Approximation Algorithms for Network Design

IT 584: Approximation Algorithms Project Presentation

202101040 - Abhinav Agrawal
202103005 - Kanishk Dad
202103017 - Dhruv Shah
202103022 - Vatsal Shah
202103040 - Pranav Patel
202103052 - Vraj Thakkar

April 25, 2024

Introduction

- Network design problems have many practical applications ranging from the design process of **telecommunication and traffic networks to VLSI chip design.**
- Many of these design problems are proven to be **computationally intractable**.
- We discuss some Approximation Algorithms techniques as a possible way to navigate this deadlock.
- Here we will discuss the Approximation techniques on a very popular network design problem, **the minimum spanning tree.**

Table of contents

The Greedy Approach for MST

What is a MST?

$G = (V, E)$ be a connected, undirected graph with vertex set V and edge set E. A minimum spanning tree T of G is a spanning tree of G such that the sum of the weights of its edges is minimized.

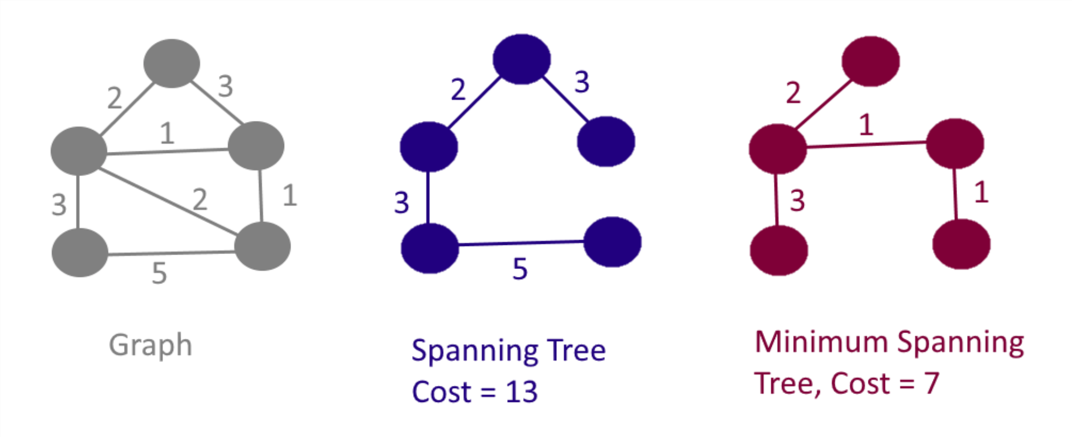


Figure 1: Graph to MST

Kruskal's Algorithm

Given a connected graph G(V,E):

1. Sort all edges in ascending order of their weights.
2. Pick the smallest edge and check if it forms a cycle with the Spanning Tree formed so far.
3. Repeat the step until V-1 edges are covered.

Proof for Kruskal's Algorithm

We will prove that each step we computed is correct for obtaining our optimal solution.

- Another way to think about Kruskal's algorithm is through the cut property.
- At any step we are crossing a cut(connecting previously unconnected groups of vertices) if the vertex is previously disconnected and has the minimum weight.

Proof for Kruskal's Algorithm

- This assumes that in each iteration we are connecting S and V-S such that the connecting edge, say (u,v), is the edge of the minimum cost.

- Assume that it was not the case and there was another MST possible. There will be an edge (u',v') that cuts S, V-S in this tree. But we know that (u,v) is the minimum edge that cuts S and V-S.

Iterative Rounding

LP formation

Consider x_e primal variable for each edge $e \in E$, and c_e is the cost of each edge. We want to

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e x_e \\ \text{subject to} \quad & \mathcal{X}(E(V)) = n - 1 \\ & \mathcal{X}(E(S)) \leq |S| - 1, \quad S \subset V, \\ & x_e \geq 0, \qquad\qquad\quad \forall e \in E \end{aligned} \tag{1}$$

where $|V| = n - 1$,

- E(S) : set of edges whose both endpoints lies in S
- $\mathcal{X}(A) = \sum_{e \in A} x_e$

Lemma

Let G be a connected graph with at least two vertices and let x^* be a basic solution of (1) and let $E^* = \{e \in E : x_e^* > 0\}$. There exists a node v such that v is exactly incident to one edge in E^*.

