Steiner Tree

Md Nafiu Rahman 1905077
Kazi Reyazul Hasan 1905082
Asad Bin Shahid 1905087
Mubasshira Musarrat 1905088
Shahriar Raj 1905105

Problem Definition

1905105

Steiner Tree Problem

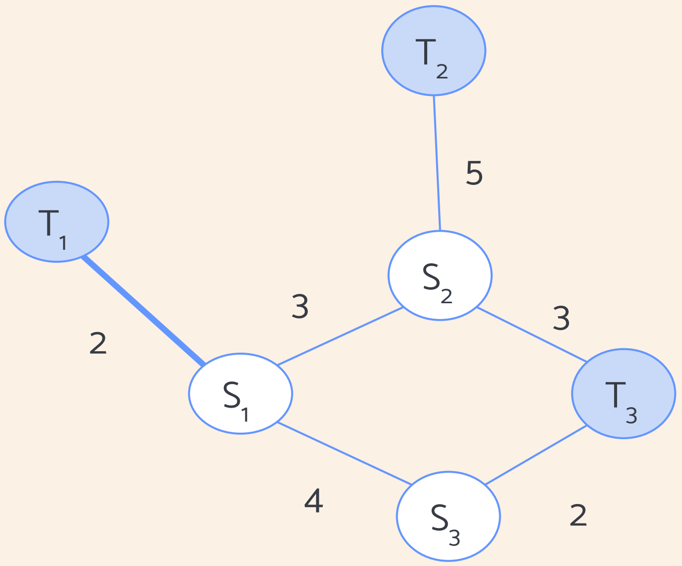
Instance

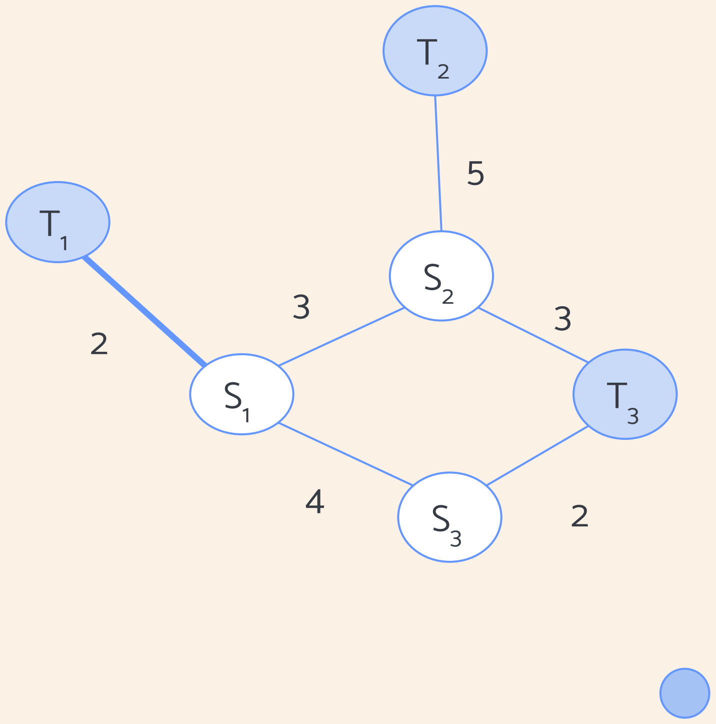
- An **undirected graph** $G = (V,E)$
- A subset of vertices $R \subseteq V$ called **terminal nodes**
- A **number** $k \in \mathbb{N}$

Question:

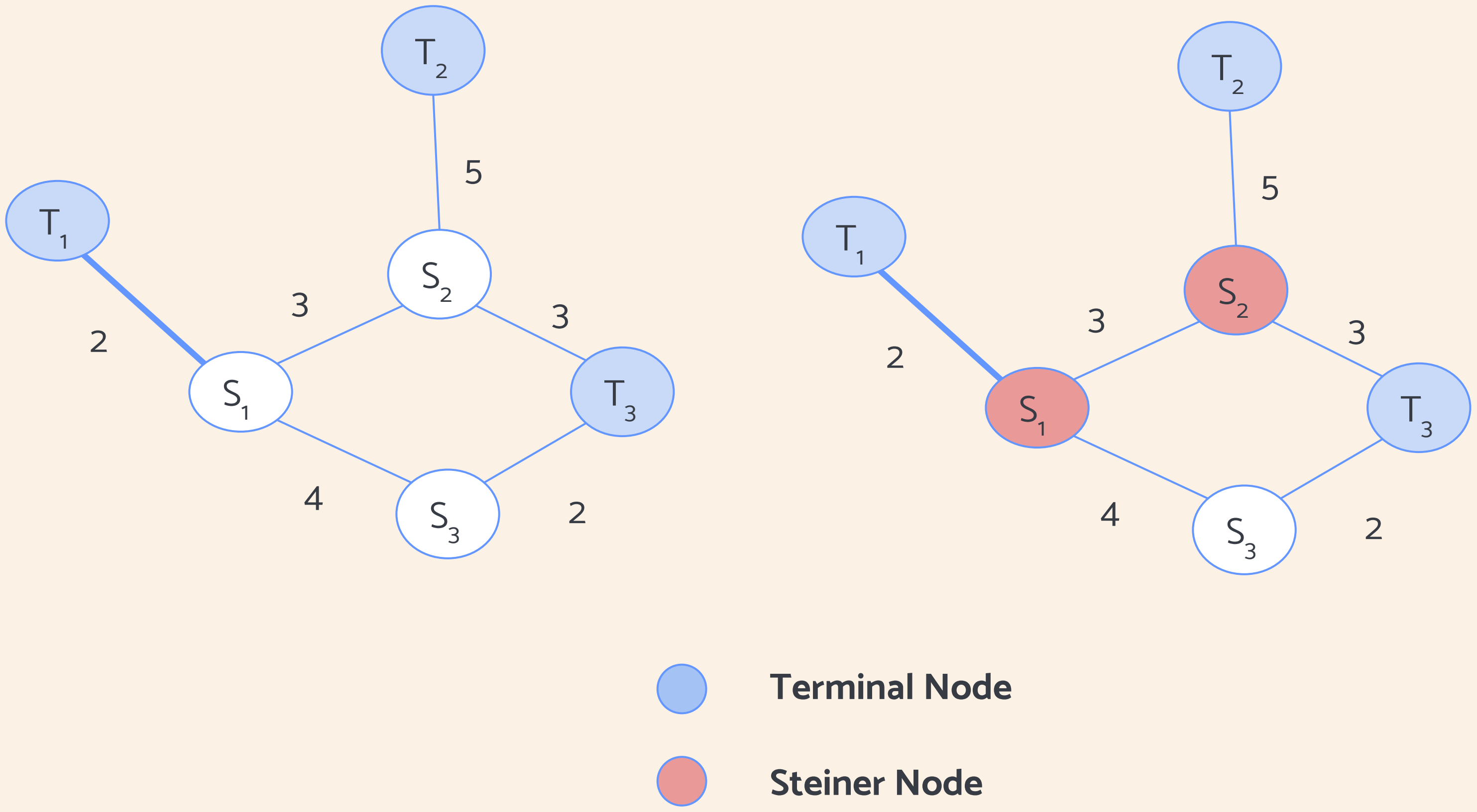
Does there exist a **subtree** of G that:

- Includes all the vertices of R (i.e., a spanning tree of the terminal nodes)
- Contains **at most k edges** (for unweighted graphs), or has a **total cost** of at most k (for weighted graphs)?





Terminal Node


T1
T2
T3
S1
S2
S3
2
3
5
3
4
2
T1
T2
T3
S1
S2
S3
2
3
5
3
4
2
Terminal Node
Steiner Node

Differences Between MST & ST

- Definition : Covers All vertices I Covers The Terminal Vertices
- Nodes: All Nodes I Terminal Nodes + Steiner Nodes
- Complexity: Polynomial I Not Polynomial
- Use Case: Power Grid Connection I Telecommunication

Different Types of Steiner Trees

- **Classical Steiner Tree** (Example: Telecommunication)
- **Rectilinear Steiner Tree** (Example: VLSI design)
- **Group Steiner Tree** (Example: Content Delivery Networks)
- **Degree Steiner Tree** (Example: Switch, Transformer)
- **K-restricted Steiner Tree** (Example: Network Design)
- **Prize-Collecting Steiner Tree** (Example: Marketing Campaign)

Complexity

The **Steiner Tree Problem** is:

NP-Complete for **unweighted graphs and NP-Hard** for **weighted graphs**

Different Exact Algorithms

Algorithm Name	Time Complexity	Notes
Dreyfus-Wagner	$O(3^k \cdot n + 2^k \cdot n^2 + n(n \log n + m))$	Classical DP algorithm.
Ericsson's Algorithm	$O(3^k \cdot n + 2^k \cdot (n \log n + m))$	Branch & Bound algorithm that combines the concept of minimum spanning tree.

Improvement to exact algorithm

$O^*(2.684^k)$	The optimal Steiner tree T can be divided into three subtrees T1, T2 and T3 in such a way that each subtree Ti represents a minimum Steiner tree within a contracted graph that contains fewer than k/2 terminal nodes. This approach uses a decomposition strategy that simplifies the problem by reducing the number of terminals in each contracted graph, allowing for more manageable subproblems. This idea was explored by Fuchs, Kern, and Wang in their 2007 work in *Mathematical Methods of Operations Research*.
$O(c^k)$ for any $c \geq 2$	It builds an optimal solution by assembling parts that each contain only a limited number of terminal nodes [Molle, Richter, and Rossmanith, STACS 2006].
$O(2^k n^2 + nm)$	It constructs an optimal solution through techniques like subset convolution and Möbius inversion [Bjorklund, Husfeldt, Kaski, Koivisto, STOC 2007].
$O(6^k n^{O(\log(k))})$	First polynomial space algorithm [Fomin, Grandoni, Kratsch ESA 2008]
$O(2^k n^{O(1)})$	Polynomial space algorithm based on the inclusion-exclusion principle [Nederlof, ICALP 2009]

Different Approximate Algorithms

Algorithm Name	Approximation Ratio	Worst Time Complexity
Minimum Spanning Tree	2(1- 1/L), where L is the no of leaves in the optimal tree	$O(\|S\|\|V\|^2)$;spans S terminals
Linear Programming Based Approximation Algorithm With Iterative Randomized Rounding	ln4+ϵ < 1.39	$O(\|S\|\|V\|^3)$;spans S terminals
Primal-Dual Approximation Algorithm	2(1- 1/L), where L is the no of leaves in the optimal tree	$O(\|V\|\log\|V\| + \|V\|^2)$

Algorithm Name	Approximation Ratio	Worst Time Complexity
A Faster Approximation Algorithm	2(1- 1/L), where L is the no of leaves in the optimal tree	$O(\|V\|\log\|V\| + \|E\|)$
Zelikovsky's Approximation Algorithm	11/6	$O(\|E\| + \|V\|^{\log 4})$
Relative Greedy Algorithm	1.694	$O(\|E\|\log\|V\|)$
Loss Contraction Algorithm	1.55	$O(\|E\|\log\|V\|)$

Heuristics

Algorithm Name	Approximation Ratio	Time Complexity	How it works?
Minimum Path Heuristic (Takahashi and Matsuyama)	2−2/k, where k is the number of terminals	$O(k \cdot n^2)$	iteratively adding the shortest path from the existing tree to the closest unconnected terminal
Contraction Heuristic (Plesnik)	Bounded above by 2 and can be 2	$O(n^3)$	recursively reduces the graph by forming neighborhoods around Steiner vertices, contracting each neighborhood class into a single vertex, and then constructing a Steiner tree on the reduced graph

Metaheuristics

Name	Time Complexity	How it works?
Genetic Algorithm	$O(N \cdot \lvert V \rvert^2)$, where N is the population size	evolves a population of Steiner trees through selection, crossover, and mutation, optimizing the tree weight iteratively
Simulated Annealing	$O(\lvert V \rvert^2 \cdot T)$, where T is the number of iterations	gradually reduces the probability of accepting higher-cost solutions to escape local minima

Metaheuristics

Name	Time Complexity	How it works?
Variable Neighborhood Search	Generally $O(n^k)$, where k is the number of neighborhoods explored	changes neighborhoods (solution structures) to escape local optima
Tabu Search	Generally $O(n^2)$ or more, depending on neighborhood structure and memory management	uses memory structures to store recently visited solutions (tabu list) to prevent cycling back to them

Metaheuristics

Name	Time Complexity	How it works?
Ant Colony Optimization (ACO)	Highly variable, generally $O(n \cdot \lvert V \rvert^2)$, where n is the number of ants, affected by the number of iterations and pheromone update strategy.	simulates the behavior of ants, where virtual "ants" explore paths and deposit pheromones on those that give shorter, lower-cost routes. Over time, these pheromones reinforce paths that form a good solution, iteratively approximating the minimal steiner tree

Exact Algorithm

1905077

Dreyfus-Wagner Algorithm

Solves the Steiner Tree Problem by calculating the minimum-cost connections for all subsets of terminal nodes using dynamic programming.

The fundamental idea

- Compute the weight of a minimum Steiner tree for a given terminal set by considering the weights of the minimum Steiner trees of **all proper subsets** of this set.
- Starting the process with two-element subsets (where the Steiner tree can be determined by shortest path computations) one finally ends up with the k-element terminal set S.

Some notations...

Notations:

- C(u,S): The minimum cost to connect a vertex u to all the vertices in the set S, where $u \notin S$.
- S: A subset of terminal vertices.
- d(u,v): The cost (weight) of the direct edge between vertices u and v.
- T: The set of terminal vertices.

