Approximation Algorithms for Network Design

Dhruv Shah[1], Vatsal Shah[2], Pranav Patel[3], Vraj Thakkar[4], Abhinav Agrawal[5], and Kanishk Dad[6]

[1]202103017@daiict.ac.in
[2]202103022@daiict.ac.in
[3]202103040@daiict.ac.in
[4]202103052@daiict.ac.in
[5]202101040@daiict.ac.in
[6]202103005@daiict.ac.in

April 21, 2024

1 Introduction

In this term paper, we discuss the survey article titled ”Approximation Algorithms for Network Design: A Survey” by Anupam Gupta and Jochen Könemann [2].
The network design problems have many practical applications ranging from the design process of telecommunication and traffic networks to VLSI chip design. Many such network design problems are NP-hard, and intractable. Therefore, in this term paper, we discuss some of the *approximation algorithms* given by [2] as a possible way to navigate this deadlock.
In a typical instance of a network design problem, given any graph (directed or undirected) with non-negative edge weights, our task would be to find a minimum-cost sub-graph that satisfies some design criteria. Therefore, we will discuss a particular problem known as the **Minimum Spanning Tree** problem and formulate some Approximation Algorithm techniques using this problem.

2 Minimum Spanning Tree: Kruskal's Algorithm

A Minimum Spanning Tree (MST) is a fundamental concept in graph theory and computer science, particularly in the realm of network design and optimization.

In a graph, which consists of vertices (nodes) and edges (connections between nodes), a spanning tree is a subgraph that connects all vertices together without creating any cycles. A minimum spanning tree is a spanning tree with the minimum possible sum of edge weights.

The key properties of a minimum spanning tree are:

1. Spanning: It covers all the vertices of the original graph.

2. Tree: It does not contain any cycles.

3. Minimum weight: The sum of the weights of its edges is minimized.

Finding a minimum spanning tree can be achieved using various algorithms, such as Prim's algorithm, Kruskal's algorithm, or Borůvka's algorithm. These algorithms differ in their approach and efficiency, but they all guarantee to find the minimum spanning tree for a given graph.

Minimum spanning trees are essential in network design, such as in telecommunications, transportation, and computer networks, where the goal is to connect nodes while minimizing the total cost or distance. They also have applications in clustering, image processing, and approximation algorithms.

We first state and prove Kruskal's Algorithm for computing the minimum spanning tree.

Theorem 2.1. *After running Kruskal's algorithm on a connected weighted graph G, its output T is a minimum weight spanning tree.*

Proof. **First**, T is a spanning tree. This is because:

- **T is a forest**. No cycles are ever created.

Algorithm 1 Kruskal's Algorithm for MST

```
Require: $G$: weighted connected undirected graph with $n$ vertices
Ensure: $T$: minimum spanning tree of $G$
1: $T :=$ empty graph
2: for $i := 1$ to $n - 1$ do
3:     $e :=$ any edge in $G$ with the smallest weight that does not form a cycle when added to $T$
4:     $T := T$ with $e$ added
5: end for
6: return $T$
```

- **T is spanning**. Suppose that there is a vertex v that is not incident with the edges of T. Then the incident edges of v must have been considered in the algorithm at some step. The first edge (in edge order) would have been included because it could not have created a cycle, which contradicts the definition of T.

- **T is connected**. Suppose that T is not connected. Then T has two or more connected components. Since G is connected, then these components must be connected by some edges in G, not in T. The first of these edges (in edge order) would have been included in T because it could not have created a cycle, which contradicts the definition of T.

Second, T is a spanning tree of minimum weight. We will prove this using induction. Let T^* be a minimum-weight spanning tree.

- If T = T^*, then T is a minimum weight spanning tree. If T $\neq T^*$, then there exists an edge e $\in T^*$ of minimum weight that is not in T. Further, T $\cup$ e contains a cycle C such that:
  a. Every edge in C has a weight less than wt(e). (This follows from the algorithm)
  b. There is some edge f in C that i s not in T^*. (Because T^* does not contain the cycle C.)

- Consider the tree $T_2 = T \setminus \{e\} \cup \{f\}$, T_2 is a spanning tree. T_2 has more edges in common with T^* than T did. And wt (T_2) $\geq$ wt (T). (We exchanged an edge for one that is no more expensive.)