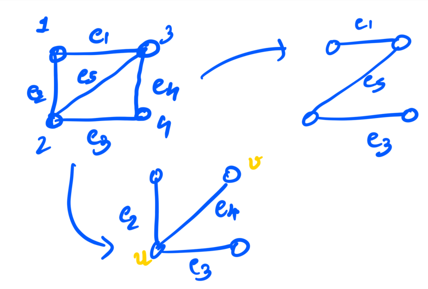


Figure 2: Caption

- Basic solution : $x^* = (x_{e_1}, x_{e_3}, x_{e_3}, x_{e_4}, x_{e_5})$,
- Each node is covered by at least one edge in E^*.
- Due to constraints of LP we have $x_{uv} = 1$

Algorithm 2 Iterative MST Algorithm

Require: Connected graph G = (V, E)

1: $F = \phi$
2: **while** $V(G) \neq \phi$ and $|V| > 1$ **do** :
3: Compute optimum basic solution x^* of (3) and remove all edges with $x^*_e = 0$ from G.
4: Find vertex u as in Lemma (4.1) and let uv be the single support edge incident to it.
5: Add edge uv to F.
6: Delete u and all incident edges from G.
7: **end while**
8: Return F.

Figure 3: Algorithm for MST

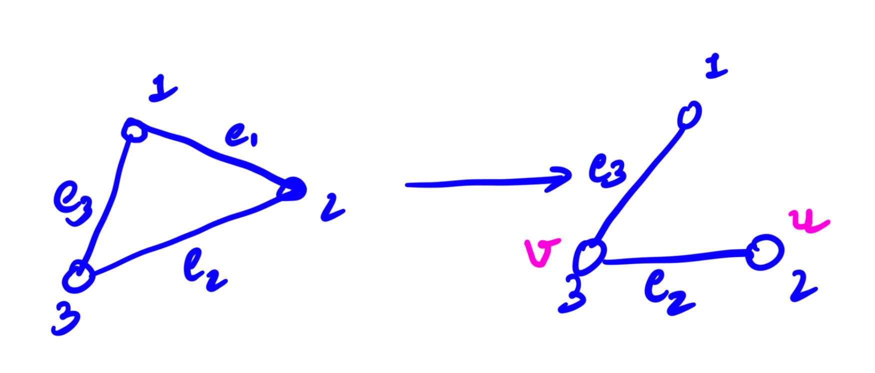


Figure 4: Algorithm Example

-

Iterative Algorithm : Correctness

- Case 1: For one vertex is G, algorithm returns ϕ.
- Case 2: For more than two vertices
  Obtain $G' = (V', E')$ from G by deleting vertex u and its incident edges.
- Let x' be the projection of x^* onto edge set of G. Hence $x' \in E'$ and x and $x'_e = x^*_e$.
- x' is feasible for graph G'.

Iterative Algorithm : Correctness

- For iteration 2 and on, we have $c(F') \leq \sum_{e \in E'} c_e x_e^*$.
- For $F = F' \cup uv$,

$$c(F) = c(F') + c_{uv} \leq \sum_{e \in E} c_e x_e^*$$

Randomized Rounding

Randomized Rounding Algorithm

- The intuition behind this algorithm is to let the decision variables x_i from the LP solution denote the probability of edge i getting selected.

Randomized Rounding Algorithm

- The intuition behind this algorithm is to let the decision variables x_i from the LP solution denote the probability of edge i getting selected.
- We later see how this approach yields a tree with an expected cost of at most $O(Z^* \log(n))$, where Z^* is the cost of any feasible solution to the minimum spanning tree LP relaxation.

Algorithm 3 Random-Round-MST

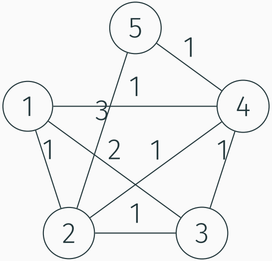
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  Let A_i be the set of edges obtained by picking each edge e independently with probability x_e.
until $G_i = (V, \bigcup_{j \le i} A_j)$ is connected
return any spanning tree of G_i

Figure 5: Randomized Rounding Algorithm

Consider the graph:

Vertices: $V = \{1, 2, 3, 4, 5\}$

Edges: $E = \{(1,2,1),(1,3,2),(1,4,1),(2,3,1),(2,4,1),(2,5,3),(3,4,1),(4,5,1)\}$



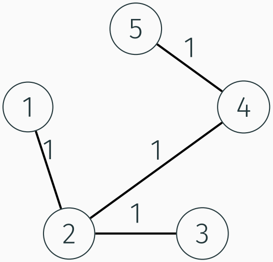
Intuation: Let the Probability of getting a head be x_i. Now for each edge, toss a coin. If it's a head(success), include the edge; otherwise, exclude it.

- Iteration 1:
  - Example selection: $A_1 = \{(1,2),(2,3),(4,5)\}$
  - Check connectivity: G_1 is not connected.

