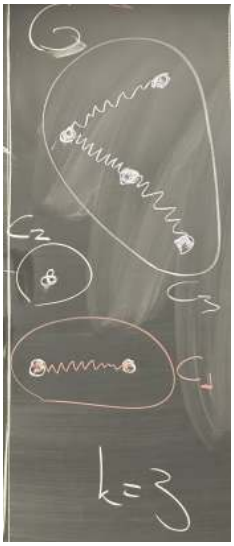
Kruskal's algorithm can be implemented to have running time $O(m \log n)$ using union-find data structure.

Worst case: $O(m \log m)$ to sort edges + $O(m\alpha(n))$ for union-find

1.4 Maximum Spanning Clustering

Given a set U of n objects $p_1, p_2, \ldots, p_n$ and a distance between any pair of p_i and p_j

$$d(p_i, p_j) = d(p_j, p_i) \geq 0$$



Definition 1.17. We partition U into k sets $C_1, C_2, \ldots, C_k$ called clusters.

$$C_1, C_2, \ldots, C_k \text{ is a } k\text{-clustering of } U \text{ if } \bigcup_{i=1}^{k} C_i = U$$

Definition 1.18. **Maximum Spanning Clustering Problem**

Find a k-clustering of U,
First, define the distance between two clusters C_i and C_j as

$$\min_{\substack{i,j \in \{1,2,\ldots,k\} \\ i \neq j}} \quad \min_{p \in C_i, q \in C_j} d(p,q)$$

A partition of U into k sets $C_1, C_2, \ldots, C_k$ such that we maximize spanning.

$$\max_{C_1, C_2, \ldots, C_k} \quad \min_{\substack{i,j \in \{1,2,\ldots,k\} \\ i \neq j}} \quad \min_{p \in C_i, q \in C_j} d(p,q)$$

1.4.1 Single Linkage Algorithm

- Step 1: Initially, each object in U forms its own cluster.
- Step 2: While the number of clusters is greater than k, repeatedly merge the two distinct clusters $C \neq C'$ that minimize

$$\min_{p \in C,\, q \in C'} d(p,q).$$

Lemma 1.2

The Single Linkage Clustering Algorithm correctly computes a maximum-spacing k-clustering.

Proof

Consider the graph $G = (V, E)$ where $V = \{p_1, p_2, \ldots, p_n\}$ and E is the set of all pairs (u, v) that are used to merge clusters during Step 2 of the algorithm. Let

$$\lambda = \max_{uv \in E} d(u,v).$$

Let $C_1, C_2, \ldots, C_k$ be the clustering produced by the algorithm. By construction, if two points

p and q lie in different clusters, then $d(p, q) \geq \lambda$; otherwise, the algorithm would have merged their clusters earlier. Hence,

$$\min_{\substack{i,j \in \{1,\ldots,k\} \\ i \neq j}} \min_{p \in C_i,\, q \in C_j} d(p, q) \geq \lambda.$$

Suppose for the sake of contradiction that this clustering is not optimal. Let $C'_1, C'_2, \ldots, C'_k$ be an optimal k-clustering. Since the two clusterings are different, there exists an edge $uv \in E$ such that $u \in C'_i$ and $v \in C'_j$ for some $i \neq j$.

Therefore,

$$\min_{\substack{i,j \in \{1,\ldots,k\} \\ i \neq j}} \min_{p \in C'_i,\, q \in C'_j} d(p, q) \leq d(u, v) \leq \lambda.$$

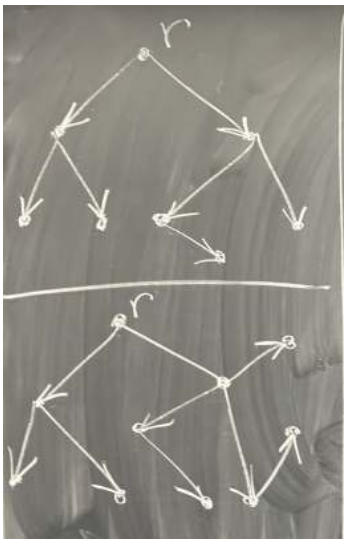
Combining the two inequalities, we conclude that the spacing of the algorithm's clustering is at least that of the optimal clustering, contradicting the assumption that the algorithm is not optimal. Hence, the Single Linkage algorithm produces a maximum-spacing k-clustering.

□

1.4.2 Minimum Cost Arborescence

Let $G = (V, E)$ be a directed graph and let $r \in V$ be a distinguished root. An *arborescence* rooted at r is a directed graph $T = (V, F)$ such that $F \subseteq E$ and

- For every vertex $v \in V$, there exists a directed path from r to v in T.
- The underlying undirected graph of T is a spanning tree of G.



Theorem 1.3

A directed graph $T = (V, F)$ is an arborescence rooted at r if and only if

- T contains no directed cycles, and
- every vertex $v \in V \setminus \{r\}$ has exactly one incoming edge in T.

Proof

($\Leftarrow$) Assume that T has no directed cycles and that every vertex $v \in V \setminus \{r\}$ has exactly one incoming edge. Then the total number of edges in T is $|V| - 1$. It suffices to show that there exists a directed path from r to every vertex $v \in V$.