We can redo the same process with T_2 to find a spanning tree T_3 with more edges in common with T^*. Then by induction, we can continue this process until we reach T^*, from which we see $wt(T) \leq wt(T_2) \leq wt(T_3) \leq ..... \leq wt(T^*)$ Since T^* is a minimum weight spanning tree, then these inequalities must be equalities and we conclude that T is a minimum weight spanning tree. $\square$

3 Primal-Dual Approaches

Now we will formulate the primal-dual interpretation of Kruskal's algorithm. For this, we first introduce a linear programming formulation for the minimum-cost spanning tree problem.

3.1 Primal form

For each edge $e \in E$, there is a variable x_e where $x_e \in \{0, 1\}$. (In LP relaxation $x_e \geq 0$)
Π: Set of all partitions of the set of all nodes V
rank $r(\pi)$: number of parts of π.
E_π : Set of edges whose both endpoints lie in different partitions.

$$\begin{array}{lll} \text{minimize} & \sum_{e\in E} c_e x_e & \\ \text{subject to} & \sum_{e\in E_\pi} x_e \geq r(\pi)-1, & \forall \pi \in \Pi, \\ & x_e \geq 0, & \end{array} \tag{1}$$

By the definition of Spanning Tree, there should be at least $r(\pi)-1$ edges, since all nodes should be connected, and we want to find a set of edges that cost us minimum to span a tree which will be our objective function value.

3.2 Dual form

Linear Programming of the dual of the above primal problem will have a variable y_π for each partition $\pi \in \Pi$, and objective function value will become constraints for each edge $e \in E$.

$$\begin{array}{lll} \text{maximize} & \sum_{\pi\in\Pi} (r(\pi)-1)y_\pi & \\ \text{subject to} & \sum_{\pi:e\in E_\pi} y_\pi \leq c_e, & \forall e \in E, \\ & y_\pi \geq 0, & \end{array} \tag{2}$$

4 Iterative Rounding

The feasible region of the problem (1) is identical to the formulation stated below, given by Chopra in [1]

$$\begin{array}{lll} \text{minimize} & \sum_{e\in E} c_e x_e & \\ \text{subject to} & x(E(V)) = n-1 & \\ & x(E(S)) \leq |S|-1, & S \subset V, \\ & x_e \geq 0, & \forall e \in E \end{array} \tag{3}$$

Here $|V| = n$, and $E(S)$ is the set of edges whose both endpoints are in S, $x(A) = \sum_{e\in A} x_e$ for $A \subset E$.

Lemma 4.1. *Let G be a connected graph with at least two vertices and let x^* be a basic solution of (3) and let $E^* = \{e \in E : x^*_e \geq 0\}$. There exists a node v such that v is exactly incident to one edge in E^*.*

The Lemma (4.1) says that according to the constraints of the LP (3) any incident edge of a leaf node should included in the solution, that is why there exists a node v such that $x^*_{uv} = 1$, such edges we will call as single support edges.

Theorem 4.2. *Given a connected graph $G = (V, E)$ and edge costs c_e for all $e \in E$, the Algorithm (2) correctly computes an MST of G.*

Proof. **Case 1** : If G has at most one vertex, the **Algorithm (2)** returns ϕ.
Case 2 : If G has more than one vertex, As G is connected by Lemma 1 of the above Algorithm (2) will identify a vertex u that is incident to a single support edge uv with $x^*_{uv} = 1$. As mentioned in the Algorithm (2) we obtain $G' = (V', E')$.