Recursive Formula

$$C(u, S) = \min\left(\min_{v \in S}\left[d(u,v) + C(v, S \setminus \{v\}) \right], \right.$$

$$\left. \min_{S1=S\setminus\{u\}, S2=S\setminus S1, v\in V}\left[C(v, S1) + C(v, S2) + d(a,v)\right] \right)$$

Base Case

If S is a singleton set, i.e., S={v} then:

C(u,{v})=d(u,v)

d(u,v) is the Shortest Distance between u and v (Can be computed using Dijkstra's algorithm)

First Case

$$C(u, S) = \min_{v \in S}[\, d(u,v) + C(v, S \setminus \{v\})\,]$$

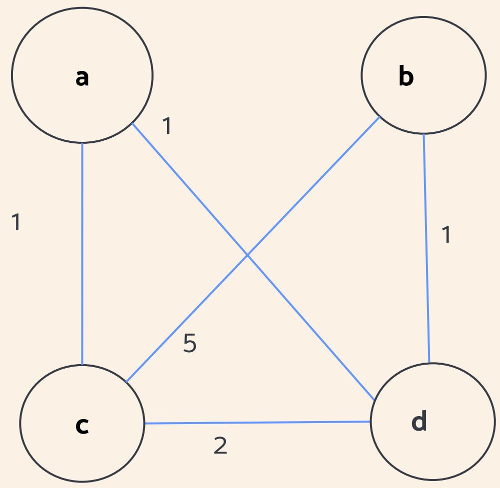
Choose an intermediate vertex $v \in S$ and split S into $\{v\}$ and $S \setminus \{v\}$.

Second Case

$$C(u, S) = \min_{S1=S\setminus\{u\}, S2=S\setminus S1, v\in V} [C(v, S1) + C(v, S2) + d(u,v)]$$

Split S into two non-empty disjoint subsets S1 and S2, compute the cost of connecting u to each subset, and then combine the results

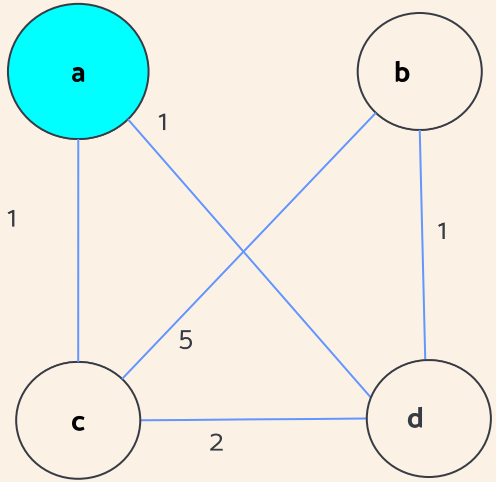
A small example



Terminal nodes: a,b,c
Normal nodes: d

The goal is to compute **C({a,b,c})** the minimum weight of the Steiner tree connecting a, b and c. The idea is to compute the weight of a minimum Steiner tree for the terminal set by considering the weights of the minimum Steiner trees of all proper subsets of this set.

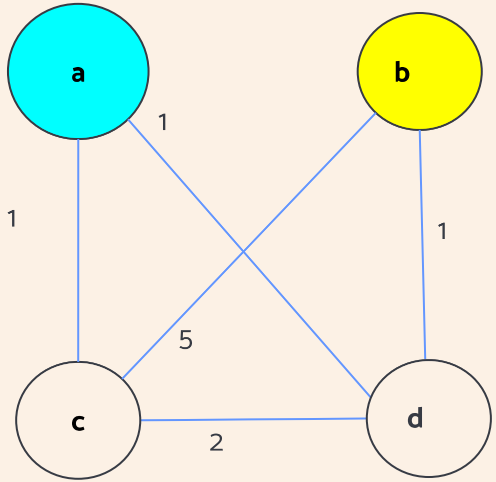
A small example



Terminal nodes: a,b,c
Normal nodes: d

We start by defining **C(a , { a }), C(b, { b }) , C(c, { c }).** Since { a } , { b } , { c } are individual nodes, we define **C(a , { a }) = 0**

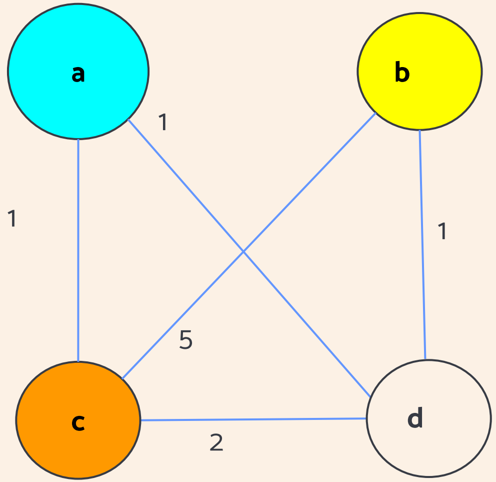
A small example



Terminal nodes: a,b,c
Normal nodes: d

We start by defining **C(a , { a }), C(b, { b }) , C(c, { c }).** Since { a } , { b } , { c } are individual nodes, we define **C(a , { a }) = 0 , C(b , { b }) = 0**

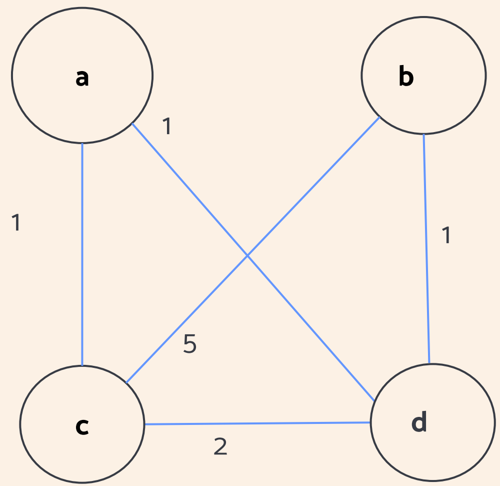
A small example



Terminal nodes: a,b,c
Normal nodes: d

We start by defining **C(a , { a }), C(b, { b }) , C(c, { c }).** Since { a } , { b } , { c } are individual nodes, we define **C(a , { a }) = 0 , C(b , { b }) = 0, C(c , { c }) = 0**

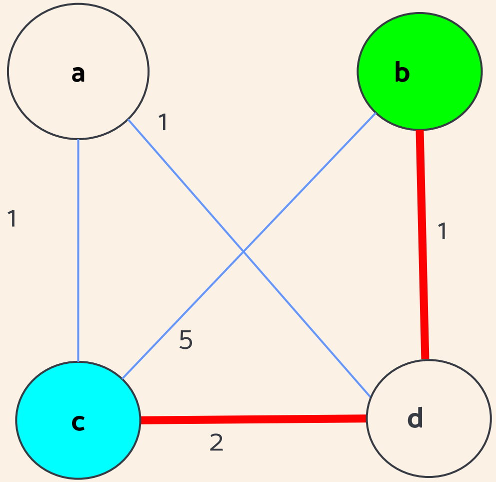
A small example



Terminal nodes: a,b,c
Normal nodes: d

We start the process with two-element subsets (where the Steiner tree can be determined by shortest path computations, using Dijkstra's algorithm, for example) and will finally end up with the k-element(3 here) terminal set S.

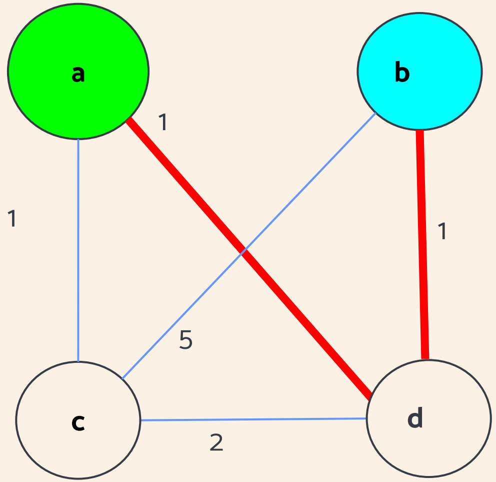
A small example



$C(b, \{ c \}) = C(c, \{ b \}) = 3$

Terminal nodes: a,b,c
Normal nodes: d

A small example

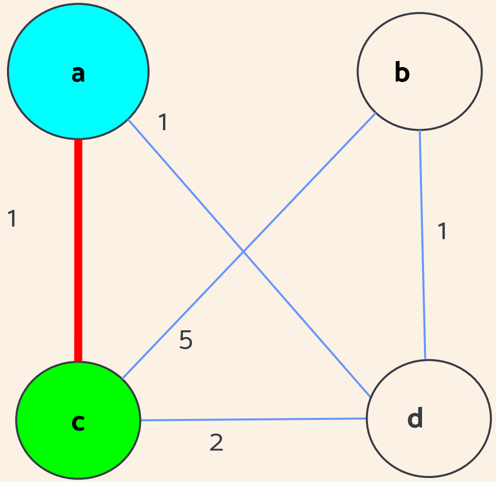


$C(b, \{ c \}) = C(c, \{ b \}) = 3$

$C(a, \{ b \}) = C(b, \{ a \}) = 2$

Terminal nodes: a,b,c
Normal nodes: d

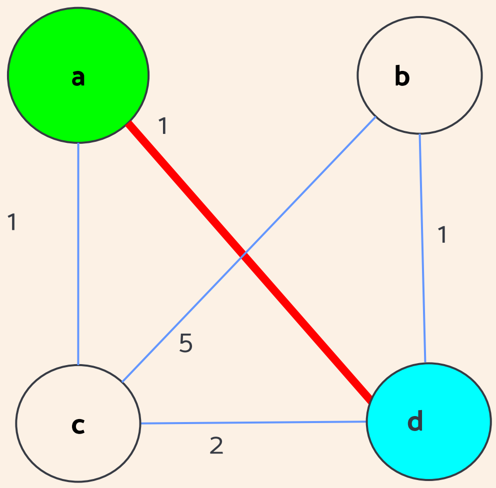
A small example



$C(b, \{ c \}) = C(c, \{ b \}) = 3$
$C(a, \{ b \}) = C(b, \{ a \}) = 2$
$C(a, \{ c \}) = C(c, \{ a \}) = 1$

Terminal nodes: a,b,c
Normal nodes: d

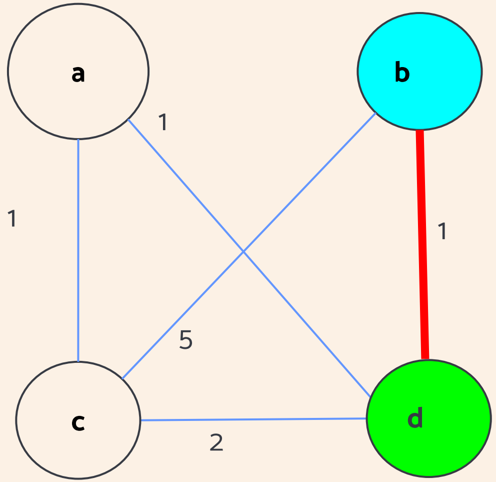
A small example



$C(b, \{ c \}) = C(c, \{ b \}) = 3$
$C(a, \{ b \}) = C(b, \{ a \}) = 2$
$C(a, \{ c \}) = C(c, \{ a \}) = 1$
$C(a, \{ d \}) = C(d, \{ a \}) = 1$

Terminal nodes: a,b,c
Normal nodes: d

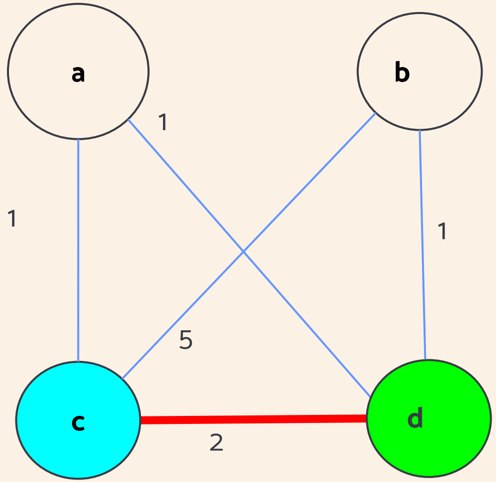
A small example



$C(b, \{ c \}) = C(c, \{ b \}) = 3$
$C(a, \{ b \}) = C(b, \{ a \}) = 2$
$C(a, \{ c \}) = C(c, \{ a \}) = 1$
$C(a, \{ d \}) = C(d, \{ a \}) = 1$
$C(b, \{ d \}) = C(d, \{ b \}) = 1$

Terminal nodes: a,b,c
Normal nodes: d

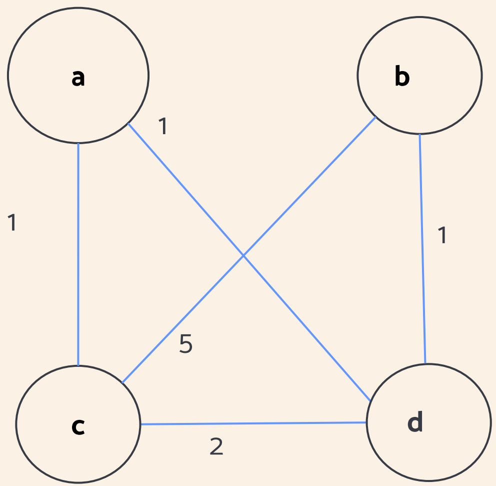
A small example



$C(b, \{ c \}) = C(c, \{ b \}) = 3$
$C(a, \{ b \}) = C(b, \{ a \}) = 2$
$C(a, \{ c \}) = C(c, \{ a \}) = 1$
$C(a, \{ d \}) = C(d, \{ a \}) = 1$
$C(b, \{ d \}) = C(d, \{ b \}) = 1$
$C(c, \{ d \}) = C(d, \{ c \}) = 2$

Terminal nodes: a,b,c
Normal nodes: d

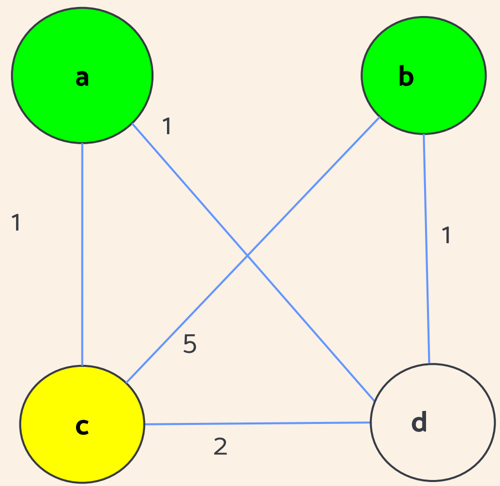
A small example



Terminal nodes: a,b,c
Normal nodes: d

The goal is to compute **C({a,b,c})** the minimum weight of the Steiner tree connecting a, b and c. The idea is to compute the weight of a minimum Steiner tree for the terminal set by considering the weights of the minimum Steiner trees of all proper subsets of this set.