Fix any $v \in V \setminus \{r\}$. Let (v_1, v) be the unique incoming edge of v. If $v_1 = r$, we are done. Otherwise, consider the unique incoming edge (v_2, v_1) of v_1. Continuing this process, either we eventually reach r, in which case a directed path from r to v exists, or we revisit a previously seen vertex, forming a directed cycle. The latter is impossible by assumption. Hence, a directed path from r to v exists.

Since every vertex is reachable from r, the underlying undirected graph of T is connected. Together with $|V| - 1$ edges, it follows that the undirected version of T is a spanning tree.

($\Rightarrow$) Suppose that T is an arborescence rooted at r. By definition, there exists a directed path from r to every vertex $v \in V$, so the underlying undirected graph of T is connected and acyclic.

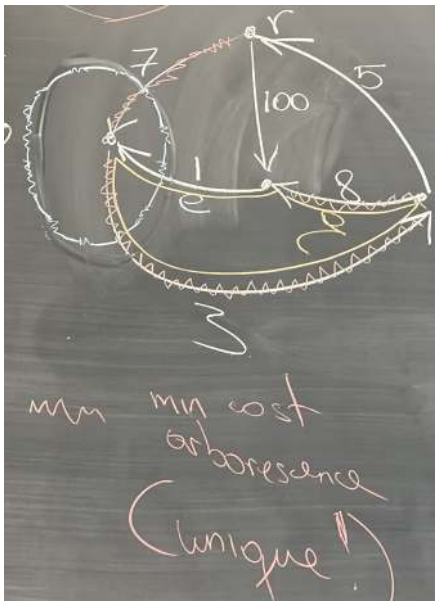
Moreover, every vertex $v \in V \setminus \{r\}$ must have at least one incoming edge, namely the last edge on the path from r to v. If some vertex had more than one incoming edge, or if r had an incoming edge, then a directed cycle would be formed. Therefore, each $v \in V \setminus \{r\}$ has exactly one incoming edge, and T contains no directed cycles.

□

1.4.3 Minimum Cost Arborescence Problem

Given a directed graph $G = (V, E)$, distinguished node $r \in V$, and edge costs $c_e, e \in E$ find on arborescence (rooted at r) of minimum total cost.

Remark 1.3



- Note that e is a min cost edge in a cut, but e is NOT in any min cost arborescence. (Cut property does not hold for arborescences)
- Note that g is a max cost edge in a cycle, but g is in every min cost arborescence. (Cycle property does not hold for arborescences)

1.4.4 Edmonds's Algorithm

Algorithm 4: Edmonds / Chu–Liu: Minimum-Cost Arborescence rooted at r

Input: Directed graph $G = (V, E)$, root r, edge costs $c(e)$

Output: Minimum-cost arborescence rooted at r

Step 1 (pick cheapest incoming edges).

For each $v \in V \setminus \{r\}$, let

$$f_v \in \arg\min_{(u,v)\in E} c(u, v), \qquad y_v := c(f_v).$$

Define reduced costs

$$c'(u, v) := c(u, v) - y_v \quad \text{for all } (u, v) \in E.$$

Let $F_0 := \{f_v : v \in V \setminus \{r\}\}$.

If F_0 has no directed cycle, return F_0 (it is an arborescence rooted at r).

Step 2 (contract one directed cycle).

Find a directed cycle C in F_0. Contract C into a supernode v_C to obtain $G_C = (V_C, E_C)$.

Step 3 (reweight edges that enter the cycle).

For every edge $(a \to x) \in E$ with $a \notin C$ and $x \in C$, create an edge $(a \to v_C)$ in E_C with cost

$$\tilde{c}(a \to v_C) := c'(a \to x) - c'(f_x).$$

For edges not entering C (outside–outside, or C–outside), keep reduced cost:

$$\tilde{c}(e) := c'(e).$$

Let the new root be r if $r \notin C$, otherwise root is v_C.

Step 4 (recurse).

Recursively compute a minimum-cost arborescence T_C in G_C w.r.t. costs $\tilde{c}$.

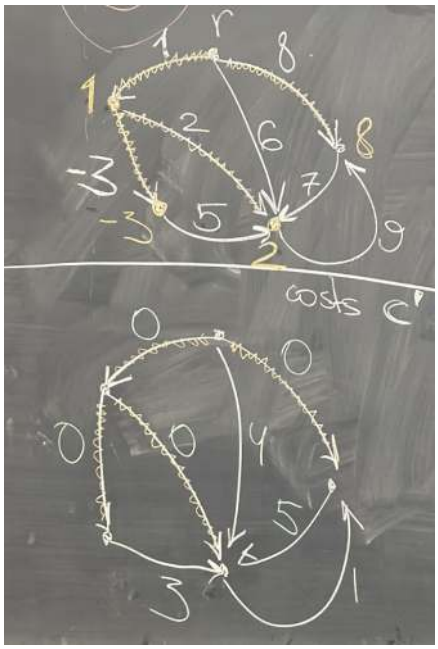
Step 5 (expand).

If T_C contains an entering edge $(a \to v_C)$, it corresponds to some original $(a \to x)$ with $x \in C$. Expand v_C back to C by:

- include all cycle edges $\{f_v : v \in C\}$,
- replace the single edge f_x (the one entering x on the cycle) by $(a \to x)$,
- keep all other edges from T_C (mapped back to original edges).