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 $F$.
```

Let x' be the projection of x^* onto the edge set of G', which means that $x' \in \mathbb{R}^{E'}$ and $x_e = x^*_e$ for all $e \in E'$

x' is feasible for (2) graph G'. Let F' be a tree computed by Algorithm (2) from iteration 2 and after. Inductively, the algorithm will return the spanning tree F' of cost at most the optimal value of (2) and hence $c(F) \leq \sum_{e \in E'} c_e x^*_e$. Finally, $F = F' \cup uv$ is a spanning tree of G and its cost is

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

Hence, the spanning tree returned by the algorithm has cost at most the cost of the LP solution x^*.

$\square$

5 Randomization

Randomization is often used in approximation algorithms in processes like randomized rounding of linear programming relaxations. In this section, we introduce randomized rounding for the minimum spanning tree problem, where we discuss how to apply randomized rounding to the linear programming relaxation of the minimum spanning tree problem.

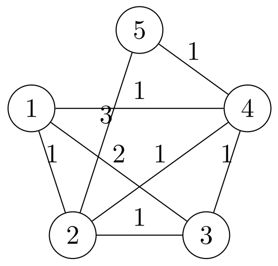
5.1 Randomized Rounding Algorithm

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

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

Let's illustrate this algorithm with a simple example. Consider the graph:
Vertices: $V = \{1, 2, 3, 4, 5\}$
Edges: $E = \{(1, 2, 1), (1, 3, 2), (1, 4, 1), (2, 3, 1), (2, 4, 1), (2, 5, 3), (3, 4, 1), (4, 5, 1)\}$



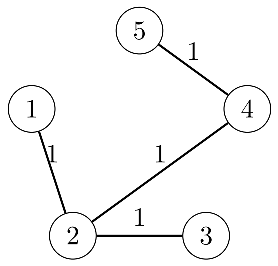
Now, let's go through the algorithm steps: **Iteration 1:**

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

Iteration 2:

- Repeat the edge selection process and toss the coins again.
- Example selection: $A_2 = A_1 \cup \{(1, 4), (2, 4)\}$
- Check connectivity: G_2 is connected.

Return Spanning Tree: G_2 is connected, so we return any spanning tree of G_2, for example:



This spanning tree covers all vertices of G_2.

Lemma 5.1. *Given any (multi)graph $H = (U, E)$ and a feasible solution $\{x_e\}_{e \in E}$ to (PSP) 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. An isolated vertex u is a vertex in the graph H that is not incident to any edge selected by the random process. Now the probability that a vertex u is isolated is given by the product of the probabilities that none of the edges incident to u are selected, this probability is:

$$\prod_{e\in\delta(\{u\})} (1 - x_e)$$

Using the bound $(1 - x) \leq e^{-x}$, and the fact that $\sum_{e\in\delta(\{u\})} x_e \geq 1$, we get:

$$\prod_{e\in\delta(\{u\})} (1 - x_e) \leq e^{-\sum_{e\in\delta(\{u\})} x_e} \leq \frac{1}{e}$$

By linearity of expectation, the expected number of isolated vertices is at most $|U|/e$. Applying Markov's inequality to X = number of isolated vertices and $t = 2|U|/e$, we have:

$$\Pr(X \leq 2|U|/e) \geq \frac{|U|/e}{2|U|/e} = \frac{1}{2}$$

.

Markov's inequality implies that we have at most $2|U|/e$ isolated vertices with a probability of at least 1/2. Since each non-isolated vertex must be part of a component of size at least 2, and there are at most $2|U|/e$ isolated vertices, the total number of components (Number of isolated nodes + half times the number of remaining nodes) is at most $|U|(2/e + 1/2(1 - 2/e)) < 0.9|U|$ with probability at least $\frac{1}{2}$.

Therefore, the lemma concludes that the number of components in the graph H is at most $0.9|U|$ with probability at least $\frac{1}{2}$. In summary, the lemma provides a probabilistic bound on the number of components in the graph H based on the probabilities x_e assigned to edges. This bound holds with high probability, ensuring that the graph has a relatively small number of components. □

Theorem 5.2. *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 start by establishing a bound on the number of rounds required by the MST-Rand-Round algorithm. This is because each round of the MST-Rand-Round algorithm adds edges to the graph, eventually forming a spanning tree. The cost of this spanning tree is determined by the sum of the costs of the edges selected during the rounds. 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 $G_L = (V, \cup_{j\leq i} A_j)$ 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 $2L$. 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 $2L = 120$. Since the expected number of rounds is $2L$, the algorithm iterates through this many rounds to construct the spanning tree. Each round contributes to the cost of the spanning tree, and because the expected number of rounds is $O(\log n)$, the total cost of the tree will be $O(\log n)$ times the cost of Z^*.

