A comprehensive study on the Maximum Leaf Spanning Tree Problem

180116 - Sanjida Islam Era
1805032 - Mahdee Mushfiq Kamal
1805091 - Ruhul Azgor
1805045 - Md. Hasanul Islam

Computer Science and Engineering, BUET
ruhulazgor@gmail.com

Algorithm Engineering Sessional Project Presentation
January 15, 2024

Presentation Overview

Spanning Tree

Spanning Tree:A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges.

- A spanning tree does not have cycles
- It cannot be disconnected

Spanning Tree Example

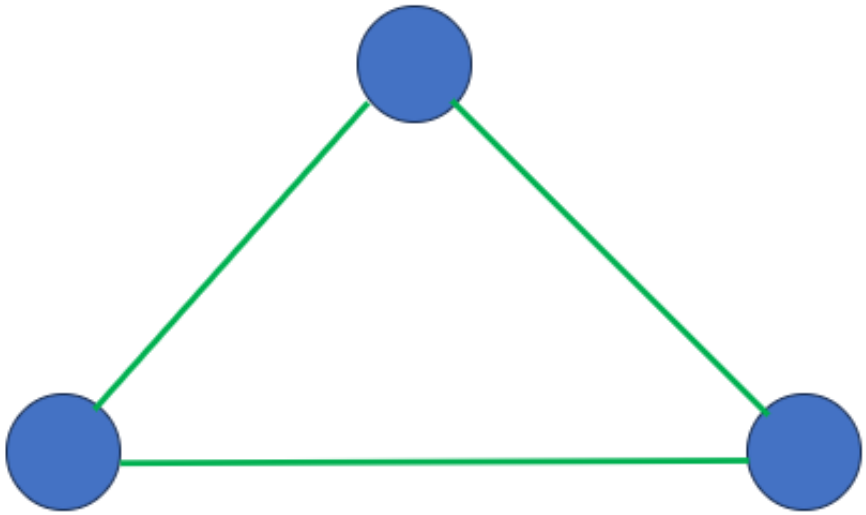


Figure: A simple Graph

Spanning Tree Example(Cont.)