Return the expanded arborescence T.

Example



Define

$$y_v = \min_{(u,v)\in E} c(u,v) \quad \forall v \in V \setminus \{r\}$$

and

$$f_v = \arg\min_{(u,v)\in E} c(u,v) \quad \forall v \in V \setminus \{r\}$$

Consider

$$c'(u,v) = c(u,v) - y_v \quad \forall (u,v) \in E$$

Then

- For every $(u,v) \in E$, we have $c'(u,v) \geq 0$.
- For every arborescence F rooted at r, we have

$$c(F) = c'(F) + \sum_{v\in V\setminus\{r\}} y_v$$

- If $\cup_{v \in V \setminus \{r\}} f_v$ is an arborescence F rooted at r, then F is a min cost arborescence rooted at r. w.r.t. c'; so it is also min cost arborescence rooted at r w.r.t. c.

CHAPTER 2

Matroids

Definition 2.1. A **matroid** is a tuple $M = (U, I)$ where U is a ground set, and $I \subseteq 2^U$ is a collection of subsets of U such that:

- U is finite
- If $A \in I$ and $B \subseteq A$, then $B \in I$ (hereditary property)
- If $A, B \in I$ and $|A| > |B|$, then there exists $e \in A \backslash B$ such that $B \cup \{e\} \in I$ (exchange property)

Example

$U = \{1, 2, 3\}$, $I = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$

$A = \{2, 3\}, B = \{1\}, |A| = 2 > 1 = |B|$, there is $e = 2 \in A \setminus B$ such that $B \cup \{2\} \in I$.

Definition 2.2. A **basis** of $M = (U, I)$ is a (inclusion-wise) maximal indecent set, i.e., it is a set $A \in I$ such that for every $e \in U \backslash A$, we have $A \cup \{e\} \notin I$.

Definition 2.3.

1. **Uniform Matroid** Consider finite set U and a positive integer k; define a matroid M

on the ground set U with independent sets

$$I = \{A \subseteq U : |A| \leq k\}$$

Check by matroid definition 2.1 for three properties.

(a) Trivial.

(b) If $A \in I$ and $B \subseteq A$, then $|B| \leq |A|$ and $|A| \leq k$ so $|B| \leq k$ and thus $B \in I$.

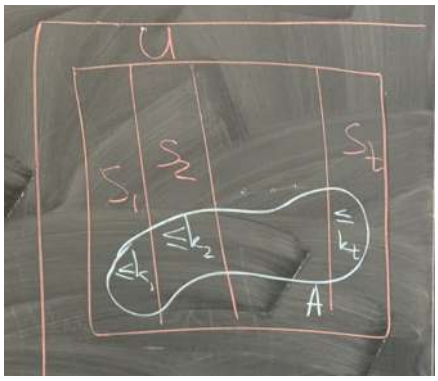
(c) If $A, B \in I$ and $|A| > |B|$, then

$$k \geq |A| \geq |B| + 1$$

and there exists $e \in A \backslash B$ then $|B \cup \{e\}| \leq |B| + 1 \leq k$ so $B \cup \{e\} \in I$.

2. **Partition Matroid** Consider a finite set U and its partition into $S_1, \ldots, S_t$, $U = S_1 \cup \cdots \cup S_t$ and positive integers $k_1, \ldots, k_t$. Define a matroid M on the ground set U with independent sets

$$I = \{A \subseteq U : |A \cap S_i| \leq k_i, \forall i = 1, \ldots, t\}$$



Check by matroid definition 2.1 for three properties.

(a) Trivial.

(b) If $A \in I$ and $B \subseteq A$, then for every $i = 1, \ldots, t$, we have

$$|B \cap S_i| \leq |A \cap S_i| \leq k_i$$

so $B \in I$.

(c) If $A, B \in I$ and $|A| > |B|$, then for every $i = 1, \ldots, t$

$$|B \cap S_i| \leq k_i \quad \text{and} \quad \exists j = 1, \ldots, t \text{ such that } |B \cap S_j| + 1 \leq |A \cap S_j| \leq k_j$$

so, there is $e \in S_j$ such that $e \in A \backslash B$

Let $B' = B \cup \{e\}$, then

- For every $i = 1, \ldots, t$ and $i \neq j$, we have

$$|B' \cap S_i| = |B \cap S_i| \leq k_i$$

- For $i = j$, we have

$$|B' \cap S_j| = |B \cap S_j| + 1 \leq k_j$$

So $B' = B \cup \{e\} \in I$.

3. **Linear Matroid** Consider a finite collection U of vector in $\mathbb{R}^d$, define a matroid M on the ground set U' with independent sets

$$I = \{A \subseteq U : A \text{ is linearly independent}\}$$

Check properties (exercise)!

4. **Graphic Matroid** Consider a finite undirected graph $G = (V, E)$, define a matroid M on the ground set E with independent sets

$$I = \{A \subseteq E : A \text{ is acyclic}\}$$

Check properties:

(a) Trivial.

(b) Trivial.

(c) If $A, B \in I$ and $|A| > |B|$, then

- (V, A) has $|V| - |A|$ connected components.
- (V, B) has $|V| - |B|$ connected components.