A small example

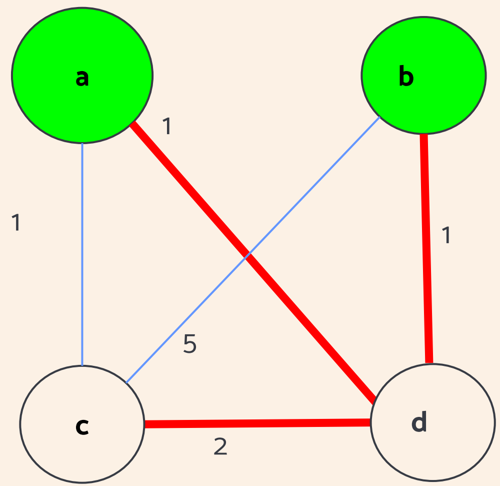


Terminal nodes: a,b,c
Normal nodes: d

$$C(a, \{b, c\}) = \min\left(\min_{v \in \{b,c\}} [\, d(a,v) + C(v, \{b, c\} \setminus \{v\}) \,],\right.$$

$$\left. \min_{S1 = S \setminus \{u\}, S2 = S \setminus S1, v \in V} [\, C(v, S1) + C(v, S2) + d(a,v) \,] \right)$$

A small example

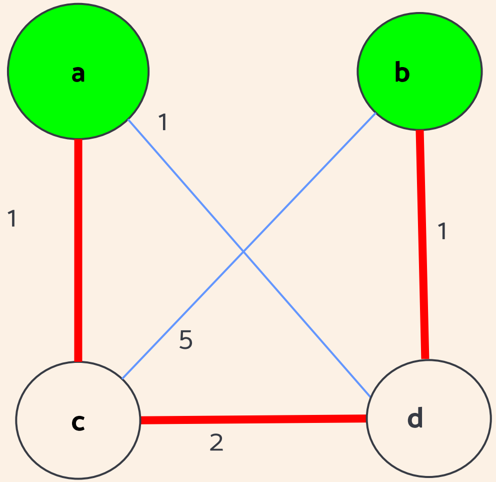


$\min_{v \in \{b,c\}} [\, d(u,v) + C(\, v \,, \{\, b, c \,\} \setminus \{\, v \,\} \,)$

$d(a,b) + C(b,c) = 2 + 3 = 5$

Terminal nodes: a,b,c
Normal nodes: d

A small example

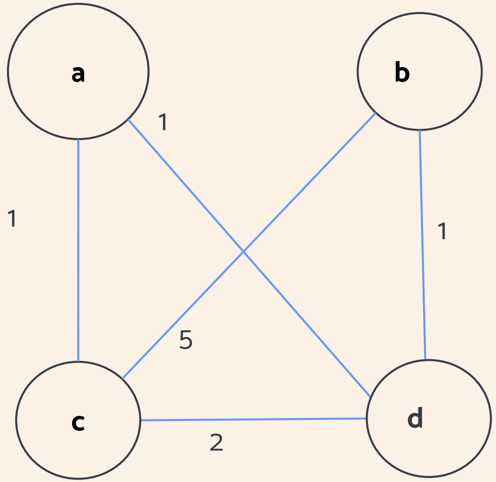


Terminal nodes: a,b,c
Normal nodes: d

$\min_{v \in \{b,c\}} [\, d(u,v) + C(\, v \,, \{ b, c \} \setminus \{ v \} \,)$

$d(a,b) + C(b,c) = 2 + 3 = 5$
$d(a,c) + C(b,c) = 1 + 3 = 4$

A small example



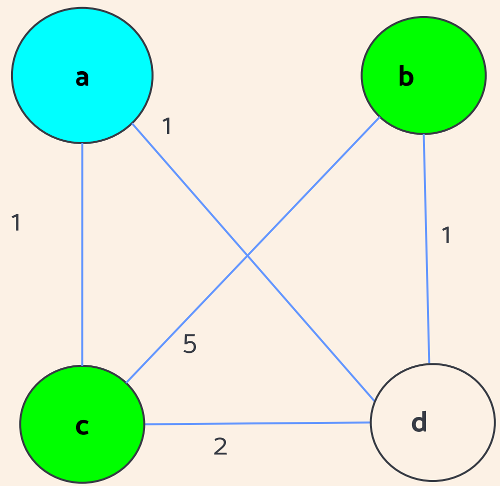
Terminal nodes: a,b,c
Normal nodes: d

$\min_{v \in \{b,c\}} [\, d(u,v) + C(\, v \,, \{\, b, c \,\} \setminus \{\, v \,\} \,)$

$d(a,b) + C(b,c) = 2 + 3 = 5$
$d(a,c) + C(b,c) = 1 + 3 = 4$

$\min(4, 5) = 4$

A small example

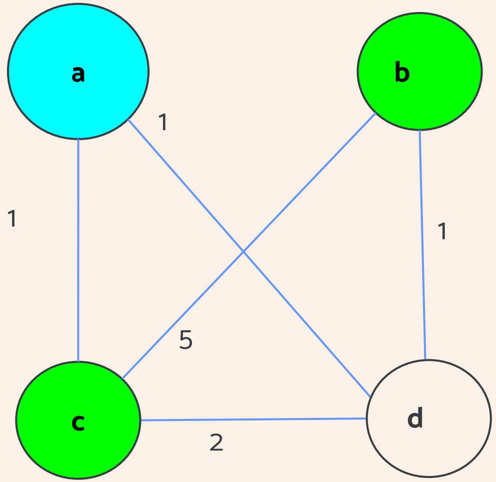


Terminal nodes: a,b,c
Normal nodes: d

$$\min_{S1=S\setminus\{u\},S2=S\setminus S1,v\in V} [\, C(v, S1) + C(v, S2) + d(a,v)])$$

$$C(a, c) + C(a, b) + C(a, a) = 3$$

A small example



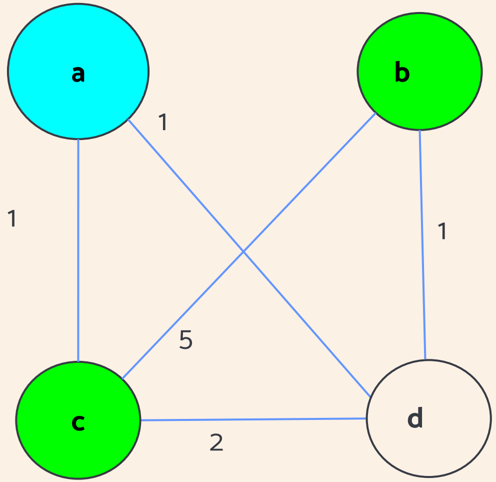
$$\min_{S1=S\setminus\{\,u\,\},S2=S\setminus S1,v\in V} [\, C(\,v\,,S1\,) + C(\,v\,,S2\,) + d(a,v)]\,)$$

C(a , c) + C(a , b) + C(a, a)= 3

C(d , c) + C(d , b) + C(d, a)= 3+1 = 4

Terminal nodes: a,b,c
Normal nodes: d

A small example

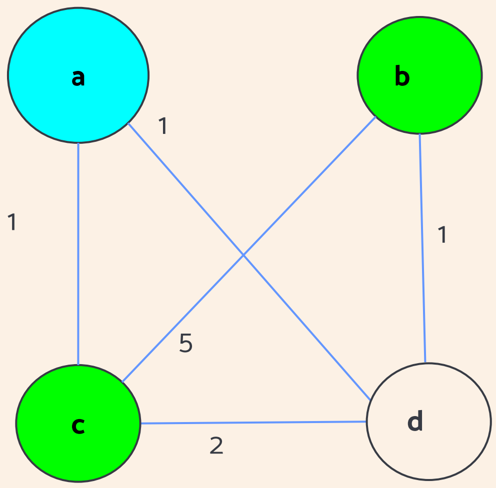


Terminal nodes: a,b,c
Normal nodes: d

$$\min_{S1=S\setminus\{u\},S2=S\setminus S1,v\in V} [\, C(v, S1) + C(v, S2) + d(a,v)] \,)$$

C(a , c) + C(a , b) + C(a, a)= 3
C(d , c) + C(d , b) + C(d, a)= 3+1 = 4
So, minimum length = (4, 3) = 3

A small example



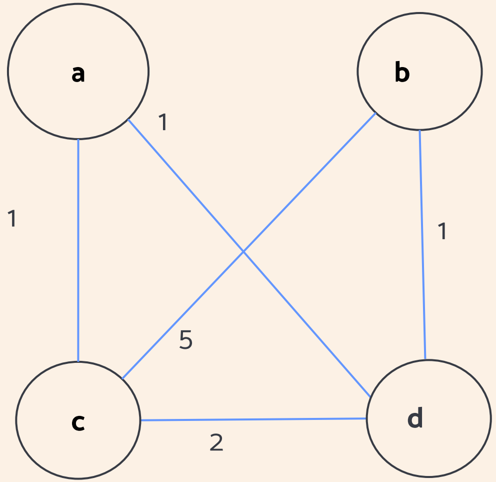
Terminal nodes: a,b,c
Normal nodes: d

$$\min_{S1=S\setminus\{ u \},S2=S\setminus S1,v\in V} [C(v , S1) + C(v , S2) + d(a,v)])$$

C(a , c) + C(a , b) + C(a, a)= 3
C(d , c) + C(d , b) + C(d, a)= 3+1 = 4
So, minimum length = (4, 3) = 3

Therefore overall minimum is min(3, 4) = 3

A small example

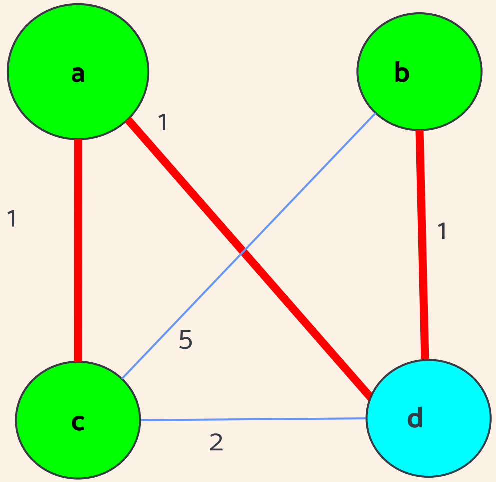


Terminal nodes: a,b,c
Normal nodes: d

Similarly....

C(b, {a,c}) = 3
C(c, {a,b}) = 3

A small example



Terminal nodes: a,b,c
Normal nodes: d

Therefore , length of steiner tree is 3.

The Algorithm

Initialization:

- Let C(u,S) represent the minimum cost to connect vertex u to all vertices in S. For singleton sets S={v} C(u,{v})=d(u,v)
- **Dynamic Programming Recurrence:** For all u$\in$V and subsets S$\subseteq$T where $|S|>1$:
  $C(u, S) = \min(\min_{v\in S}[d(u,v) + C(v , S \setminus \{ v \})] , \min_{S1=S\setminus\{ u \},S2=S\setminus S1,v\in V} [C(v , S1) + C(v , S2) + d(a,v)])$

- **Table Construction:** Compute C(u,S) iteratively for all u$\in$V and subsets S$\subseteq$T in increasing order of $|S|$.
- The minimum cost for the Steiner Tree is: $\min_{u\in V}C(u,T)$

Implementation Details

- Written in C++ (dreyfus_wagner.cpp)
- DP table in Cset
- ComputeTableLookup() is the recursive function that runs the dp algorithm
- findMinimumSteinerTree() finds the minimum tree iterating over all vertices in the terminal set
- Results are found using bash script

Github : https://github.com/NafiuRahman77/CSE462-Project

Time Complexity....?

Initialization (Shortest Paths): Computing shortest paths p(u,v) for all pairs u,v using Dijkstra's algorithm n times takes $O(n^2 \log n + n \cdot m)$

Time Complexity....?

$$C(u, S) = \min_{S1 \cup S2 = S,\ S1 \cap S2 = \varphi} [\, C(u, S1) + C(u, S2)\,]$$

The number of recursive calls corresponding to the above equation can be bounded from above by 3^k: for all $X = \varnothing$, $X \subseteq S$ we have to consider all $X' = \varnothing$, $X' \subseteq X$, and all $v \in V \setminus X$. The number of combinations can be upper-bounded by

$$\sum_{i=1}^{k} \binom{k}{i} \cdot \sum_{j=1}^{i-1} \binom{i}{j} \cdot n \leq n \cdot \sum_{i=1}^{k} \binom{k}{i} \cdot 2^{i-1} \leq n \cdot 3^k.$$

Since each combination leads to two table lookups (recursive calls corresponding to **C(u,S1)** and **s(u,S2)** and since we have to perform only constantly many operations for a fixed combination of **S1,S2 and v**, we obtain the upper bound **$O(3^k \cdot n)$** for the running time here.

Time Complexity....?

In this case **$O(2^k \cdot n)$** of pairs **S** and **u** are possible. For each fixed pair **S** and **u**, however, due to the consideration of **$v \in S$**, we get an additional factor of **$O(n)$**. Altogether, we have an upper bound of **$O(2^k \cdot n^2)$**

Time Complexity....?