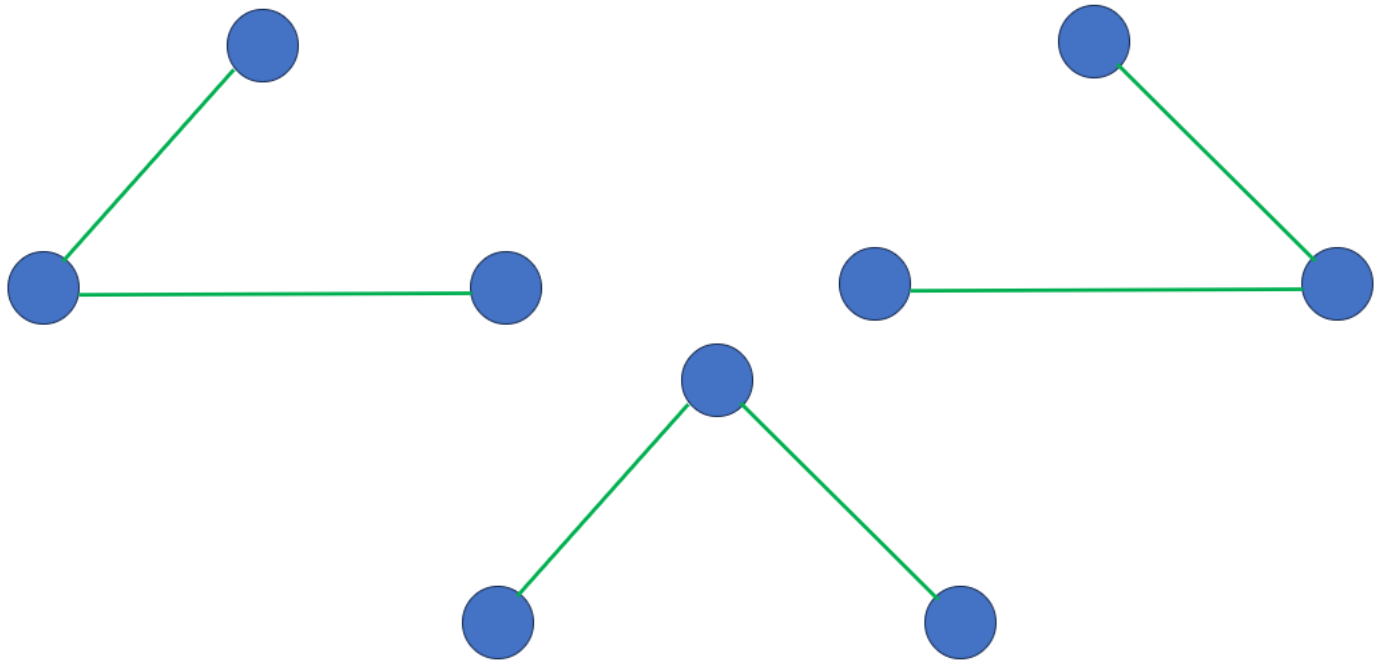


Figure: Spanning trees generated from the given example

Maximum Leaf Spanning Tree (MaxLST) Problem

Input: A connected graph G and an integer k
Parameter: An integer k that represents the minimum number of leaves desired in the spanning tree
Question: Does the graph G have a spanning tree that contains at least k leaves?

NP-completeness of MaxLST Problem

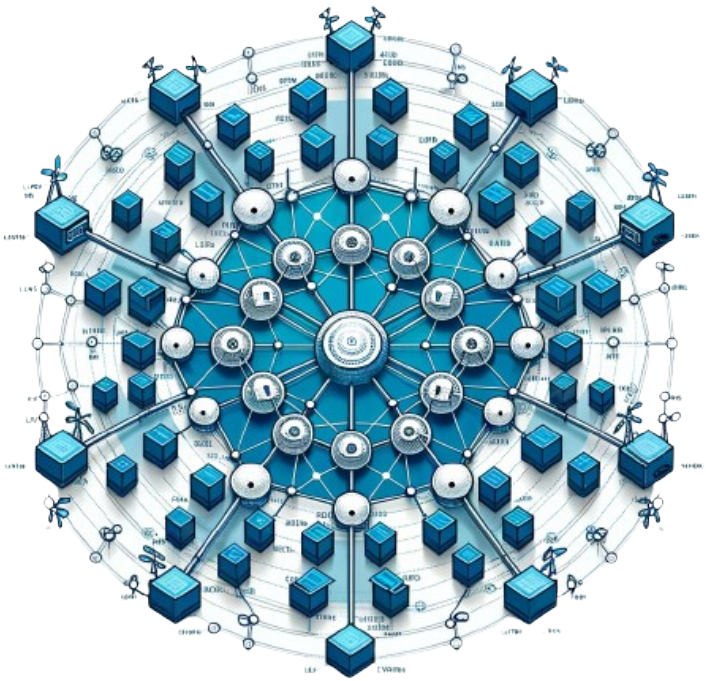
It is in NP: Given a spanning tree, we can verify whether it has at least k leaves in polynomial time by checking the degree of each vertex.

It is in NP-hard: We can reduce a known NP-hard problem, namely Vertex Cover, to the Minimum Connected Dominating Set (MinCDS) problem. Then we can reduce MinCDS to MaxLST.

Applications of MaxLST

Forest Fire Detection:
In 2022, Hosseinmardi, Farvaresh, and Kamalabadi used MaxLST to optimize a Wireless Sensor Network (WSN) for cost-effective forest fire detection in Iran's Arasbaran region[6].