Randomized Rounding Algorithm: Iterations

- **Iteration 1:**
  - Example selection: $A_1 = \{(1,2),(2,3),(4,5)\}$
  - Check connectivity: G_1 is not connected.
- **Iteration 2:**
  - Example selection: $A_2 = A_1 \cup \{(1,4),(2,4)\}$
  - Check connectivity: G_2 is connected.

- **Iteration 1:**
  - Example selection: $A_1 = \{(1,2),(2,3),(4,5)\}$
  - Check connectivity: G_1 is not connected.
- **Iteration 2:**
  - Example selection: $A_2 = A_1 \cup \{(1,4),(2,4)\}$
  - Check connectivity: G_2 is connected.
- G_2 is connected, so we return any spanning tree of G_2.



Lemma

Given any (multi)graph $H = (U, E)$ and a feasible solution $\{x_e\}_{e \in E}$ to the Primal Problem for this graph, suppose we pick each edge with probability x_e independently. Then the number of components is at most $0.9|U|$ with probability at least $\frac{1}{2}$.

Proof.

- Isolated Vertex - Vertex not incident to any edge selected by the random process.
- Probability that a vertex u is isolated is: $\prod_{e\in\delta(\{u\})}(1-x_e)$
- Since, $(1-x) \leq e^{-x}$ and $\sum_{e\in\delta(\{u\})} x_e \geq 1$
- We have $\prod_{e\in\delta(\{u\})}(1-x_e) \leq e^{-\sum_{e\in\delta(\{u\})} x_e} \leq \frac{1}{e}$.
- Now, by the linearity of expectation, expected number of isolated vertices are at most $|U|/e$.
- Applying Markov Inequality, we get $\Pr(X \leq 2|U|/e) \geq \frac{|U|/e}{2|U|/e} = \frac{1}{2}$.

$\square$

Proof.

- Since, number of non-isolated nodes must be part of component of size at least 2.
- Total Number of Components = Number of Isolated Nodes + $\frac{1}{2}$ Number of remaining Nodes.
- Therefore, Total Number of Components is at most $|U|(2/e + 1/2(1 - 2/e)) < 0.9|U|$ with probability at least $\frac{1}{2}$

$\square$

Theorem

The expected cost of the tree returned by MST-Rand-Round is $O(\log n)$ times the cost of the LP solution Z^*.

Proof.

- We show that the expected number of rounds before the procedure stops is $\log n$.
- Let $L = 20 \log n$. **We claim** that the graph $G_L = (V, \cup_{j \le i} A_j)$ is disconnected with probability at most $1/2$.
- If G_L is disconnected, an identical argument shows that G_{2L} will also be disconnected with probability at most $1/2$, and so on.
- Therefore, the expected number of rounds will be at most $2L$, i.e. $O(\log n)$. For instance, consider a graph with $n = 8$ vertices. Let $L = 20 \log 8 = 60$. It means that after 60 rounds, the expected number of rounds for the algorithm to stop is $2L = 120$.

$\square$

Proof.

- To prove G_L is disconnected with probability at most $1/2 \sim G_L$ is connected with probability at least $1/2$;
- Let C_i denote the number of components of the graph G_i. An index i is considered successful if either:
  - G_{i-1} is connected (i.e., if $C_{i-1} = 1$)
  - if $C_i \leq 0.9 \cdot C_{i-1}$.
- **Claim:** The probability of i being successful is at least $1/2$ **regardless of all random choices made in previous rounds.**

$\square$

Proof.

Claim: The probability of i being successful is at least $1/2$.

Proof:

- If G_{i-1} is connected, then i is always successful.
- Otherwise, let H_{i-1} be the graph on C_{i-1} vertices obtained by contracting all edges in G_{i-1}. Since each cut in H_{i-1} corresponds to a cut in G_{i-1},
- $\sum_{e \in \delta(S)} x_e \geq 1$ holds for H_{i-1}. By Lemma, the number of components after another round of randomly adding edges will cause $C_i \leq 0.9 \cdot C_{i-1}$ with probability at least $1/2$.

$\square$

Proof.

- The probability that the graph G_L is not connected is bounded above by the probability that a sequence of L independent unbiased coin-flips contains fewer than $10 \log n$ heads.
- Note that while the events in the random rounding process are not independent, we proved a lower bound of $\frac{1}{2}$ on the probability of success regardless of the history.
- Since the number of rounds L is $20 \log n$, we observe that $10 \log n$ "heads" suffice to ensure the graph remains connected with high probability.
- Hence the probability we see fewer than $10 \log n$ heads (and hence the probability that G_L is not connected) is at most $\frac{1}{2}$, which completes the proof.