But $|V| - |A| < |V| - |B|$, so (V, A) has less connected components than (V, B). Thus, there exists one connected component of G_A that intersects of least two connected components of G_B.

So there is an edge $e \in A$ that connect two connected components of G_B, and so $B \cup \{e\}$ is acyclic.

Remark 2.1

Given a matroid, all bases m it has the same cardinality (follows from exchange property)

Remark 2.2

Spanning tree in a connected graph G are the bases of the graphic matroid defined on G.

2.1 Max Weight Independent Set Problem in Matroids (MWIS)

Definition 2.4. Given a matroid $M = (U, I)$ and weights $w_e, e \in U$ find a max weight independent set i.e. find $A \in I$ and

$$w(A) = \sum_{e \in A} w_e = \max_{B \in I} \sum_{e \in B} w_e$$

Remark 2.3

To find a maximum weight bases we can solve MWIS with weights $w'_e := w_e + M, e \in U$ for a sufficiently large M.

Indeed, for every independent set A we have $w'(A) = |A| \cdot M + \sum_{e \in A} w_e$ so the independent set with maximum weight

(To find a minimum weight bases we can solve MWIS with weights $w'_e := M - w_e, e \in U$ for a sufficiently large M.)

Assumption: All weights are non-negative, i.e. $w_e \geq 0$ for every $e \in U$.

Note that we can remove negative elements; i.e., consider

$$U_{\geq 0} = \{e \in U, w_e \geq 0\}$$

$$I_{\geq 0} = \{A \in I, A \subseteq U_{\geq 0}\}$$

Leads to another matroid $M_{\geq 0} = (U_{\geq 0}, I_{\geq 0})$.

So solving MWIS on $M = (U, I)$ is the same as solving MWIS on $M_{\geq 0} = (U_{\geq 0}, I_{\geq 0})$.

Remark 2.4

Note that there is an optimal solution A for MWIS on $M_{\geq 0} = (U_{\geq 0}, I_{\geq 0})$ that is a basis for $M_{\geq 0}$

2.1.1 Greedy Algorithm for MWIS

Greedy Algorithm (Assumption $w_e \geq 0$ for every $e \in U$):

1. Sort elements in U as $e_1, e_2, \ldots, e_n$ so that $w_{e_1} \geq w_{e_2} \geq \ldots \geq w_{e_n}$
   Set $A = \emptyset$ and $i = 1$.
2. While $i \leq n$: If $A \cup \{e_i\} \in I$, then $A \leftarrow A \cup \{e_i\}$. $i \leftarrow i + 1$.

Theorem 2.1

Given $w_e \geq 0$ for $e \in U$, Greedy Algorithm correctly computes MWIS.

Proof

Proof by contradiction, assume that A was output by Greedy Algorithm, and A^* is a MWIS such that $w(A) < w(A^*)$. We can assume that both A and A^* are bases (by the previous remark 2.4).

So,

$$A = \{a_1, a_2, \ldots, a_k\} \quad A^* = \{a_1^*, a_2^*, \ldots, a_k^*\}$$

where

$$w_{a_1} \geq w_{a_2} \geq \ldots \geq w_{a_k} \quad w_{a_1^*} \geq w_{a_2^*} \geq \ldots \geq w_{a_k^*}$$

For $j = 0, 1, \ldots, k$, define

$$A_0 = \emptyset; \quad j \geq 1, \quad A_j = \{a_1, a_2, \ldots, a_j\}$$

Similarly, define

$$A_0^* = \emptyset; \quad j \geq 1, \quad A_j^* = \{a_1^*, a_2^*, \ldots, a_j^*\}$$

Consider the minimum $j = 1, 2, \ldots, k$ such that $w(A_{j-1}) \geq w(A_{j-1}^*)$ but $w(A_j) < w(A_j^*)$.

Consider A_{j-1} and A_j^*. We have $|A_j^*| > |A_{j-1}|$, so by exchange property there is $e \in A_j^* \setminus A_{j-1}$ such that $A_{j-1} \cup \{e\} \in I$.

Note that

$$w_e \geq w_{a_j^*}$$

since $w_{a_1^*} = \min_{f \in A_j^*} w_f$ and $e \in A_j^*$.

Also,

$$w_{a_j^*} > w_{a_j}$$

since $w(A_{j-1}) \geq w(A_{j-1}^*)$ but $w(A_{j-1}) + w_{a_j} < w(A_{j-1}^*) + w_{a_j^*}$.

We have $w_e \geq w_{a_j^*} > w_{a_j}$ and $A_{j-1} \cup \{e\} \in I$, contradiction, $e \notin A_{j-1}$.

□

2.2 Matroid Intersection Problem

Giving two matroids $M_1 = (U_1, I_1)$ and $M_2 = (U_2, I_2)$, find a maximum weight set that is independent in both matroids, i.e. find

$$\arg \max_{A \in I_1 \cap I_2} w(A)$$

2.2.1 Maximum Weight Bipartite Matching

Given a bipartite graph $G = (V, E)$ with bipartition $V = W \cup U$ and edge weights w_e for $e \in E$, find a maximum weight bipartite matching if

$$\forall u \in U \quad |\delta(u) \cap M| \leq 1$$
$$\forall w \in W \quad |\delta(w) \cap M| \leq 1$$

Define $U_1 = U_2 = E$,

$$I_1 = \{A \subseteq E, \forall u \in U, |\delta(u) \cap A| \leq 1\}$$
$$I_2 = \{A \subseteq E, \forall w \in W, |\delta(w) \cap A| \leq 1\}$$

Then the maximum weight matching problem can be solved as a matroid intersection problem on $M_1 = (U_1, I_1)$ and $M_2 = (U_2, I_2)$.