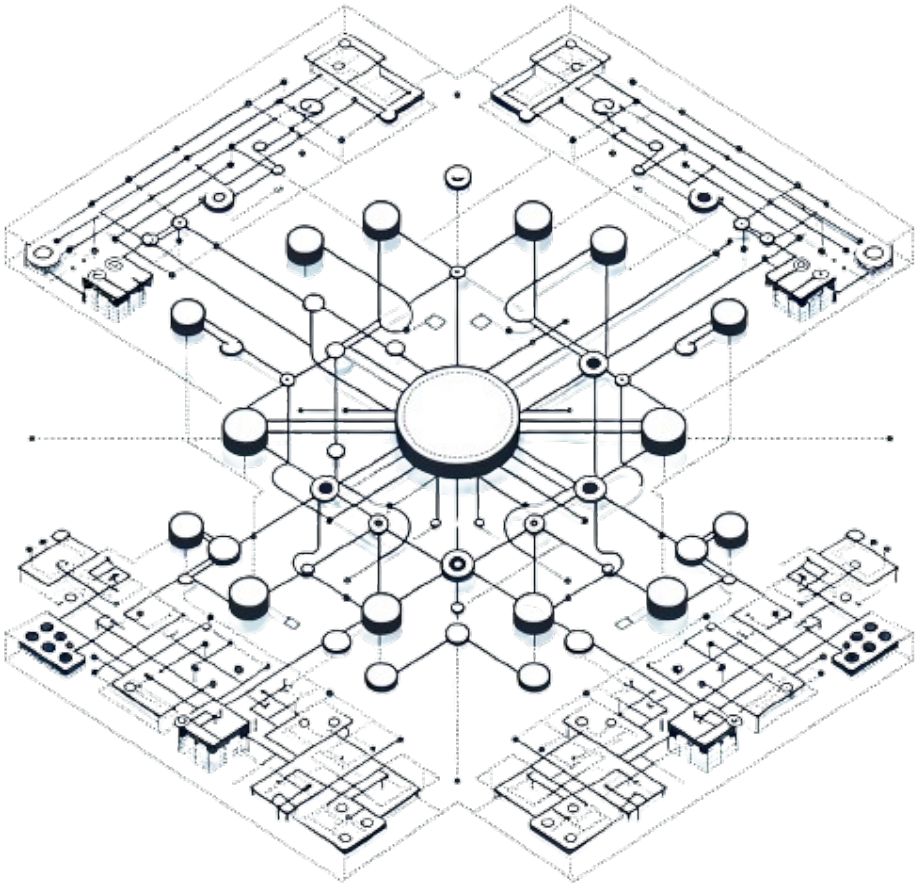
Applications of MaxLST(Cont.)



Broadcasting Networks:

- Broadcasting networks use nodes to spread data.
- Reducing the number of broadcasting nodes simplifies the network's layout, leading to fewer connection points and a more straightforward design.
- Each reduced node simplifies and trims network complexity.

Applications of MaxLST(Cont.)



Circuit Layouts :
In circuit design, the layout has minimal connection points with components directly connected, streamlining the circuit and reducing potential signal delays.

NP-hardness of Max-Leaf Spanning Tree

NP Hard Problem:

- A problem H is NP-hard if every problem in NP can be reduced to it in polynomial time.
- In other words, an NP-hard problem is at least as hard as the hardest problems in NP.
- Boolean satisfiability problem (SAT) is the first proved NP-hard problem. (Cook, 1971).
- Maximum Clique, Minimum Vertex Cover, Maximum Independent Set, Minimum Set Cover Problem etc are well known NP-hard problems. (Karp, 1972)

Prooving the NP-hardness of Max-Leaf Spanning Tree

- To proof the NP-hardness of a problem *H*, we need to reduce an existing NP-hard problem to it.
- To proof the NP-hardness of Max-Leaf Spanning Tree, we reduce the Minimum Vertex Cover problem to Minimum Connected Dominating Set (MinCDS) problem. and then reduce Minimum Connected Dominating Set to Max-Leaf Spanning Tree problem.

Vertex Cover Problem

- A vertex covers its incident edges. Vertex cover problem seeks a minimal subset of vertices that covers all the edges in the graph.
- In a graph $G = (V, E)$, a set S is said to be a vertex cover if and only if $S \subset V$ and for all edges $(u, v) \in E$, $u \in S$ or $v \in S$. The Vertex Cover problem is to find a vertex cover S, of minimum size, i.e., to minimize $|S|$
- Instance: A connected graph $G = (V, E)$ and an integer k Question Does G have a vertex cover of size at most k?

Minimum Connected Dominating Set Problem

- A vertex dominates itself and its neighbors. Minimum connecting dominating set problem seeks a minimal subset of vertices such that all vertices in the graph are connected and are dominated by some vertex in that subset.
- Given a connected graph $G = (V, E)$, a connected dominating set $S \subseteq V$ is such that for every vertex $u \in V$, either $u \in S$ or u has a neighbor $v \in S$, and the subgraph of G induced by S is connected.
  The connecting dominating set problem is to find a connecting dominating set S, of minimum size, i.e., to minimize $|S|$
- Instance: A connected graph $G = (V, E)$ and an integer k
  Question: Does G have a connected dominating set of size at most k?