So the algorithm runs in

$$O(3^k \cdot n + 2^k \cdot n^2 + n(n \log n + m))$$

Experiment Insights

1905087

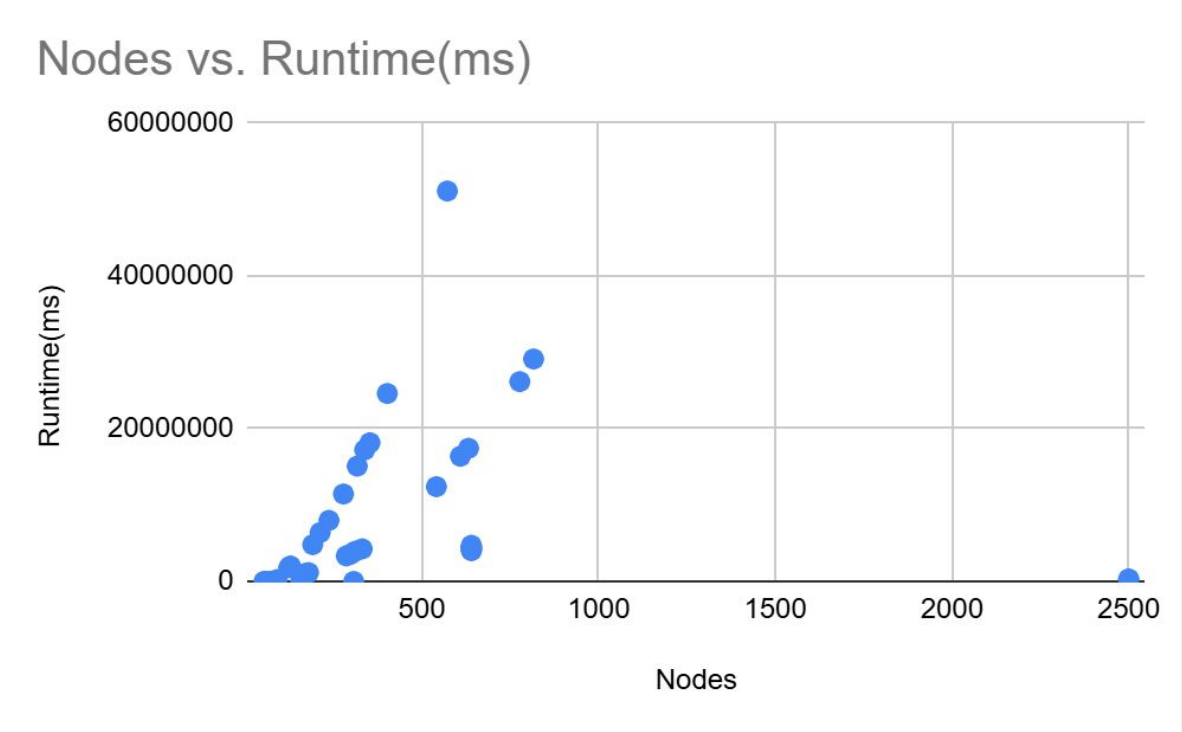
Experiment Insights

Dataset

PACE challenge 2018
Steiner Tree Track 1
Exact with low number of terminals

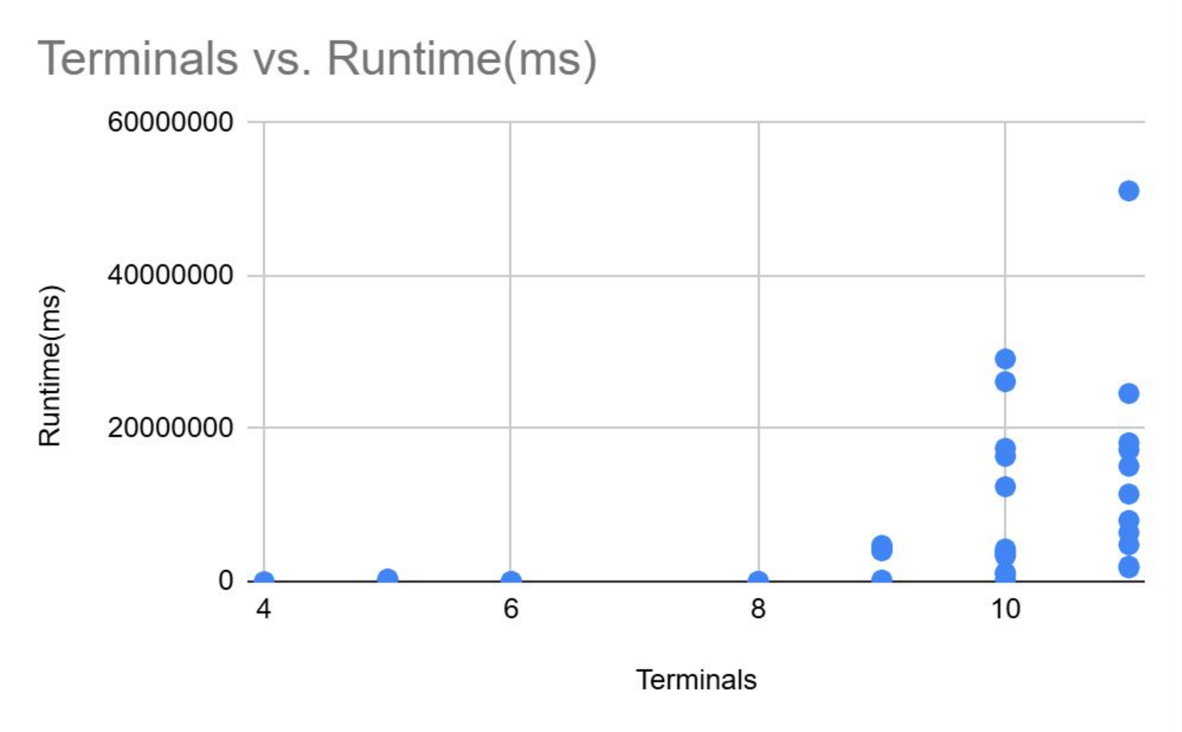
Experiment Insights

Results Overview



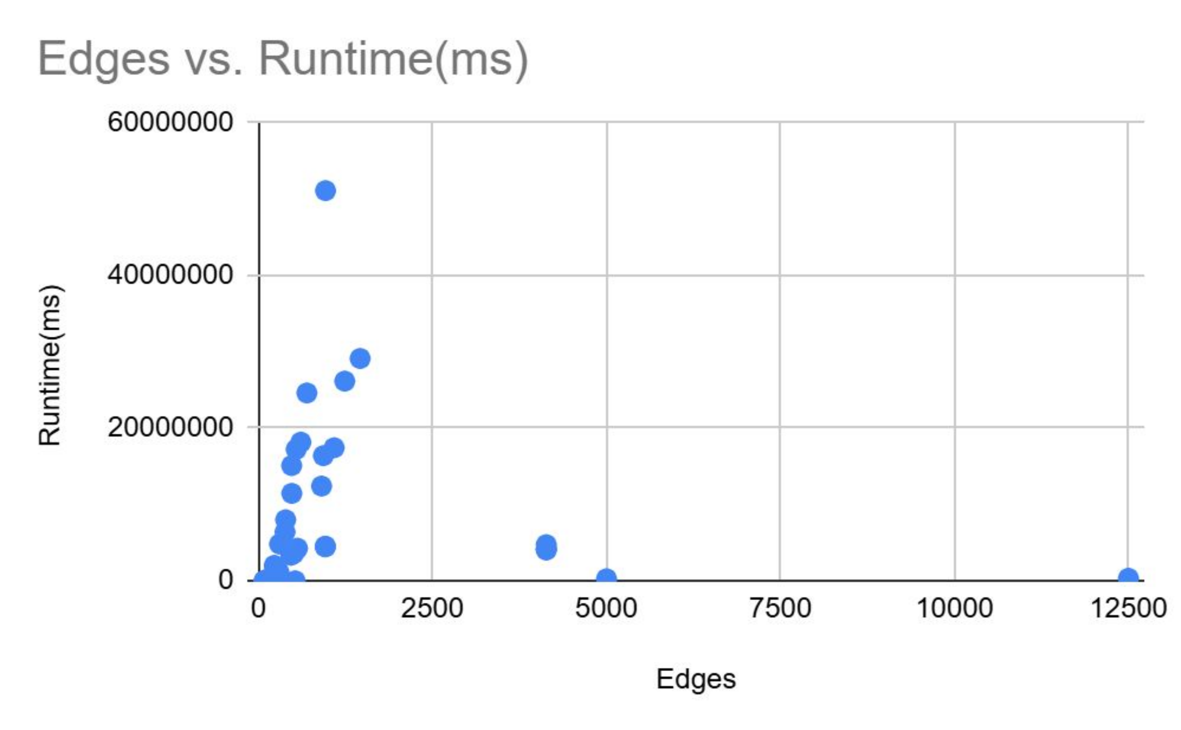
Experiment Insights

Results Overview



Experiment Insights

Results Overview



Experiment Insights

Three approaches

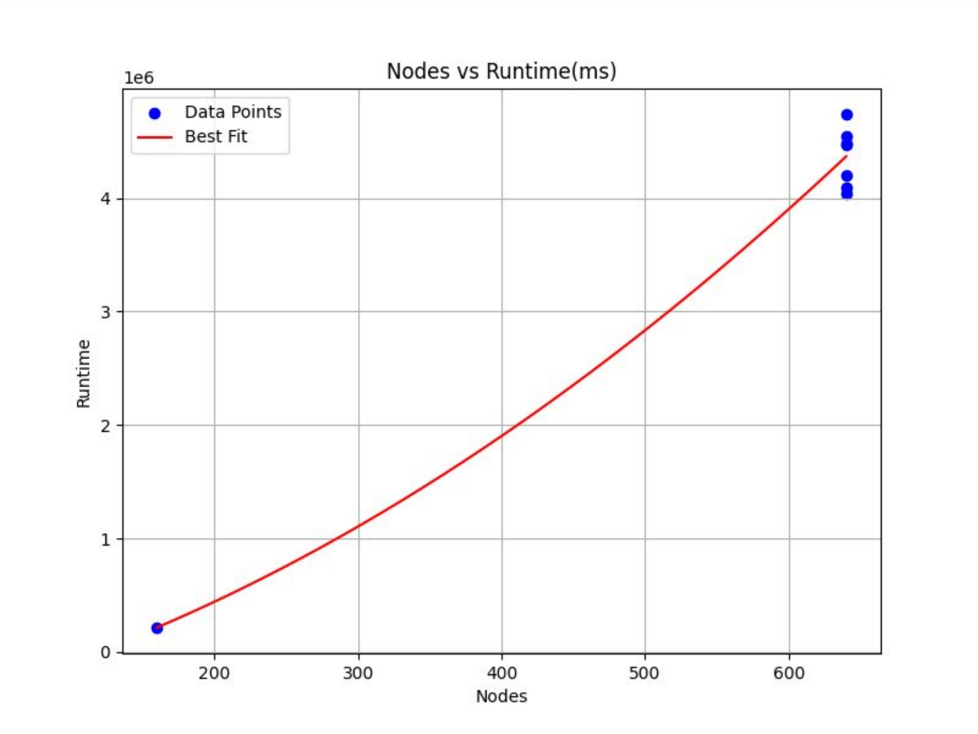
- Fixing number of terminals and varying nodes
- Fixing number of nodes and varying number of terminals
- Varying number of edges (graph density)

Experiment Insights

Approach 1: Terminals fixed

9 terminals

Nodes	Edges	Runtime(ms)
160	269	213577
640	960	4482487
640	960	4540714
640	960	4468618
640	4135	4739241
640	4135	4202292
640	4135	4042783
640	4135	4093606

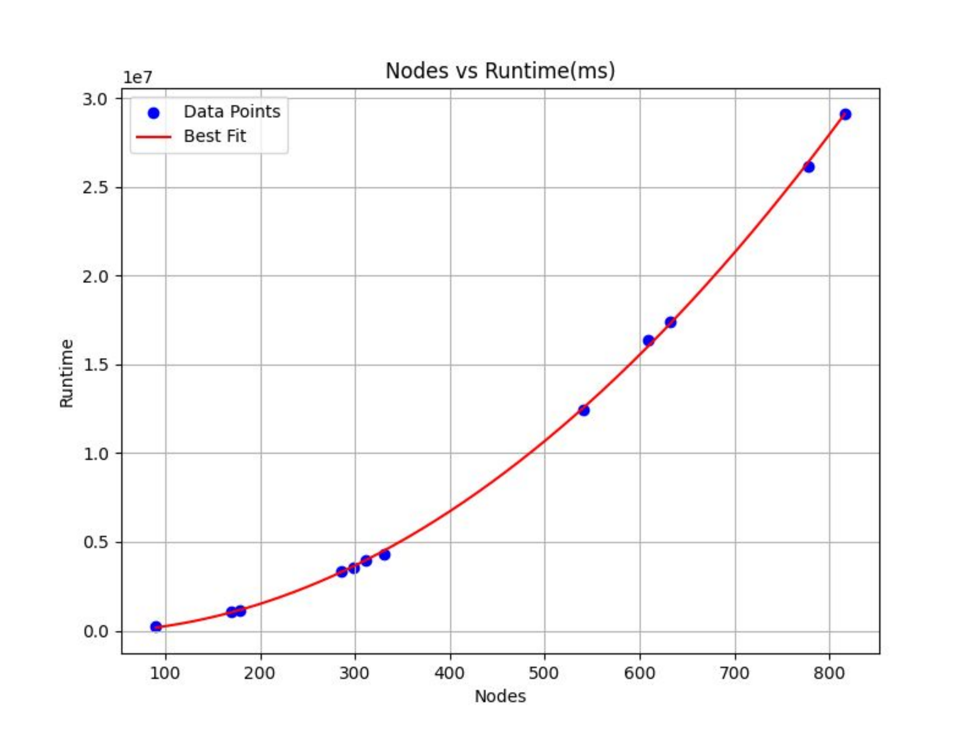


Experiment Insights

Approach 1: Terminals fixed

10 terminals

Nodes	Edges	Runtime(ms)
90	135	249577
169	280	1050595
179	293	1163662
286	465	3353328
298	503	3570315
311	530	3957323
331	560	4272861
541	906	12406357
609	932	16384636
632	1087	17422713
777	1239	26141050
816	1460	29104560