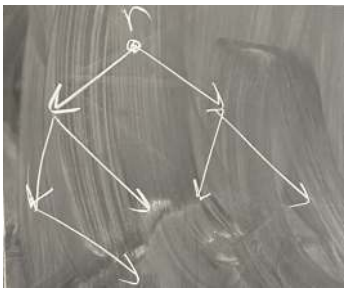
2.2.2 Minimum Weight Common Basis

Given two matroids $M_1 = (U_1, I_1)$ and $M_2 = (U_2, I_2)$ weight w_e for $e \in U_1 \cup U_2$, find a minimum weight A that is a basis in both M_1 and M_2. i.e. find

$$\arg \min_{A \in B_1 \cap B_2} w(A), \quad \text{where } B_1, B_2 \text{ are the set of bases of } M_1, M_2$$

2.2.3 Minimum Cost Arborescence Problem

Given a directed graph $G = (V, E)$ and a distinguished node $r \in V$, edge weights w_e for $e \in E$, find a minimum weight arborescence F rooted at r.

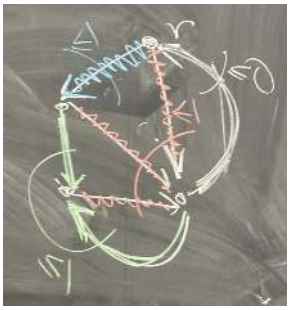


Let $U_1 = U_2 = E$,

Define $M_1 = (U_1, I_1)$ as the graphic matroid on the "undirected version" of G.

Define $I_2 = \{A \subseteq E : \quad |\delta^{\text{in}}(v) \cap A| \leq 1, \forall v \in V \setminus \{r\}, \quad |\delta^{\text{in}}(r) \cap A| \leq 0\}$.

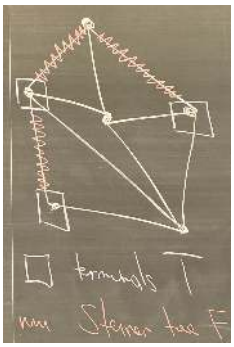
So Minimum Cost Arborescence can be solved as Minimum Weight Common Basis problem.



2.2.4 Minimum Steiner Tree Problem

Definition 2.5.

Given a graph $G = (V, E)$, costs $c_e, e \in E$ with $c_e \geq 0$, and a set $T \subseteq V$ called **terminals**. Find a tree F of minimum cost $c(F) = \sum_{e \in F} c_e$ that connects all the terminals T.



Useful transformation

Given a graph $G = (V, E)$, construct a complete graph $G' = (V, E')$ with costs

$$c'_{uv} = \text{length of the shortest path in } G \text{ with respect to } c \text{ between } u \text{ and } v.$$

In G' with cost $c'_e, e \in E'$, we have $\triangle$-inequalities: For every $u, v, w \in V$, $c'_{uw} \leq c'_{uv} + c'_{vw}$.

Lemma 2.1

(a) Given a Steiner tree F in $G = (V, E)$ with terminals T, we have that F is also a Steiner tree in $G' = (V, E')$ with terminals T.

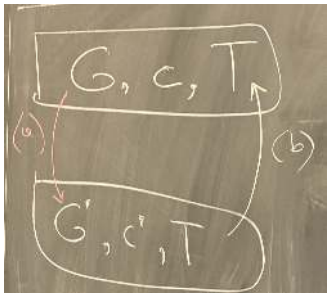
Moreover,

$$c'(F) \leq c(F).$$

(b) Given a Steiner Tree F' in $G' = (V, E')$ with terminals T, there is a Steiner tree F'' in $G = (V, E)$ with terminals T.

Moreover,

$$c(F'') \leq c'(F').$$



Proof

(a) F connects all terminals in T in G'. For every edge $e = (u, v) \in F$, we have $c'_{uv} \leq c_{uv}$. Thus,

$$c'(F) = \sum_{e \in F} c'_e \leq \sum_{e \in F} c_e = c(F).$$

(b) Consider $\cup_{uv \in F'} P_{uv}$, where P_{uv} is the shortest u-v path in G with respect to $c_e, e \in E$. So, $c'_{uv} = \sum_{e \in P_{uv}} c_e$.

Thus,

$$c(\cup_{uv \in F'} P_{uv}) \leq \sum_{uv \in F'} c(P_{uv}) = \sum_{uv \in F'} c'_{uv} = c'(F').$$

Note that $\cup_{uv \in F'} P_{uv}$ connects all the terminals in T.

Select F'' from $\cup_{uv \in F'} P_{uv}$ by considering edge by edge, and removing edges contained in a cycle.

This way we get a tree F'' connecting all the terminals in T and

$$c(F'') \leq c'(F').$$

□

Algorithm Solve MST on the graph $G'[T] = (T, E'[T])$ with respect to costs $c'_e, e \in E'[T]$.