$\square$

Matching Based Augmentation

Matching Based Augmentation

Matching-based augmentation approximates minimum spanning trees (MSTs) in graphs. The idea is to iteratively grow a spanning tree by augmenting it with edges obtained from near-perfect matchings of the graph's nodes. It is most useful when the graph is represented as a metric space.

Matching Based Augmentation: Algorithm

Let a metric space (V,d) where the cost of matching two nodes (u,v) is the distance d(u,v) between them. The algorithm is as follows:

1. Start with a metric space (V,d) and define V_0 = V.

Matching Based Augmentation: Algorithm

Let a metric space (V,d) where the cost of matching two nodes (u,v) is the distance d(u,v) between them. The algorithm is as follows:

1. Start with a metric space (V,d) and define V_0 = V.
2. Then find a min-cost near-perfect matching M_0 (which leaves at most one node unmatched) on the nodes V_0.

Matching Based Augmentation: Algorithm

Let a metric space (V,d) where the cost of matching two nodes (u,v) is the distance d(u,v) between them. The algorithm is as follows:

1. Start with a metric space (V,d) and define V_0 = V.
2. Then find a min-cost near-perfect matching M_0 (which leaves at most one node unmatched) on the nodes V_0.
3. For each matched pair, pick one of the nodes and add them to the set V_1; also add in the unmatched node, if any.

Matching Based Augmentation: Algorithm

Let a metric space (V,d) where the cost of matching two nodes (u,v) is the distance d(u,v) between them. The algorithm is as follows:

1. Start with a metric space (V,d) and define V_0 = V.
2. Then find a min-cost near-perfect matching M_0 (which leaves at most one node unmatched) on the nodes V_0.
3. For each matched pair, pick one of the nodes and add them to the set V_1; also add in the unmatched node, if any.
4. Find a min-cost near-perfect matching M_1 on V_1; pick one of the newly matched vertices (and the unmatched vertex, if any) and form V_2.

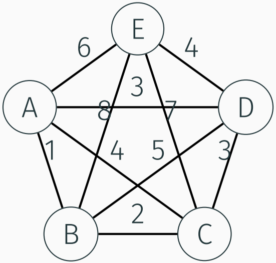
Let a metric space (V,d) where the cost of matching two nodes (u,v) is the distance d(u,v) between them. The algorithm is as follows:

1. Start with a metric space (V,d) and define V_0 = V.
2. Then find a min-cost near-perfect matching M_0 (which leaves at most one node unmatched) on the nodes V_0.
3. For each matched pair, pick one of the nodes and add them to the set V_1; also add in the unmatched node, if any.
4. Find a min-cost near-perfect matching M_1 on V_1; pick one of the newly matched vertices (and the unmatched vertex, if any) and form V_2.
5. Continue the same as in *step* 4 until $|V_t| = 1$ for some t.

Let a metric space (V,d) where the cost of matching two nodes (u,v) is the distance d(u,v) between them. The algorithm is as follows:

1. Start with a metric space (V,d) and define V_0 = V.
2. Then find a min-cost near-perfect matching M_0 (which leaves at most one node unmatched) on the nodes V_0.
3. For each matched pair, pick one of the nodes and add them to the set V_1; also add in the unmatched node, if any.
4. Find a min-cost near-perfect matching M_1 on V_1; pick one of the newly matched vertices (and the unmatched vertex, if any) and form V_2.
5. Continue the same as in *step* 4 until $|V_t| = 1$ for some t.
6. Return the set of edges T = $\cup_{j=0}^{t-1} M_j$.

Let's illustrate this algorithm with a simple example:



We have vertices $V = \{A, B, C, D, E\}$ and edge weights as follows:

- $d(A, B) = 1$, $d(A, C) = 4$, $d(A, D) = 3$, $d(A, E) = 6$
- $d(B, C) = 2$, $d(B, D) = 5$, $d(B, E) = 8$
- $d(C, D) = 3$, $d(C, E) = 7$
- $d(D, E) = 4$

Matching Based Augmentation: Example

→ Iteration 1:

- M_0 = {(A,B), (C,D)}
- V_1 = {A,D,E}

→ Iteration 1:

- M_0 = {(A,B), (C,D)}
- V_1 = {A,D,E}

→ Iteration 2:

- M_1 = {(A,D)}
- V_2 = {A,E}

Matching Based Augmentation: Example

→ Iteration 1:

- M_0 = {(A,B), (C,D)}
- V_1 = {A,D,E}

→ Iteration 2:

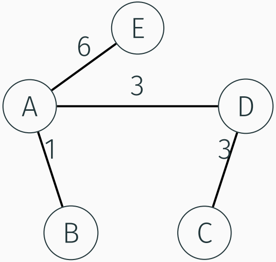
- M_1 = {(A,D)}
- V_2 = {A,E}

→ Iteration 3:

- M_2 = {(A,E)}
- V_3 = {A}

And the algorithm terminates here. The set of edges we got for the MST is {(A, B), (C, D), (A, D), (A, E)}. The cost of MST is = 13.