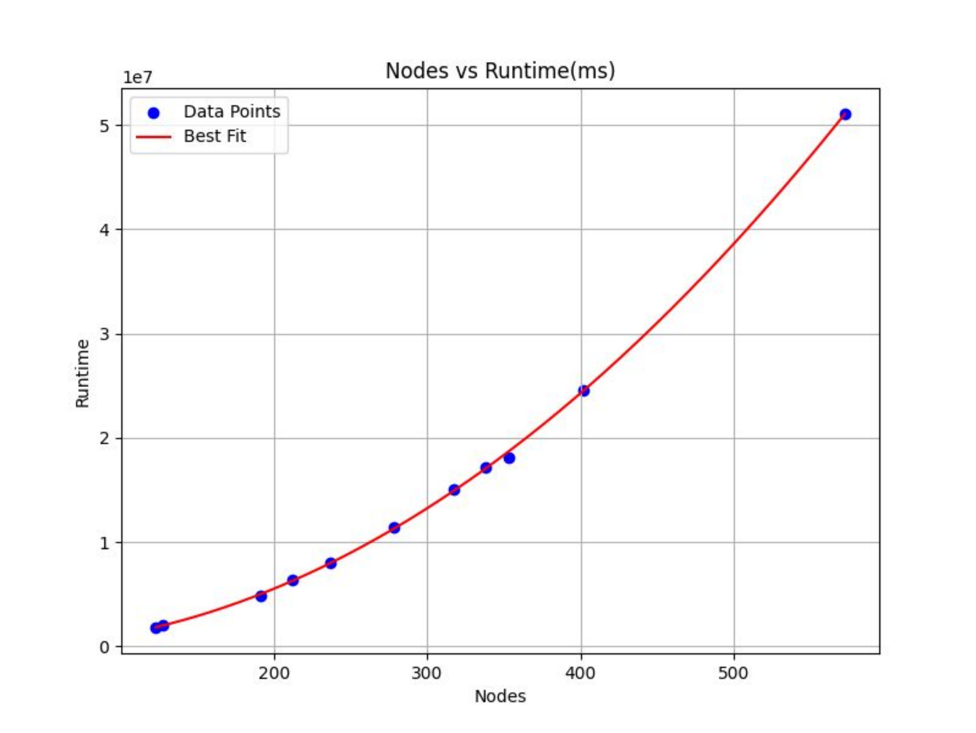


Experiment Insights

Approach 1: Terminals fixed

11 terminals

Nodes	Edges	Runtime(ms)
123	233	1805080
128	227	2034264
191	302	4823321
212	381	6400632
237	390	7995517
278	478	11450510
317	476	15094701
338	541	17216589
353	608	18154499
402	695	24592146
572	963	51074709

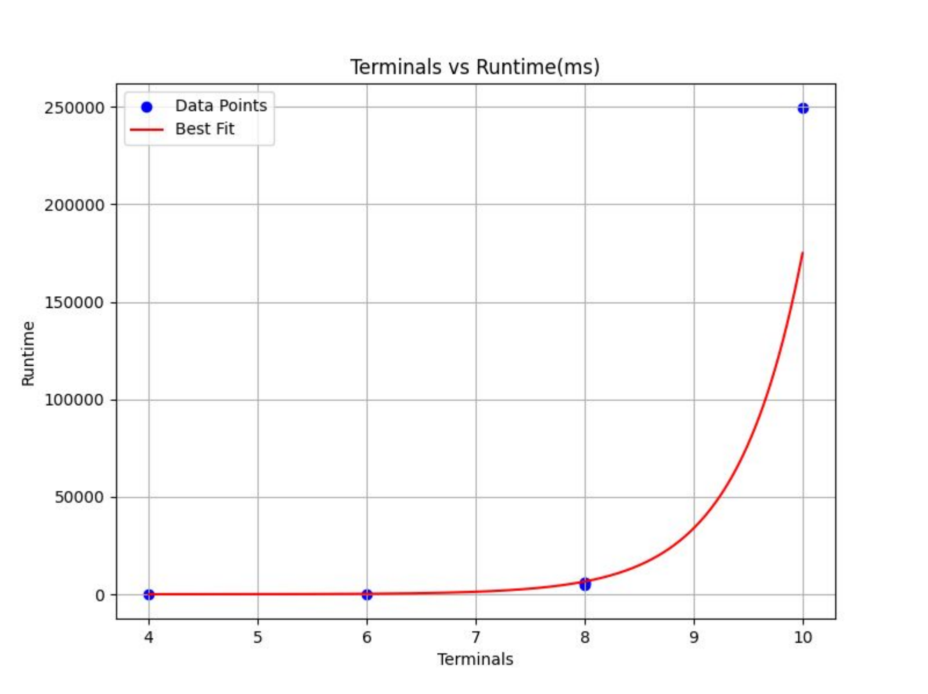


Experiment Insights

Approach 2: Node Range fixed

0-100 Nodes

Nodes	Edges	Terminals	Runtime
53	80	4	11
55	82	6	235
57	84	8	4762
64	288	8	5842
64	288	8	5945
90	135	10	249577

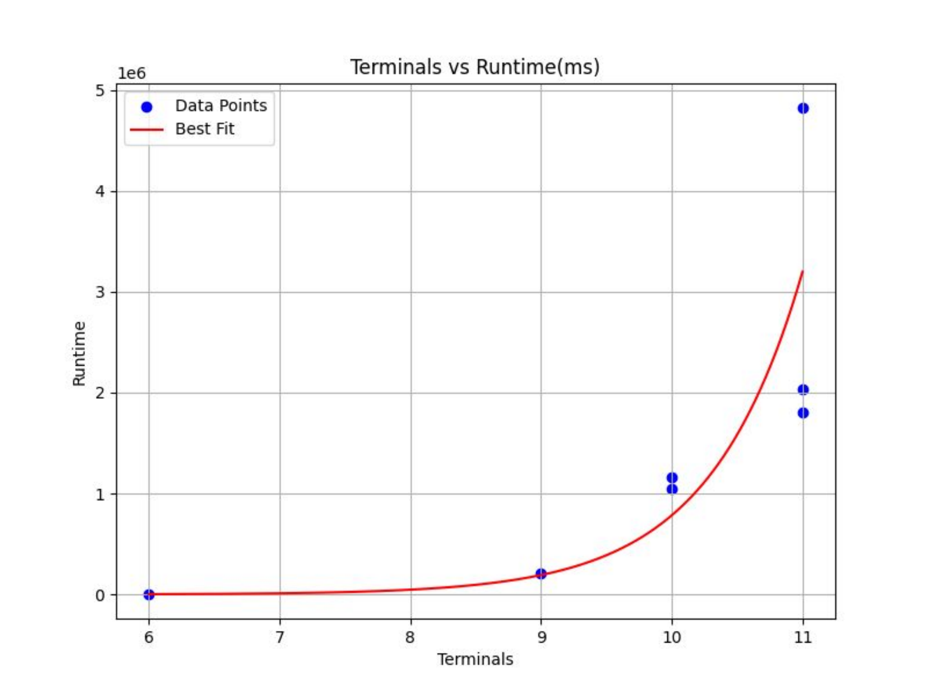


Experiment Insights

Approach 2: Node Range fixed

100-200 Nodes

Nodes	Edges	Terminals	Runtime
157	266	6	2369
160	269	9	213577
169	280	10	1050595
179	293	10	1163662
123	233	11	1805080
128	227	11	2034264
191	302	11	4823321

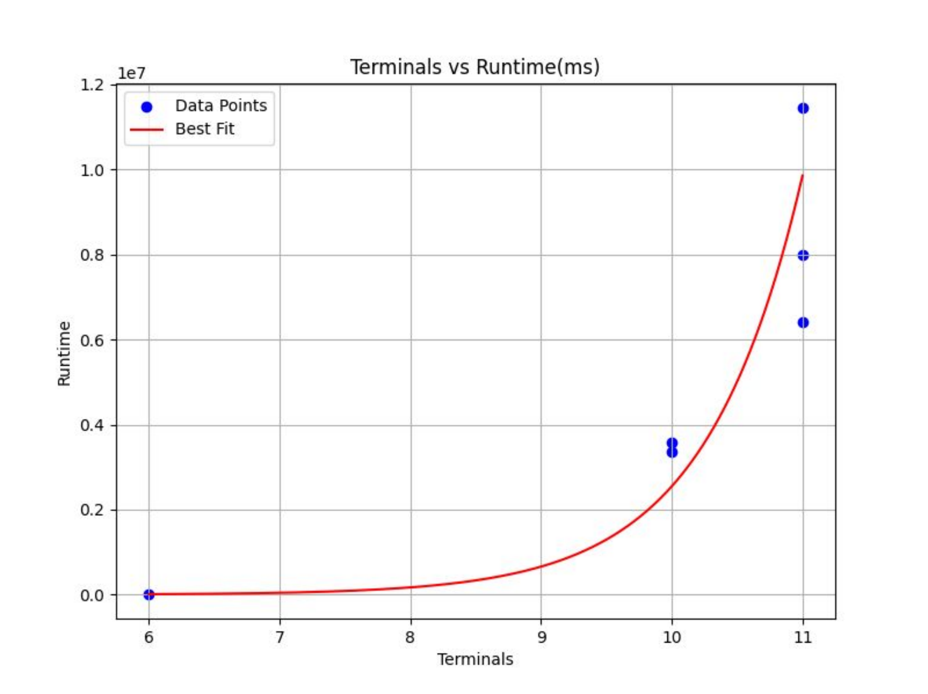


Experiment Insights

Approach 2: Node Range fixed

200-300 Nodes

Nodes	Edges	Terminals	Runtime
307	526	6	10060
286	465	10	3353328
298	503	10	3570315
212	381	11	6400632
237	390	11	7995517
278	478	11	11450510

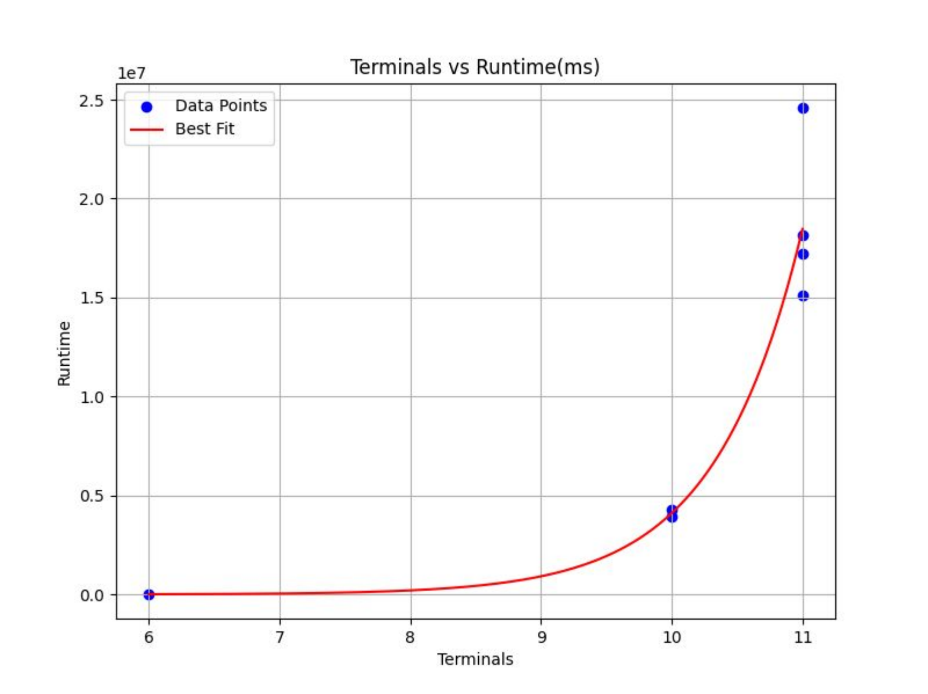


Experiment Insights

Approach 2: Node Range fixed

300-400 Nodes

Nodes	Edges	Terminals	Runtime
307	526	6	10060
311	530	10	3957323
331	560	10	4272861
317	476	11	15094701
338	541	11	17216589
353	608	11	18154499
402	695	11	24592146

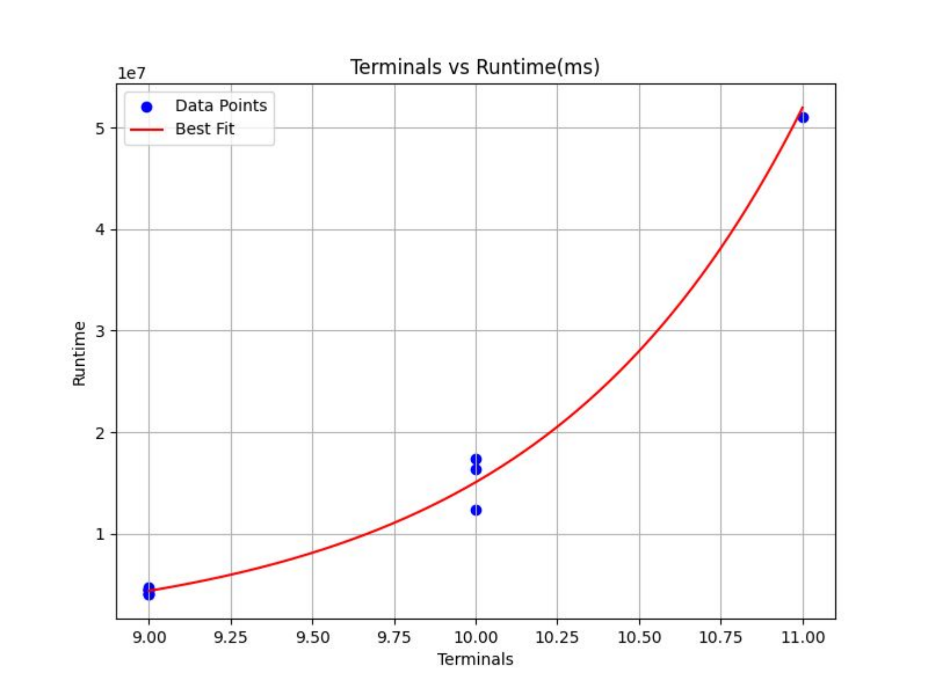


Experiment Insights

Approach 2: Node Range fixed

540-640 Nodes

Nodes	Edges	Terminals	Runtime
640	960	9	4468618
640	960	9	4482487
640	960	9	4540714
640	4135	9	4739241
640	4135	9	4202292
640	4135	9	4042783
640	4135	9	4093606
541	906	10	12406357
609	932	10	16384636
632	1087	10	17422713
572	963	11	51074709