Obtain MST F'. Obtain F'' from F' as in Lemma (b) output F''.

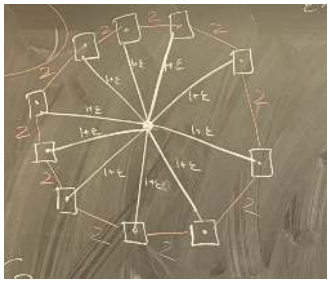
Remark 2.5

Clearly, this algorithm outputs F'' with cost $\leq MST(G', e', T)$
(Minimum cost of spanning tree in $G'[T]$ with respect to c')

There are instances where

$$2 \cdot OPT(G, c, T) \approx MST(G', c', T) = c(F'')$$

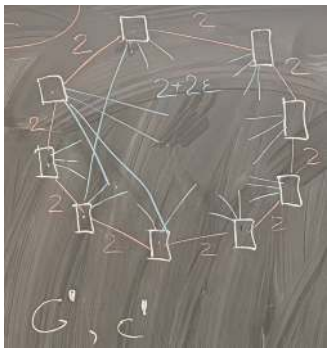
$$OPT(G, c, T) = (1 + \epsilon) \cdot n, \quad |T| = n, \epsilon > 0 \text{ small}$$



$$\begin{aligned} c(F'') &= MST(G', c', T) \\ &= 2n - 2 \end{aligned}$$

Note

$$\frac{MST(G', c', T)}{OPT(G, c, T)} = \frac{2n - 2}{(1 + \epsilon)n} \approx \frac{2}{1 + \epsilon} \approx 2$$



Algorithm for Steiner Tree Problem

1. Compute MST in $G'[T] = (T, E'[T])$ with respect to $c'_e, e \in E'[T]$. Obtain MST F' (A Steiner tree in G' with cost c').

2. Obtain a Steiner tree F'' in G from F' with cost $c(F'') \leq c'(F')$. (By Lemma 2.1 (b)) output F''.

Recall

There are instances where

$$c(F'') = MST(G', c', T) \approx 2 \cdot OPT(G, c, T)$$

- $c(F'')$ cost of output for the above algorithm.
- $MST(G', c', T)$ cost of MST in $G'[T]$ with respect to c'.
- $OPT(G, c, T)$ min cost of a Steiner tree in G w.r.t. c and terminals T.

Lemma 2.2

$MST(G', c', T) \leq 2 \cdot OPT(G, c, T)$.

Remark 2.6

$OPT(G, c, T) = OPT(G', c', T)$, so it is enough to show

$$MST(G', c', T) \leq 2 \cdot OPT(G', c', T)$$

Proof

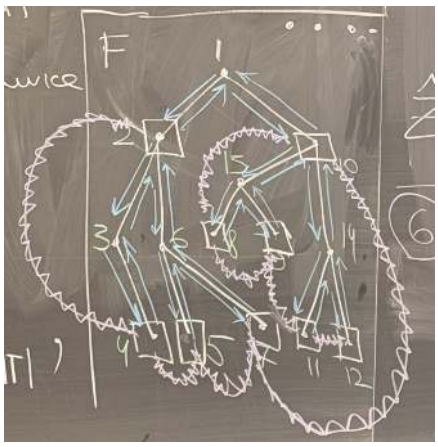
Consider a minimum cost Steiner tree F in G' w.r.t. c' and terminals T.

So, $c'(F) = OPT(G', c', T)$.

We can construct a spanning tree H in the graph $G'[T]$ such that

$$c'(H) \leq 2 \cdot c'(F)$$

So $MST(G', c', T) \leq 2 \cdot c'(F)$

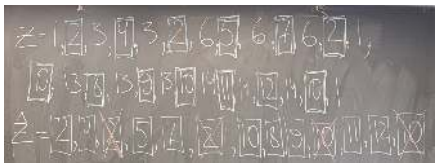


Consider a close walk Z that traverses each edge in F exactly twice,

$$Z = u_1, u_2, \ldots, u_k, u_{k+1} = u_1$$

Consider the subsequence of Z that only contains vertices in T,

$$\hat{Z} = u_{i_1}, u_{i_2}, \ldots, u_{i_{|T|}}$$



Where u_{i_1} is the first terminal visited in Z and u_i is the first terminal used after $u_{i_{j-1}}$ that has NOT been visited defined.

Consider the path H (spanning tree in $G'[T]$) corresponding to $\hat{Z}$.

$$c'(H) = c'(\hat{Z}) = \sum_{j=1}^{|T|-1} c'(u_{i_j}, u_{i_{j+1}}) \leq \sum_{j=1}^{k} c'(u_j, u_{j+1}) = c'(Z) = 2 \cdot c'(F)$$

□

Definition 2.6. An **α-approximation algorithm**, $\alpha \geq 1$, for a minimization problem is an efficient algorithm such that for every input instance the algorithm returns a solution if cost at most α optimal cost.

Remark 2.7

Algorithm for Steiner tree problem is a 2-approximation algorithm for the Minimum Cost Steiner Tree problem.