The MST formed is as follows:



The cost of MST given by our algorithm $= 13$.

Matching Based Augmentation: Theorem

Theorem

The above algorithm returns a tree of cost at most $O(\log n)$ times that of the MST.

The claim is that for each iteration i, all nodes in $V \setminus V_i$ are connected to at least one node in V_i using the edges in the matchings $\bigcup_{j<i} M_j$. This is proven inductively:

- **Base Case:** For $i = 0$, $V_0 = V$, and no nodes are removed yet. Hence, all nodes are connected to themselves in V_0.
- **Inductive Step:** Assume the claim is true for iteration $i - 1$. In iteration i, nodes in $V_i \setminus V_{i-1}$ are newly selected nodes, taken from the previously matched pairs. These nodes are dropped because they were already connected to other nodes using the matching M_{i-1}. Therefore, the nodes in $V_i \setminus V_{i-1}$ are still connected to V_i, confirming the inductive step.

- **Termination:** When the process stops (when $|V_t| = 1$), all other nodes are connected to this one remaining node. As the algorithm connects all nodes and adds edges between nodes, the resulting set of edges forms a spanning tree.
- Since the final set of edges T consists of exactly $n - 1$ edges (one less than the total number of nodes), it is a **spanning tree**.

Matching Based Augmentation: Proof of Cost of MST

- As we find a near-perfect matching take a node from each edge of the matching and add it to the set V_i. Hence, at each step, we reduce the size of the vertices to *at most half* and the algorithm stops when $|V_i| = 1$. Hence, it takes at most $\mathcal{O}(\log n)$ rounds.

Matching Based Augmentation: Proof of Cost of MST

- As we find a near-perfect matching take a node from each edge of the matching and add it to the set V_i. Hence, at each step, we reduce the size of the vertices to *at most half* and the algorithm stops when $|V_i| = 1$. Hence, it takes at most $\mathcal{O}(\log n)$ rounds.
- Now, to prove our theorem, we need to show that the cost of each matching is at most $d(T^*)$, i.e. the cost of the optimal MST T^*.

- As we find a near-perfect matching take a node from each edge of the matching and add it to the set V_i. Hence, at each step, we reduce the size of the vertices to *at most half* and the algorithm stops when $|V_i| = 1$. Hence, it takes at most $\mathcal{O}(\log n)$ rounds.
- Now, to prove our theorem, we need to show that the cost of each matching is at most $d(T^*)$, i.e. the cost of the optimal MST T^*. Now, the Euler Tour C of the MST T^* has cost at most $2d(T^*)$ since the Euler Tour might take an edge multiple times to travel all the nodes in the graph. To make the argument simpler, assume there are '$2k$' nodes in V_i. If we rename these nodes to be $\{x_0, x_1, \ldots, x_{2k-1}, x_{2k} = x_0\}$ in the order we encounter them as we go around the tour, then by the triangle inequality, $\sum_{i=0}^{2k-1} d\left(x_i, x_{i+1}\right) \leq 2d\left(T^*\right)$. Now, we partition the Euler tour C into k pairs of nodes, either $\{x_{2i}, x_{2i+1}\}$ or $\{x_{2i+1}, x_{2i+2}\}$, where $0 \leq i < k - 1$. So, one of the sums is at most half the cost of the Euler tour, i.e., $d\left(T^*\right)$. And this is a valid matching of the nodes in V_i, therefore the cost of each matching M_i is at most $d(T^*)$, the cost of the MST T^*.

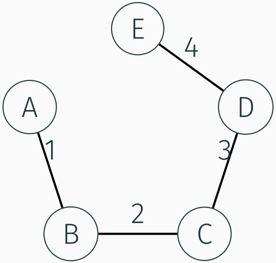
thus

$$M_i \le d(T^*);$$

$$\sum M_i \le \mathcal{O}(\log n) d(T^*)$$

.

The optimal MST formed will be as follows:



The cost of optimal MST formed = 10.
And, (log(n))*10 = (log(5))*10 = 23.
and our cost = 13.

THANK YOU!

ANY QUESTIONS?