Experiment Insights

Approach 3: Edges and Density

If everything else remains constant, positive correlation between edge count and runtime

Nodes	Edges	Terminals	Runtime
2500	3125	5	184439
2500	5000	5	272812
2500	12500	5	342680

Experiment Insights

Approach 3: Edges and Density

No general correlation between density and runtime

Nodes	Edges	Terminals	Runtime	Density
640	960	9	4468618	0.004694835681
640	960	9	4540714	0.004694835681
609	932	10	16384636	0.005034137067
632	1087	10	17422713	0.00545146342
572	963	11	51074709	0.005896905196
541	906	10	12406357	0.006202505648
402	695	11	24592146	0.008622721802
338	541	11	17216589	0.009499060629
317	476	11	15094701	0.009503653716
353	608	11	18154499	0.009786247747
331	560	10	4272861	0.01025359334
307	526	6	10060	0.01119839901
298	503	10	3570315	0.01136646103
286	465	10	3353328	0.01140964299
278	478	11	11450510	0.01241461704

Experiment Insights

Conclusions

- Runtime grows polynomially with node count
- Runtime grows exponentially with terminal count
- No strong correlation between edge count/density

Approximation Algorithm

1905088

Shortest Path Heuristic Approximation Algorithm

- proposed by Takahashi & Matsuyama in 1979
- 2(1-1/k)-approximation, where k = no of terminals

Step 1:

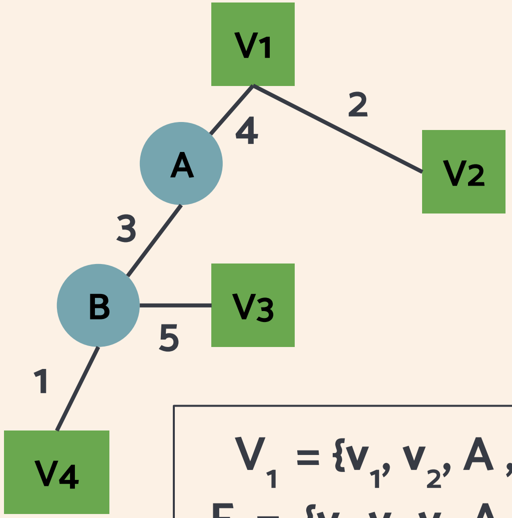
$$\boxed{\begin{aligned} V_1 &= \{v_1\} \\ E_1 &= \phi \end{aligned}}$$

Start with a **Subgraph** **$T_1 = (V_1, E_1)$**, consisting a **single terminal, say v_1.**

Step 2:

1. Find a terminal in the remaining terminals, say $\mathbf{v_i}$, such that $\mathbf{c(V_{i-1},v_i)}$ **is minimized**. (Using **Dijkistra's Algorithm**)

2. Construct Tree, $T_i = (V_i, E_i)$ by **adding PATH($\mathbf{V_{i-1}, v_i}$) to $\mathbf{T_{i-1}}$.**



$V_1 = \{v_1, v_2, A , B, v_4, v_3\}$

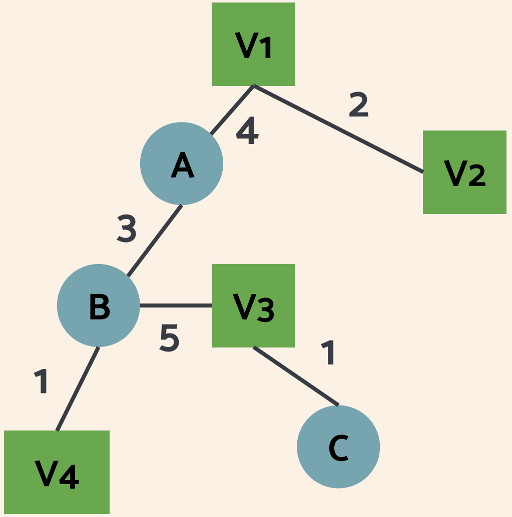
$E_1 = \{v_1\text{-}v_2, v_1\text{-}A, A\text{-}B, B\text{-}v_4, B\text{-}v_3\}$

Local Search

- proposed by M.G.A. Verhoeven, M.E.M. Severens, E.H.L. Aarts in 1996

Initialization:

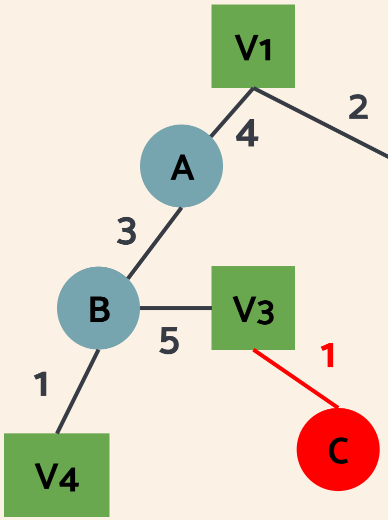
1. Take the **edge_set** of Steiner Tree solution gotten from greedy heuristic (Shortest Path Heuristic).
2. Compute **current cost** of the solution.
3. Identify all nodes present in the solution by traversing edgeSet (**nodes_in_solution)**.
4. **Initialize best_cost** as the cost of the current solution.

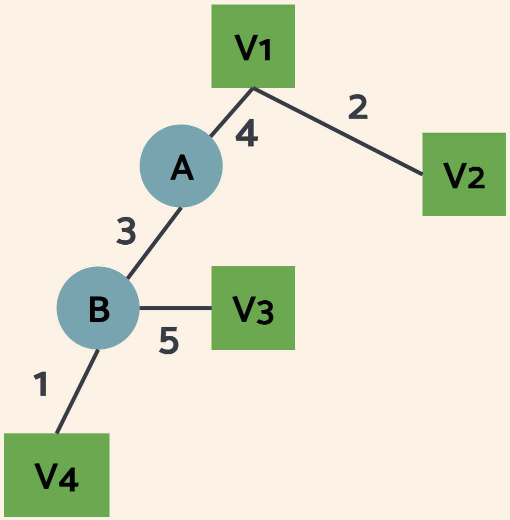


Phase 1: Attempt to Remove Non-Terminal Nodes:

1. Iterate through each node in **nodes_in_solution**
2. **Skip terminal nodes**
3. For each **non-terminal node**:
   a. Identify **all edges connected to this node** from edgeSet
   b. Create a new solution by **removing these edges** from the current solution
   c. Check if the remaining edges keep **all terminals connected**
   d. **If terminals remain connected**,compute **the cost of the new solution**
   e. If the new cost is **lower** than the current best_cost **update edge_set & best_cost**.

Phase 1: Attempt to Remove Non-Terminal Nodes:

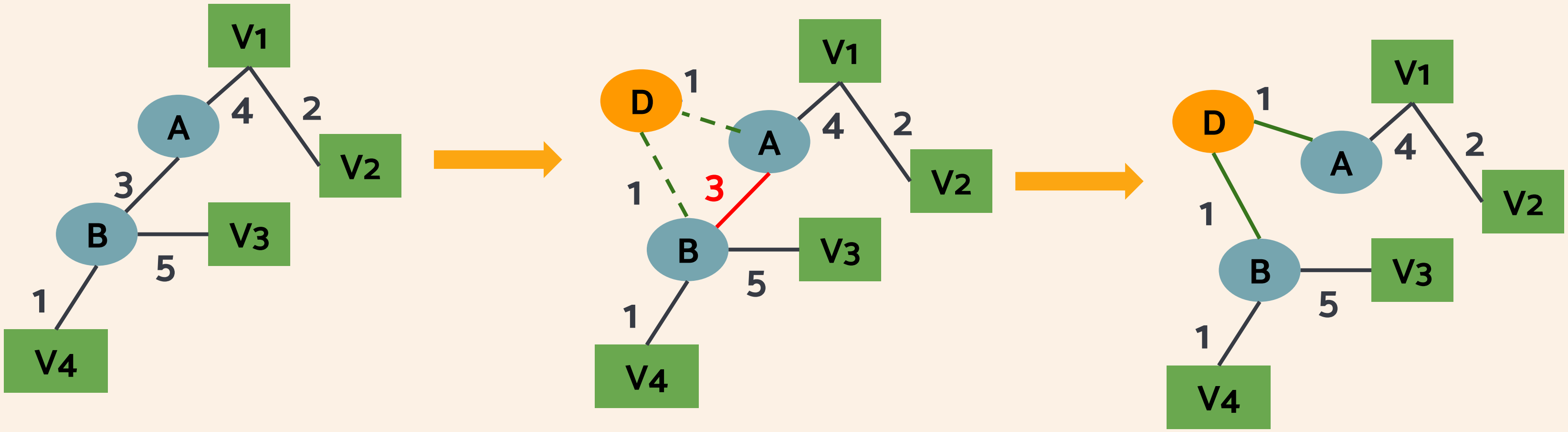




Phase 2: Attempt to Add Non-Terminal Nodes:

1. Iterate through all nodes in the graph
2. **Skip** nodes that are **already in the solution** or are **terminal nodes**
3. For each **non-terminal node**:
    a. Identify **edges** connecting this node to **nodes in the current solution**
    b. Check if adding this node introduces **at least 2 new edges**. **Add** these edges to the current solution
    c. If new edges are added, **prune** previous edges & **check for connectivity**
    d. If terminals remain **connected**,compute **the cost of the new solution**
    e. If the new cost is **lower** than the current best_cost **update edge_set & best_cost**.

Phase 2: Attempt to Add Non-Terminal Nodes:



Phase-1 **removes non-terminal leaves**	Phase-2 **improves connectivity** by including additional nodes and their edges if they contribute to lower cost
Uses **iterative first improvement/best first search**, which **decreases runtime** compared to **iterative best improvement**	Takes each terminal node as a starting point to get the **best possible steiner tree** in the greedy (shortest path heuristic) stage. **Multiple initialization** helps the algorithm **to not get stuck at local optima**.

Time Complexity Analysis

Shortest Path Heuristic Stage:

- Finding the shortest path between two vertices takes **$O(N^2)$** time complexity by **Dijkstra's Algorithm**., where N is the no of nodes in the tree.
- The algorithm finds the shortest path between **k terminals**, which takes **$O(kN^2)$** time complexity.
- It is **run k times** to get the best possible steiner tree, making the complexity **$O(k^2N^2)$**

Local Search Stage:

Phase 1:

- If there are N nodes in the tree, removing each non-terminal node takes approximate **$O(N)$** time.
- Computing The cost for removing each non-terminal node, requires a traversal through approximately all the edges in the tree, making the time complexity **$O(E)$**

∴ Overall time complexity = **$O(NE)$**

Time Complexity Analysis

Local Search Stage:

Phase 2:

- If there are N nodes in the tree, adding each non-terminal node takes approximate **$O(N)$** time.
- Pruning the edges require approximately **$O(NE)$** time complexity as the process similar to phase 1.

$\therefore$ Overall time complexity = **$O(N^2E)$**

$\therefore$ Overall time complexity for local search = **$O(NE) + O(N^2E)$ = $O(N^2E)$**

Time complexity of the Algorithm = $O(k^2N^2) + O(N^2E)$

Approximation Ratio Analysis

- Shortest path heuristic approach reports an

Approximation ratio ≤ 2(1-1/k)

*for all n & k(2≤k≤n-1); **k= no of terminals***

(if k=n approximation ratio = 1)

- Local search, in practice, brings the

Approximation ratio ~ 1.55

Worst Case Bound : 2

Because of local search, the approximation ratio is not tight

MinPath vs Minpath+Local Search

	MinPath	MinPath+Local Search
Time Complexity	$O(kn^2)$	$O(k^2n^2) + O(n^2e)$
Approximation Ratio	$\leq 2(1-1/k)$	~ 1.55
Approximation Ratio Bound	Tight	Not Tight

Implementation Details

- Basic code implementation in **C++**
- **Bash Script** to run the instances & generate csv
- **Python** to generate graphs on the results

Github : https://github.com/NafiuRahman77/CSE462-Project

Experiment Insights Algorithm II

1905082

PACE 2018

Dataset Link:
https://github.com/PACE-challenge/SteinerTree-PACE-2018-instances/blob/master

Track Number	Data	Used for
Track A	197	Both Algorithms
Track C	50	Approximation Algorithms

Track A

Total instances used	197
Number of terminals	4-104
Number of nodes	53-19083
Number of edges	80-204480

Track C

Total instances used	50
Number of terminals	16-88
Number of nodes	320-38282
Number of edges	640-71521

Defining Dense/Sparse

This logarithmic relationship often serves as a good divider:

- If *E/V* is less than *logV*, the graph is likely **sparse**.
- If *E/V* is greater than *logV*, the graph is moving towards being **dense**.