Reducing VC to minCDS problem

Given an instance of Vertex Cover problem with graph G, we construct an instance of MinCDS problem with graph G' as follows:

- G' has all edges and vertices of G.
- For every edge $(u, v) \in G$, we add intermediate node on a parallel path in G', called w. Keeping (u, v) intact in G' , we add vertex w and edges (u, w) and (w, v) in G'.
- For every $(u, v) \notin G$, we add edge (u, v) in G'.

It can be proven that, G has a vertex cover of size at most k if and only if G' has a connected dominating set of size at most k.

Reducing VC to minCDS problem

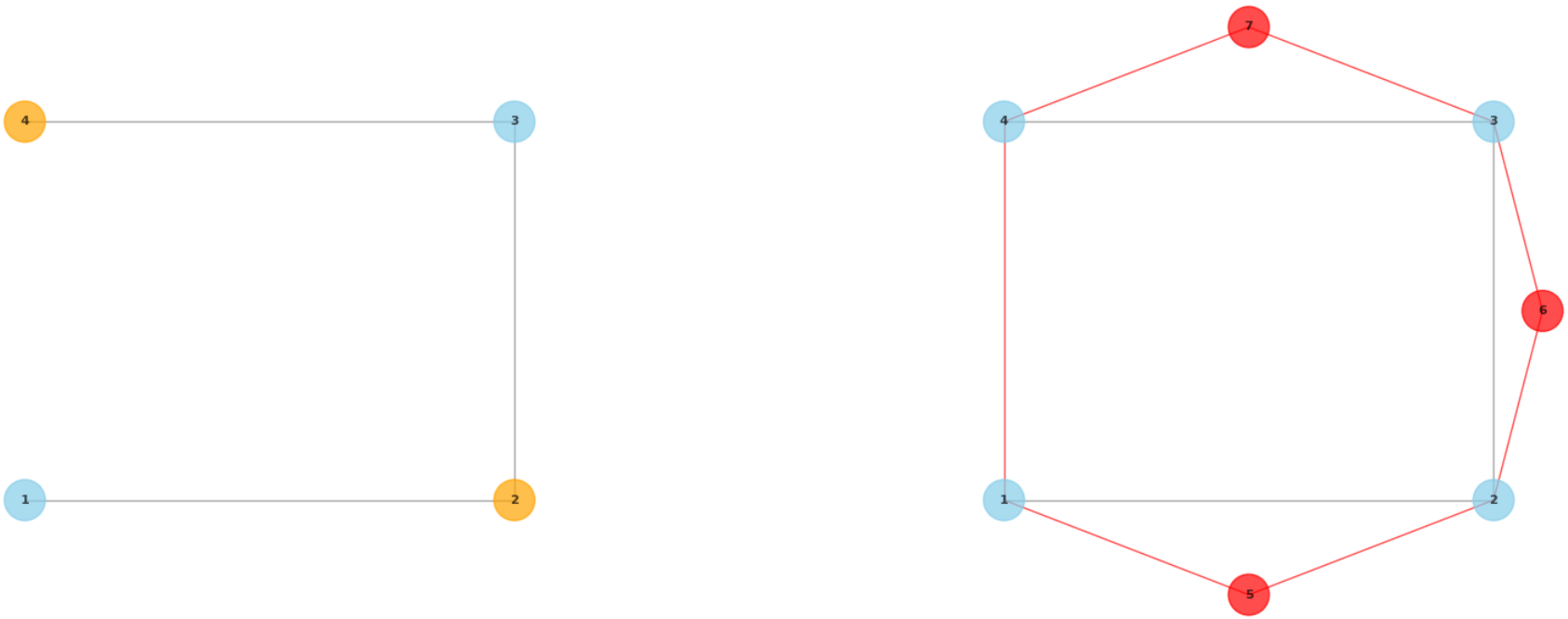


Figure: G and G'

Reducing VC to minCDS problem

Proof, Necessity:

- Nodes in S, dominates all the nodes (existing + new).
- G' is connected.

Proof, Sufficiency:

- Each newly created w in G' dominates existing nodes in G.
- w can be replaced by either u or v.
- Each w is corresponding to an edge $(u, v) \in G$.
- w replacing u or v is a vertex cover for the corresponding edge.