CHAPTER 3

Linear Programming

Note that Minimum Cost Steiner Tree problem (instance (G, c, T)) can be formulated as an integer problem.

$$\begin{aligned}
&\min \sum_{e \in E} c_e x_e \\
&\text{s.t.} \\
&\quad \sum_{e \in \delta(S)} x_e \geq 1, \forall S \subseteq V, S \cap T \neq \emptyset, T \setminus S \neq \emptyset \\
&\quad x \geq 0, x \in \mathbb{Z}^E
\end{aligned}$$

Note that we can also add constraints to the above IP without changing feasible solution

$$\sum_{e \in \delta(S)} x_e \geq 0, \forall S \subseteq V, S \cap T = \emptyset \text{ or } T \subseteq S$$

Network connectivity problems usually can be formulated using out requirement functions $f : 2^V \to \{0, 1\}$, where we want to find a set of edges F such that $|\delta(S) \cap F| \geq f(S)$ for every $S \subseteq V$.

$$\begin{aligned}
&\min \sum_{e \in E} c_e x_e \\
&\text{s.t.} \\
&\sum_{e \in \delta(S)} x_e \geq f(S), \forall S \subseteq V \\
&x \geq 0, x \in \mathbb{Z}
\end{aligned}$$

Simplified IP to

$$\begin{aligned}
&\min \sum_{e \in E} c_e x_e \\
&\text{s.t.} \\
&\sum_{e \in \delta(S)} x_e \geq 1 \text{ for every } S \in D \\
&x \geq 0, x \in \mathbb{Z}
\end{aligned}$$

Here $D = \{S \subseteq V : f(S) = 1\}$ is the set of **demand cuts**.

For example in Minimum Cost Steiner Tree

$$f : 2^V \to \{0, 1\}, f(S) = \begin{cases} 1 & S \cap T \neq \emptyset, T \setminus S \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

For every $S \in 2^V$

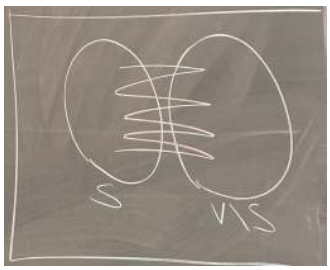
Definition 3.1. A cut requirement function

$$f : 2^V \to \{0, 1\}$$

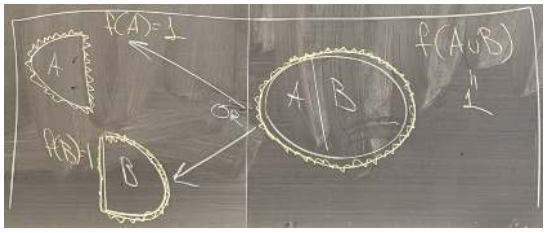
is called **proper** if

(i) $f(V) = 0$ and $f(\emptyset) = 0$.

(ii) $f(S) = f(V \setminus S)$ for every $S \subseteq V$.



(iii) For every $A \subseteq V, B \subseteq V$ such that $A \cap B = \emptyset$, if $f(A \cup B) = 1$, then $f(A) = 1$ or $f(B) = 1$. (Not exclusive it is possible that $f(A) = 1$ and $f(B) = 1$)



Note that cut requirement function for Minimum Cost Steiner Tree is proper.

3.1 Minimum Cost Generalized Steiner Tree Problem

Given a graph $G = (V, E)$, cost $c_e \geq 0, e \in E$, and k pairs $(s_i, t_i), i = 1, 2, \ldots, k$, $s_i, t_i \in V$, $s_i \neq t_i$, find a set of edges $F \subseteq E$ such that s_i and t_i are connected in (V, F) for every $i = 1, 2, \ldots, k$ and such that $c(F) = \sum_{e \in F} c_e$ is minimum.

The corresponding cut requirement function is

$$f : 2^V \to \{0, 1\}, \text{ for every } S \subseteq V$$

$$f(S) = \begin{cases} 1 & \text{if } |S \cap \{s_i, t_i\}| = 1 \text{ for some } i = 1, 2, \ldots, k \\ 0 & \text{otherwise} \end{cases}$$

Check that $f : 2^V \to \{0, 1\}$ is proper.

(i) $f(V) = 0$ and $f(\emptyset) = 0$ is trivial.

(ii) $f(S) = f(V \setminus S)$ for every $S \subseteq V$ is trivial.

(iii) Consider $A, B \subseteq V, A \cap B = \emptyset$ and $f(A \cup B) = 1$. Then for some $i = 1, 2, \ldots, k$, $|(A \cup B) \cap$

$\{s_i, t_i\}| = 1$. WLOG, $(A \cup B) \cap \{s_i, t_i\} = \{s_i\}$, then $s_i \in A$ or $s_i \in B$. So $f(A) = 1$ or $f(B) = 1$.