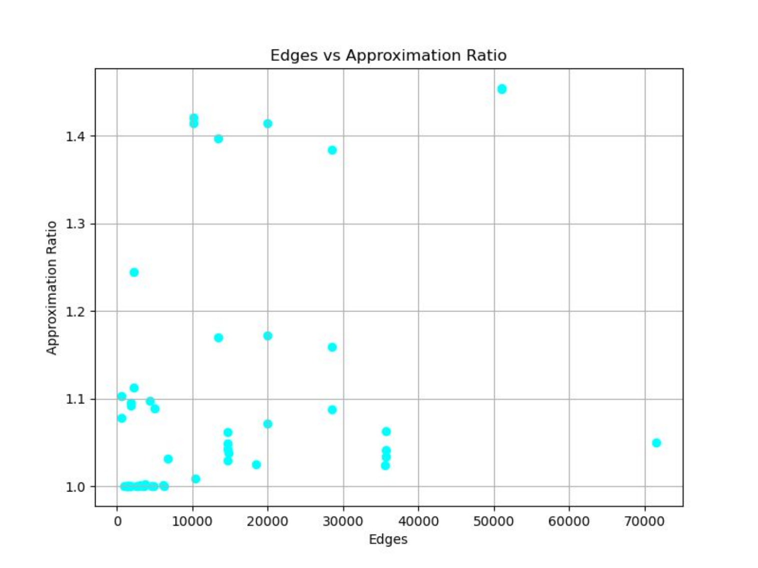
E/V<= logV; Sparse

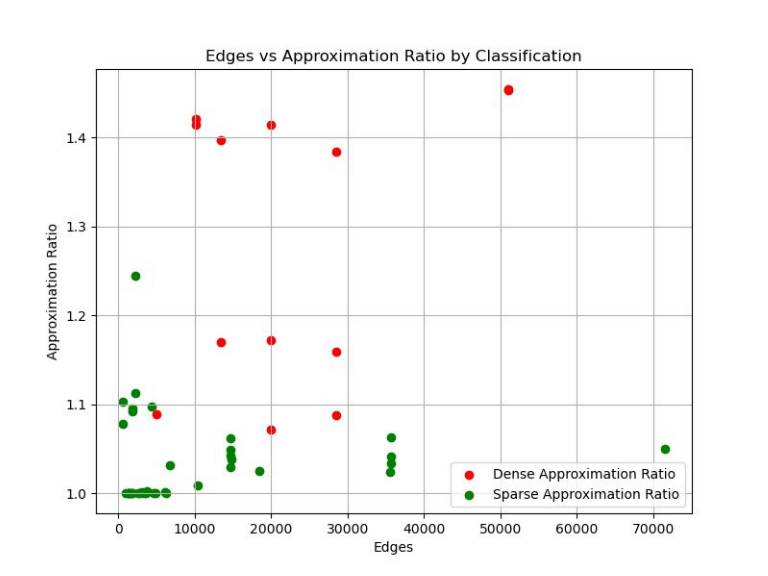
E/V>logV; Dense

E= Edge Count, V = Node Count

Track C
Experiments

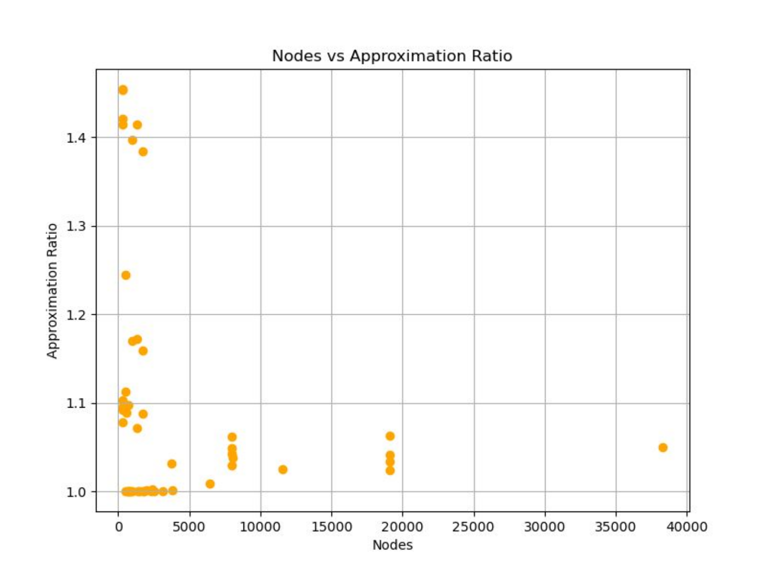
Approximation Ratio Analysis

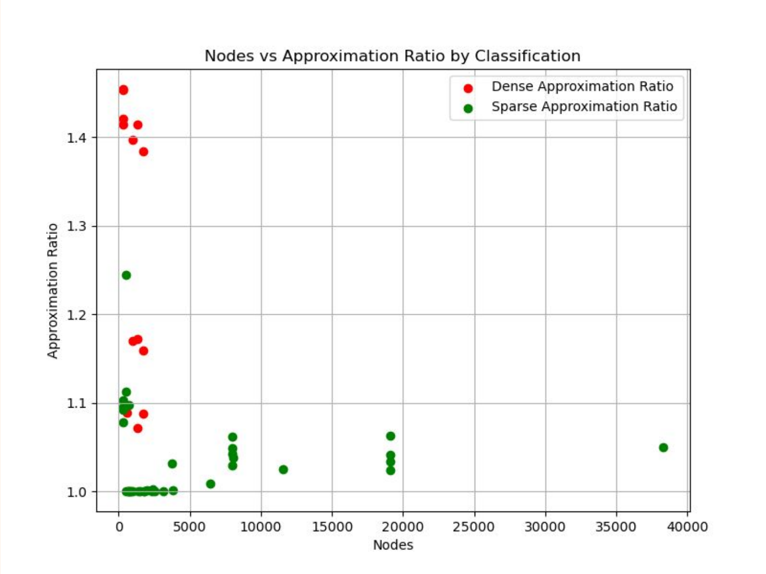




Increase of number of edges increases approximation ratio in many cases **(not clear pattern).**

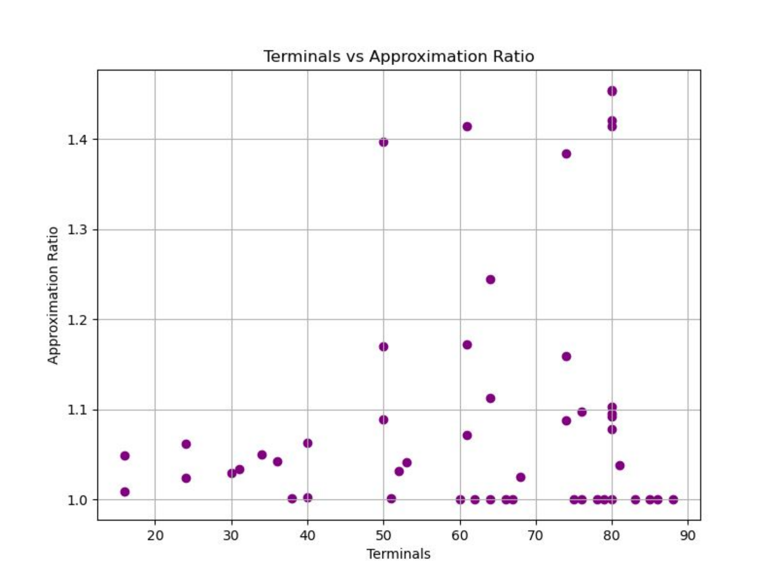
Approximation Ratio Analysis

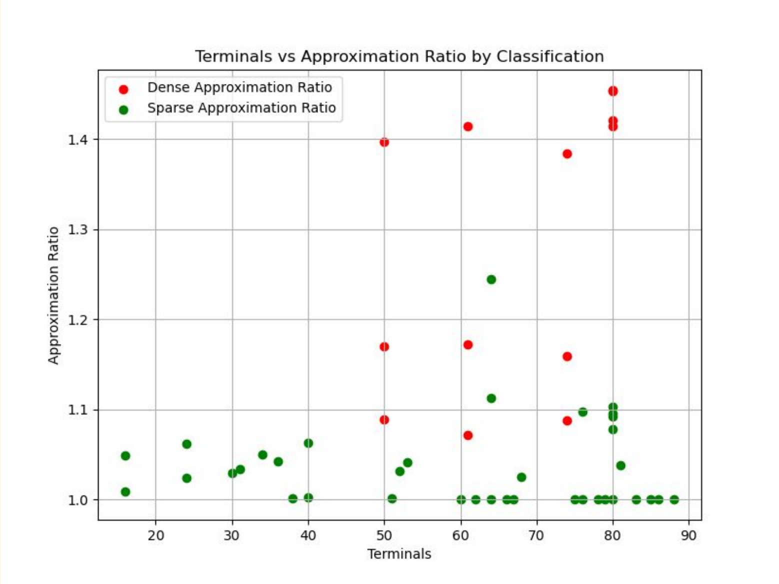




No relation can be derived from node increase.

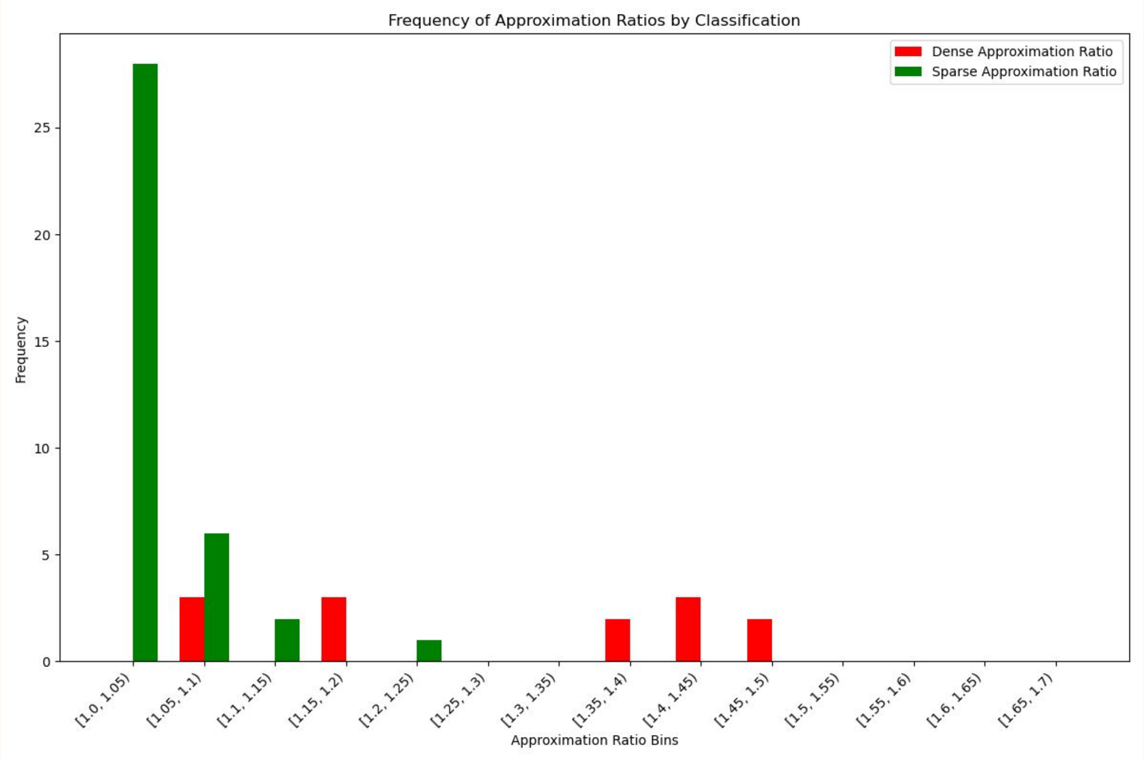
Approximation Ratio Analysis





Increase of number of terminals increases approximation ratio **for dense graphs (clear pattern).**

Approximation Ratio Frequency



How well our algorithm ran on 50 instances.
Recall, **Approximation ratio ~ 1.55**
The plot validates our algorithm.

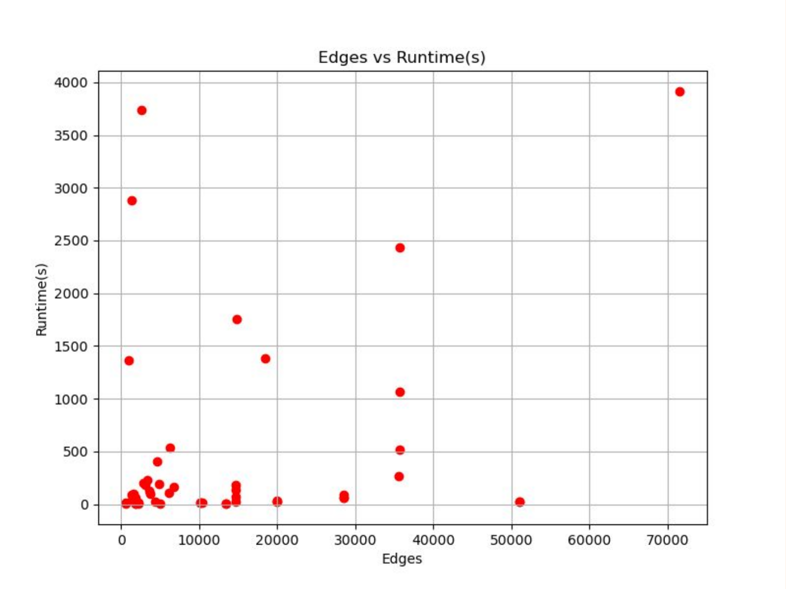
Conclusion on Ratio

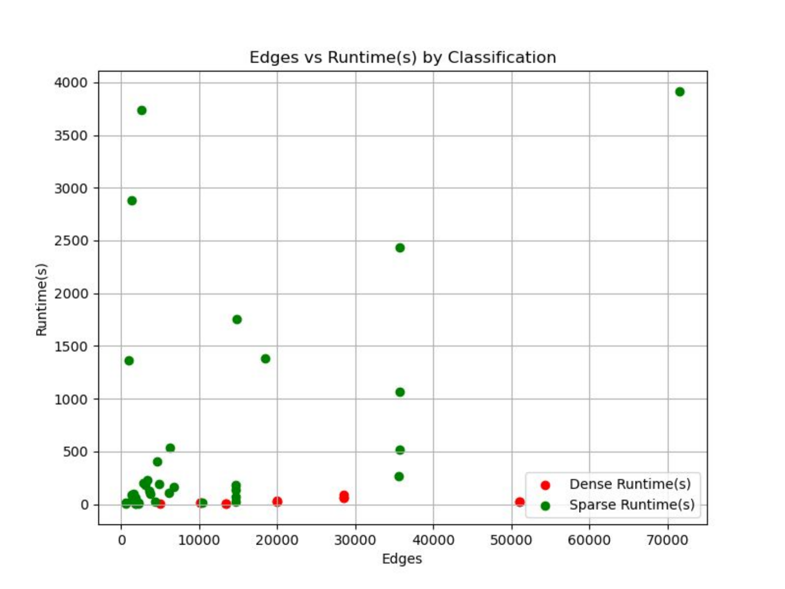
- **Increasing the number of terminals in dense graphs clearly raises the approximation ratio** due to the high connectivity and the many potential paths, making optimization more complex. **In sparse graphs, the effect on the approximation ratio is less evident** as the limited connections obscure straightforward relationships.