Reducing minCDS to MaxLST problem

A connected graph $G = (V, E)$ has a connected dominating set of size at most k if and only if it has a spanning tree of at least $n - k$ leaves. Proof, Necessity:

- Let T be a connected subgraph of G and $V(T) = S$ of $|S| - 1$ edges.
- For each vertex $v \in V - S$, we add an edge which makes v a leaf. So, $|V - S| \geq n - k$.

Proof, Sufficiency:

- Let T be a spanning tree of G with at least $n - k$ leaves.
- Let S be the set of vertices in G that are not leaves in T.
- $|S|$ is connected and has at most $n - (n - k) = k$ nodes.

Exact Algorithms

Table: Exact Exponential Algorithms

Algorithm	Runtime	Year	Author	Comment
Naive brute-force	$\Omega(2^n)$	-	-	Considering all possible leafset
Simple Branching [7]	$O(4^k \text{poly}(n))$	2008	Kneis, Langer, and Rossmanith	Parameterized in the number of leaves k
Improved Branching [2]	$O(3.72^k \text{poly}(n))$	2010	Daligault et al.	Parameterized in the number of leaves k
Connected Dom Set [4]	$O(1.9407^n)$	2008	Fomin, Grandoni, and Kratsch	Equivalent to MaxLST
Branching Algo [3]	$O(1.8966^n)$	2011	Fernau et al.	Branching algo based on some properties
Worst Case Lower Bound[3]	$\Omega(1.4422^n)$	-	-	Lower Bound

Approximation Algorithms

Table: Approximation Algorithms

Algorithm	Approx Ratio	Complexity	Author	Comment
Local Optimization [10]	Approx. 5	$O(n^4)$	Lu and Ravi	Year1996
Local Optimization [10]	Approx. 3	$O(n^7)$	Lu and Ravi	Year1996
Greedy [9]	Approx. 3	$O((m+n)\alpha(m,n))$	Lu and Ravi	Year1998
Incrementally building a subgraph[11]	Approx. 2	$O(m)$	Solis-Oba, Bonsma, and Lowski	Use expansions rule, Year2017
Expands the tree	Approx. 2	$O(m)$	Liao and Lu	Simple, Year2023

Does PTAS exist for MaxLST?

No PTAS exists unless $P = NP$

In 1994, Galbiati, Maffioli, and Morzenti concluded that, the Maximum Leaves Spanning Tree Problem does not have a polynomial time approximation scheme, unless P = NP [5].

Randomized, Heuristic, Meta-heuristic Algorithms

Heuristics

- BFS Algorithm
- Priority BFS
- Max Priority BFS

Meta Heuristics

- Simulated Annealing
- Genetic Algorithm
- Artificial Bee Colony Algorithm

Randomized, Heuristic, Meta-heuristic Algorithms

Graph		Opt.	BFS	PBFS	MaxPBFS	SA			ABC			GA		
n	m		solution	solution	solution	solution	iter.	time	solution	iter.	time	solution	iter.	time
4	4	9	8	8	**9**	**9**	14	0.16	**9**	0	0.07	**9**	0	0.68
	5	11	10	10	**11**	**11**	134	0.17	**11**	0	0.09	**11**	0	0.93
	6	14	12	13	13	**14**	198	0.23	**14**	0	0.12	**14**	1	1.12
	7	16	14	14	15	**16**	623	0.33	**16**	39	0.17	**16**	1	1.39
	8	18	16	17	**18**	**18**	177	0.39	**18**	0	0.21	**18**	0	1.72
	9	21	18	19	20	20	88	0.48	20	0	0.23	**21**	1	1.90
5	5	14	10	13	**14**	**14**	28	0.26	**14**	0	0.14	**14**	0	1.23
	6	18	12	17	17	**18**	1246	0.35	17	0	0.17	**18**	1	1.45
	7	20	14	19	**20**	**20**	3	0.45	**20**	0	0.21	**20**	0	1.72
	8	23	16	21	**23**	**23**	1737	0.58	**23**	0	0.31	**23**	0	2.11
	9	27	18	25	**26**	25	0	0.69	**26**	0	0.31	**26**	0	2.38
6	6	22	12	18	21	**22**	999	0.45	**22**	48	0.24	**22**	1	1.85
	7	26	14	**26**	**26**	**26**	0	0.59	**26**	0	0.29	**26**	0	2.19
	8	30	16	**30**	**30**	**30**	0	0.74	**30**	0	0.34	**30**	0	2.56
	9	34	18	32	33	**34**	700	0.90	**34**	1	0.51	**34**	1	3.13
7	7	29	14	27	**29**	**29**	143	0.76	**29**	0	0.37	**29**	0	2.89
	8	33	16	**33**	**33**	**33**	0	0.99	**33**	0	0.47	**33**	0	3.49
	9	39	18	37	37	**39**	3714	1.25	37	0	0.64	38	2	3.85
8	8	38	16	34	36	**38**	5019	1.29	37	11	0.76	**38**	1	4.36
	9	45	18	42	42	**45**	8693	1.62	43	145	0.97	**45**	41	6.13
9	9	51	18	47	48	**51**	4884	1.99	50	149	1.29	**51**	2	5.53