LP relaxation for Network Design IP

$$\begin{aligned} \min & \sum_{e \in E} c_e x_e \\ \text{s.t.} & \\ & \sum_{e \in \delta(S)} x_e \geq 1 \text{ for every } S \in D \\ & x \geq 0 \end{aligned}$$

(Primal LP)

$$\begin{aligned} \max & \sum_{S \in D} 1 \cdot y_S \\ \text{s.t.} & \sum_{S \in D : e \in \delta(S)} y_S \leq c_e, \forall e \in E \\ & y_S \geq 0, \forall S \in D \end{aligned}$$

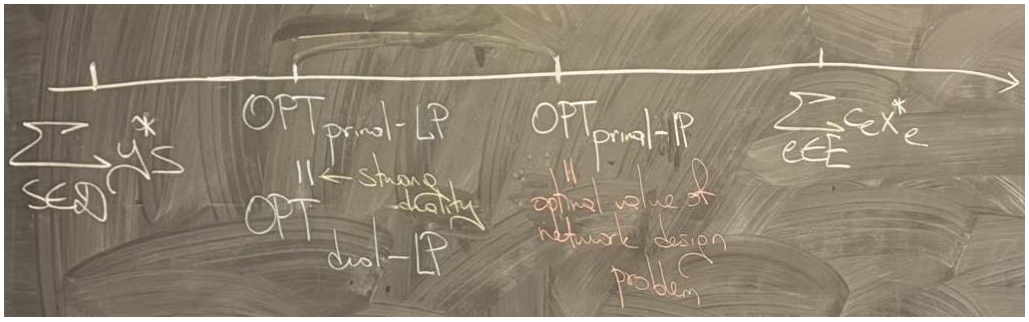
(Dual LP)

Out plan find $x^* \in \{0,1\}^E$ feasible for primal LP and y^* feasible for dual LP such that

$$\underbrace{\sum_{e \in E} c_e x_e^*}_{\text{value for } x^*} \leq 2 \underbrace{\sum_{S \in D} y_S^*}_{\text{value for } y^*}$$

So we would get

$$\sum_{e \in E} c_e x_e^* \leq 2 \cdot \text{OPT}_{\text{primal IP}}$$



3.2 Network Design Framework

Definition 3.2. Given a graph $G = (V, E)$, edge costs $c_e \geq 0$ for $e \in E$, and a cut requirement function $f : 2^V \to \{0, 1\}$, find a set of edges F minimizing $c(F) = \sum_{e \in F} c_e$ such that for every $S \subseteq V$,

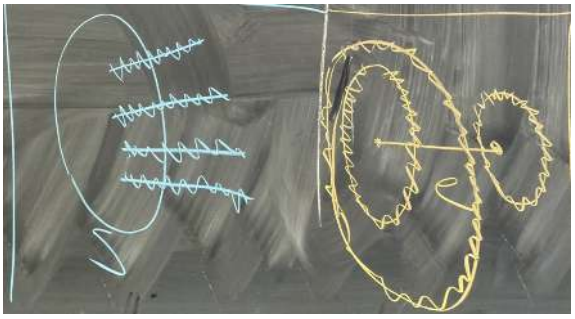
$$|F \cap \delta(S)| \geq f(S)$$

$D = \{S \subseteq V : f(S) = 1\}$

Consider LP relaxation for the problem and its dual:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq 1 \quad \forall S \in D \\ & x_e \geq 0 \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{S \in D} 1 \cdot y_S \\ \text{s.t.} \quad & \sum_{S \in D : e \in \delta(S)} y_S \leq c_e \quad \forall e \in E \\ & y_S \geq 0 \end{aligned}$$



We assume that the function f is proper.

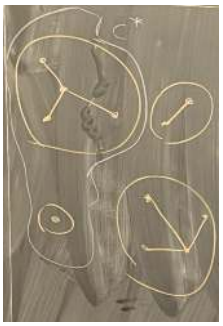
Plan is to find a feasible set F and a dual feasible solution y^* such that

$$c(F) \leq 2 \sum_{S \in D} y^*_S$$

leading two a 2-approximation algorithm.

Proposition

$F \subseteq E$ is feasible for the network design problem if and only if for every connected component C of (V, F), we have $f(C) = 0$.



Proof

($\Rightarrow$) For the sake of contradiction, assume F is feasible but for some of the connected components C we have $f(C) = 1$. Then

$$|F \cap \delta(C)| = 0 < 1 = f(C)$$

So F is not feasible, which is a contradiction.

($\Leftarrow$) For the sake of contradiction, assume $f(C) = 0$ for each connected component C of (V, F) but F is NOT feasible. So there is $C^* \subseteq V$ such that

$$1 = f(C^*) > |F \cap \delta(C^*)| = 0$$

Then $C^* = \cup_{i=1}^{k} C_i$ where C_i's are connected components of (V, F).

By (iii) of the definition of proper function, $f(C_i) = 1$ for some $i \in I$.

This is a contradiction.

□

3.2.1 Primal-Dual Algorithm

Step 1: $F \leftarrow \emptyset$, $y_S \leftarrow 0$ for all $S \in D$.

Step 2: Let Φ be the set of connected components C of (V, F) such that $f(C) = 1$.

If $\Phi = \emptyset$, go to Step 3.

Otherwise, increase y_S for all $S \in \Phi$ uniformly until some edge e becomes tight,

Update $F \leftarrow F \cup \{e\}$ and go to Step 2.

Step 3: Reverse delete edges in F in the order they were added. $F = \{e_t, e_{t-1}, \ldots, e_1\}$ where e_t is the last edge added. For $i = t, t-1, \ldots, 1$, if $F \setminus \{e_i\}$ is feasible, update $F \leftarrow F \setminus \{e_i\}$.

Return F.

Example