- In all scenarios, **dense graphs show a higher approximation ratio compared to sparse graphs**, reflecting the increased complexity introduced by numerous connections and options for network configuration.

- The instances are bounded within **Approximation ratio ~ 1.55 as derived.**

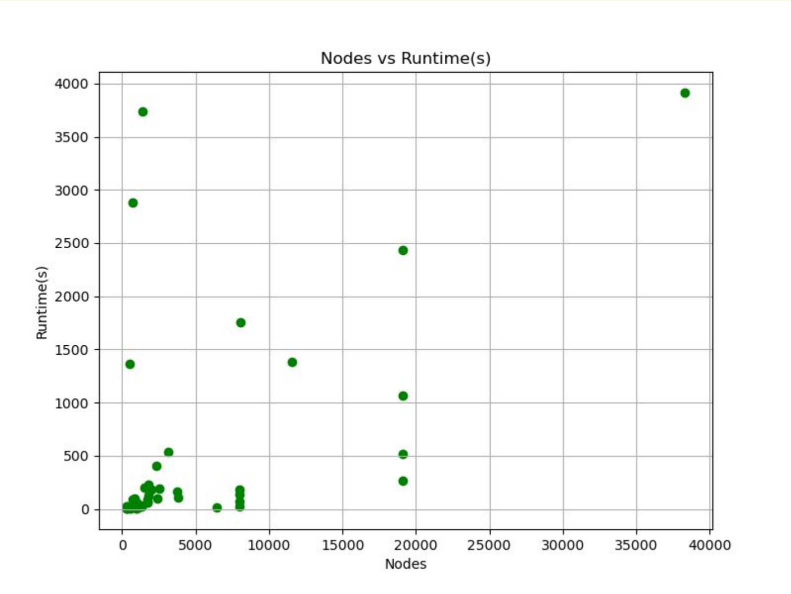
Runtime Analysis

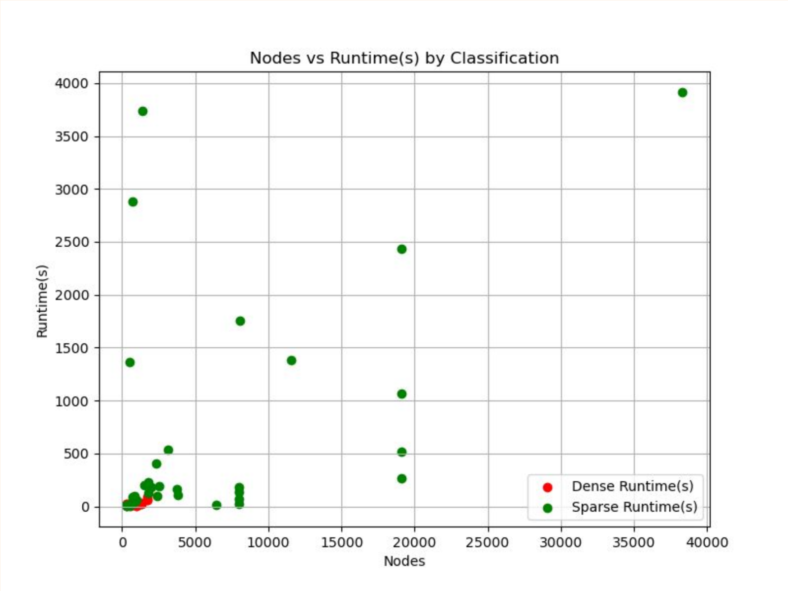




Runtime increases as edge number increases.

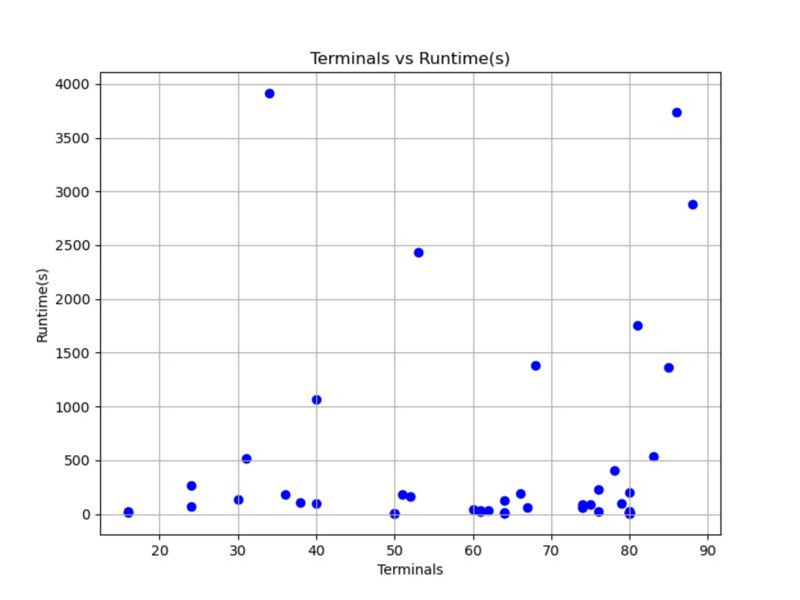
Runtime Analysis

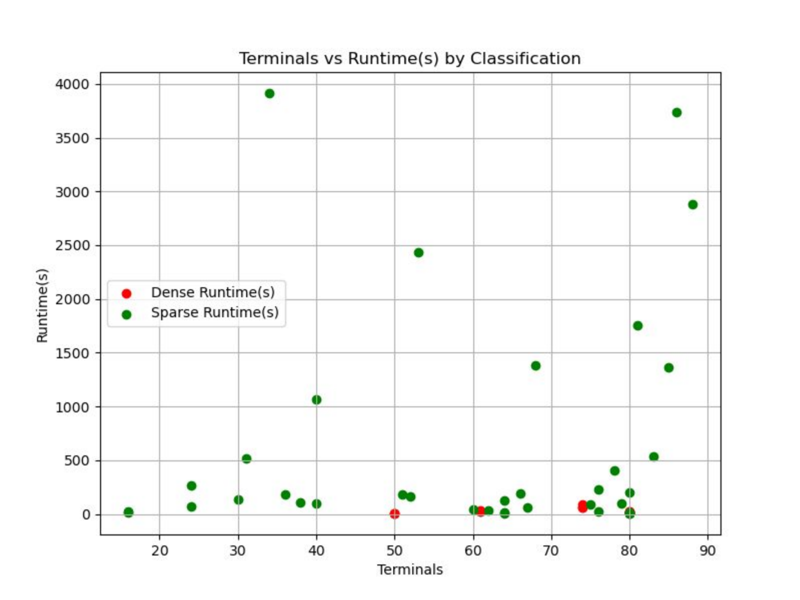




Runtime increases as node number increases.

Runtime Analysis





Runtime increases as number of terminals increases.

Conclusion on Runtime

- Dijkstra's may need to traverse comparatively more nodes to find the shortest path between any two nodes in a sparse graph, **due to the lack of direct paths.**

- In sparse graphs, each edge removal or addition can drastically change the connectivity, potentially requiring more careful and thus **slower examination.**

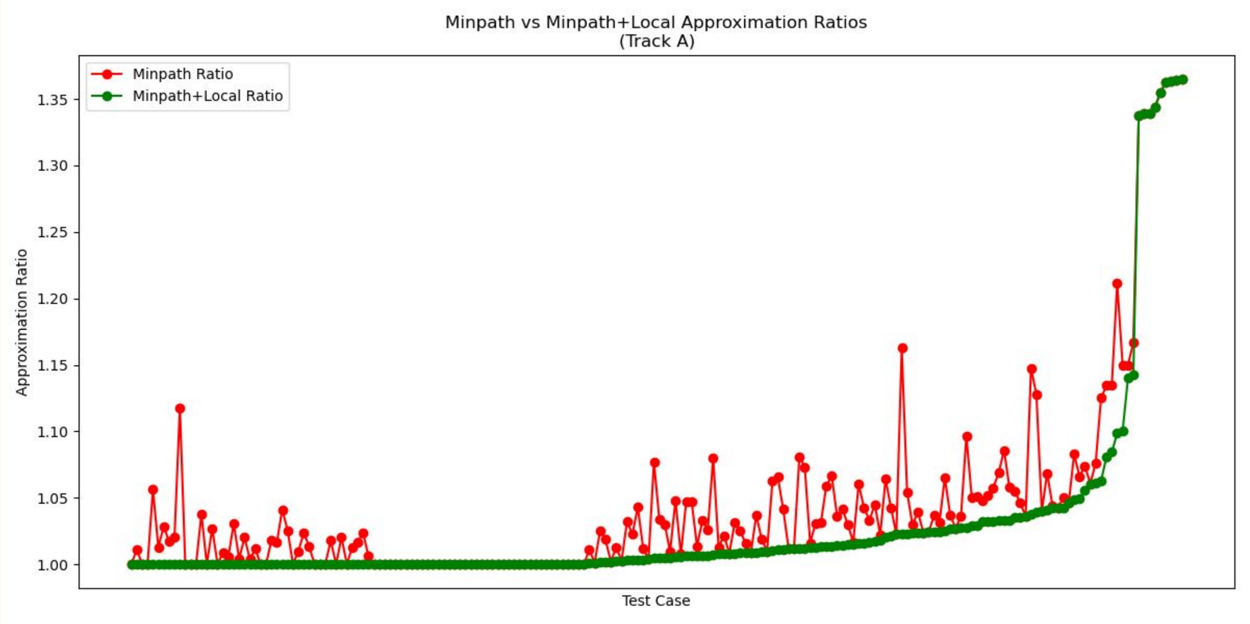
- In dense graphs, while there are more potential changes to evaluate, each change might not require as extensive a re-evaluation of the solution's viability, **leading to quicker iterations** but more of them.

Final Verdict

- For Dense Graphs, Approximation Ratio ↑ Runtime ↓
- For Sparse Graphs, Approximation Ratio ↓ Runtime ↑

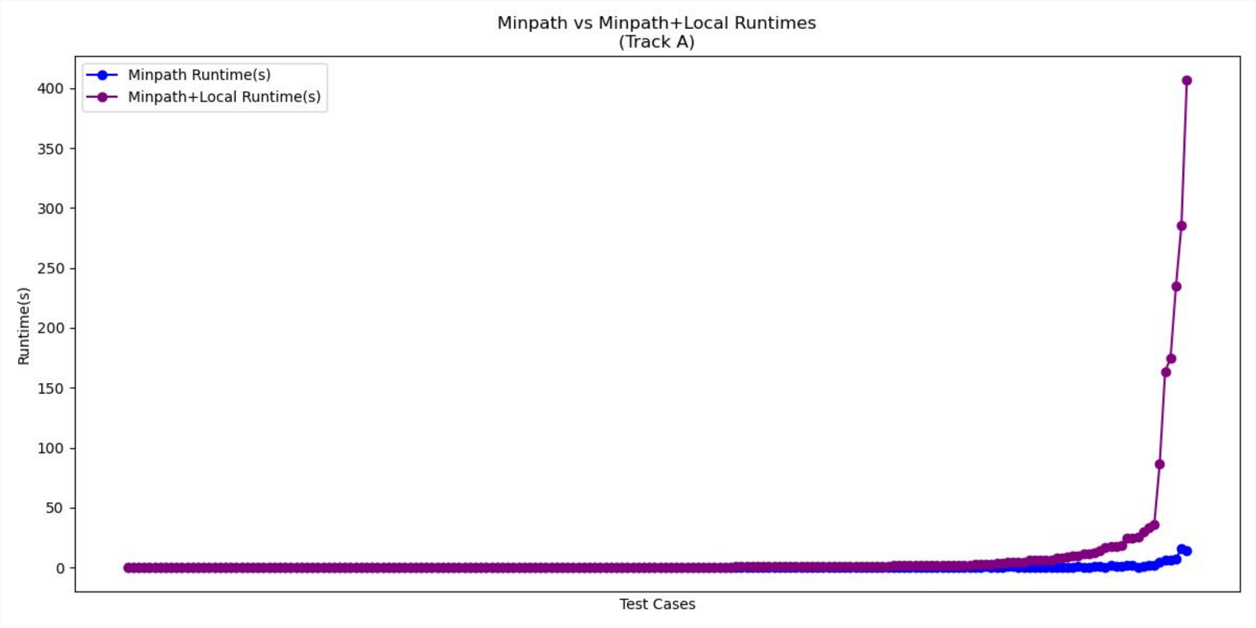
Track A Experiments

Approximation Ratio Analysis



- **Approximation ratio smoothens after local search** as seen in graph for most instances.
- Minpath tries to find a solution as soon as possible, and local search tries to improve on it.

Runtime Analysis



- **Increase of non-terminal nodes directly affect local search,** increasing runtime.
- Non-terminal nodes have trivial effect on just minpath algorithm.

Memory Usage

Data Structures Used

- **Graph:** $n \times d \times \text{sizeof}(\text{pair}{<}\text{int, int}{>})$
- **Maps:** $m \times (\text{sizeof}(\text{pair}{<}\text{int, int}{>}) + \text{sizeof}(\text{int}) + \text{overhead})$
- **Vectors:** Depending on their usage, $k \times \text{sizeof}(\text{int})$ for each vector
- **Boolean Vector:** $n \times \text{sizeof}(\text{bool})$ or less if optimized

n (number of vertices)
d (average degree per node, edges per node)
m (total number of edges)
k (number of elements in a vector, such as terminals)

pair<int, int>	8 bytes
int	4 bytes
Bool	1 byte

We utilized the <sys/resource.h> header to measure the memory of the process, observing a range from **3.5 MB to 10 MB** as we transitioned from smaller to larger, denser graphs

References

1. an approximate solution for the steiner problem in graphs
2. Verhoeven, M. G. A., Severens, M. E. M., & Aarts, E. H. L. (1996). Local search for Steiner tree problems in graphs. In V. J. Rayward-Smith, C. R. Reeves, & G. D. Smith (Eds.), Modern Heuristic Search Methods (pp. 117-129). Wiley.

Thanks!