Next, we focus on proving that the graph G_L constructed after L rounds of the algorithm is connected with high probability. We prove that the graph G_L is connected with probability at least 1/2. For that, we define C_i as 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 $C_i \leq 0.9 \cdot C_{i-1}$.

Now the probability of i being successful is at least 1/2 **regardless of all random choices made in previous rounds.** This ensures that the algorithm makes progress towards constructing a connected graph. Let us show how this is true, there are two cases: If G_{i-1} is connected, then index 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}, we have that $\sum_{e\in\delta(S)} x_e \geq 1$ holds for H_{i-1}. By Lemma 22, 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.

Now let us bound the probability that the graph G_L is not connected after L rounds of the algorithm. Given the above claims, 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 $\frac{\log n}{\log 0.9}$ 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 $\frac{\log n}{\log 0.9} < 10 \log n$ heads (and hence the probability that G_L is not connected) is at most $\frac{1}{2}$, which completes the proof. This indicates that the algorithm has a high probability of maintaining graph connectivity throughout the L rounds.

□

6 Matching Based Augmentation

Matching-based augmentation is an approach used to approximate 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 particularly useful when the graph is represented as a metric space.

6.1 Algorithm for Matching Based Augmentation

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. First, 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.
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 $T = \cup_{j=0}^{t-1} M_j$.

Theorem 6.1. *The above algorithm returns a tree of cost at most $\mathcal{O}(\log n)$ times that of the MST.*

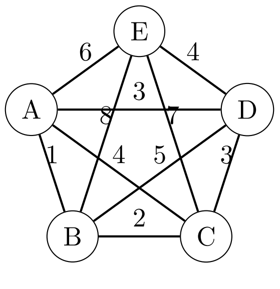
Proof. First, we will show that the set of edges given by the above algorithm gives a minimum spanning tree. We can prove this inductively by saying that all nodes in $V \setminus V_i$ are connected to some node in V_i using the edges obtained by $\cup_{j<i} M_i$.

Now, when $i = 0$, i.e., when the algorithm starts, any two nodes in V are connected, either directly or indirectly using the near-cost perfect matching M_0. For any $i > 0$, the nodes in $V \setminus V_i$ are connected to some nodes in set V_i using the edges from the near-perfect matching in M_{i-1}, since such a matching leaves at most one node unmatched. Moreover, any node in $V_i \setminus V_{i-1}$ is dropped because it is connected to another node using the matching in M_{i-1}.

Now, when we stop we have a single node, and all other nodes are connected to this node, using the edges in $\cup_{j<i} M_i$, which are precisely $n-1$, where n is the number of nodes. Hence, we obtain a spanning tree.

In our algorithm, in each iteration, we find a near-perfect matching and 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, $\sum_{i=0}^{2k-1} d(x_i, x_{i+1}) \leq 2d(T^*)$. 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, $d(T^*)$. 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^*. □

Let's illustrate this algorithm with a simple example. Consider the following graph:



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

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

- $d(D, E) = 4$

→ **Iteration 1**:

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

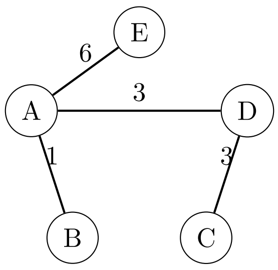
→ **Iteration 2**:

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

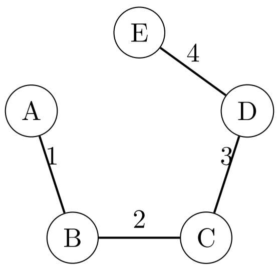
→ **Iteration 3**:

- M_1 = {(A,E)}
- V_2 = {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 optimal MST formed will be as follows:



The cost of optimal MST formed = 10.
And, 10*(log(n)) = 10*(log(5)) = 23.
The cost of MST given by our algorithm is 13, which is less than 23. This shows that our algorithm gives $\mathcal{O}(\log n)$ approximation.

7 Contribution

Pranav Patel and Vraj Thakkar - Studied the Primal-Dual Formulation of Kruskal's algorithm and Iterative rounding. (Section 3, 4)
Dhruv Shah and Vatsal Shah - Studied the Primal formulation of the Kruskal's Algorithm and the Randomization Analysis (Section 5)
Abhinav Agrawal and Kanishk Dad - Studied the Matching Based Augmentation Strategy to Approximate the MST (Section 6)

References

[1] Sunil Chopra. On the spanning tree polyhedron. *Operations Research Letters*, 8(1):25–29, 1989.

[2] Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.