Table 4: *experiments on small complete grid graphs*

Figure: Reference: Brüggemann and Radzik [1]

Experiments

Algorithms:

- Priority BFS
- Max Priority BFS
- Genetic Algorithm
- Artificial Bee Colony Algorithm

Graphs:

- Random graph
- D-regular graph
- Grid graph
- Planar Graph

An Exact Exponential Algorithm

- MLSP problem can be solved using naive brute force algorithm in $\Omega(2^n)$ time.
- In 2011 Fernau et al. gave an $O(1.8966^n)$ algorithm to solve the MLST problem. [3]
- The algorithm has a worst case lower bound of $\Omega(1.4422^n)$
- Previously best known result was an $O(1.9407^n)$ algorithm.

An Exact Exponential Algorithm

The basic idea of the algorithm is to build a subtree of the graph while repeatedly branching of the leaves until a spanning tree is found. The algorithm maintains 5 disjoint set of vertices.

An Exact Exponential Algorithm

Sets

IN(Internal Nodes): Internal Nodes are nodes that were decided to be an internal node of the spanning tree in the course of the algorithm.

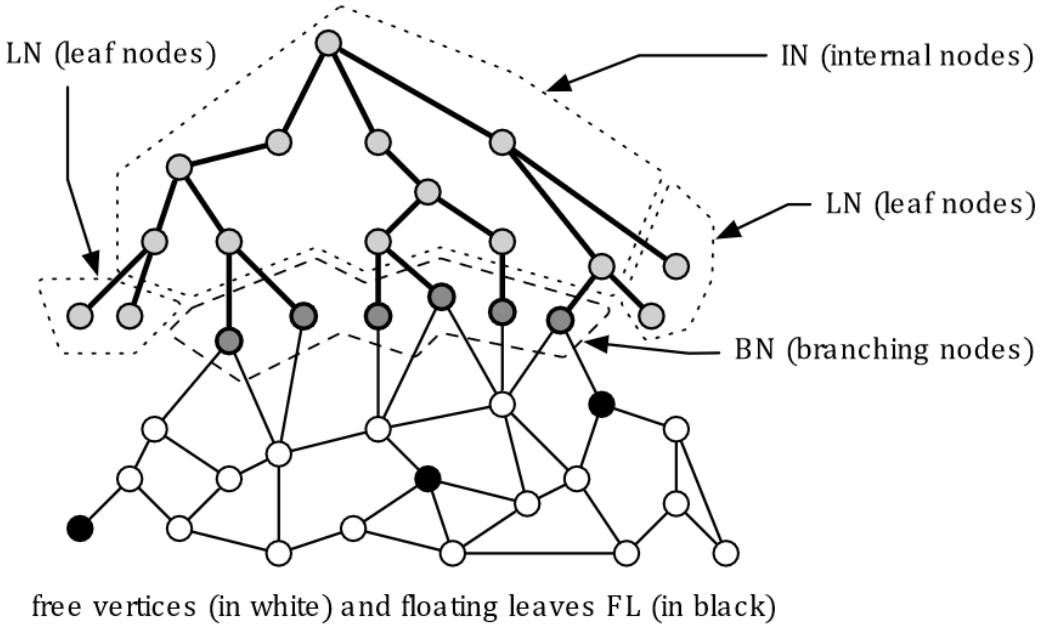


Figure: Internal Nodes

An Exact Exponential Algorithm

Sets

LN (Leaf Nodes) Leaf nodes of the current spanning tree that have no more neighbors to process.

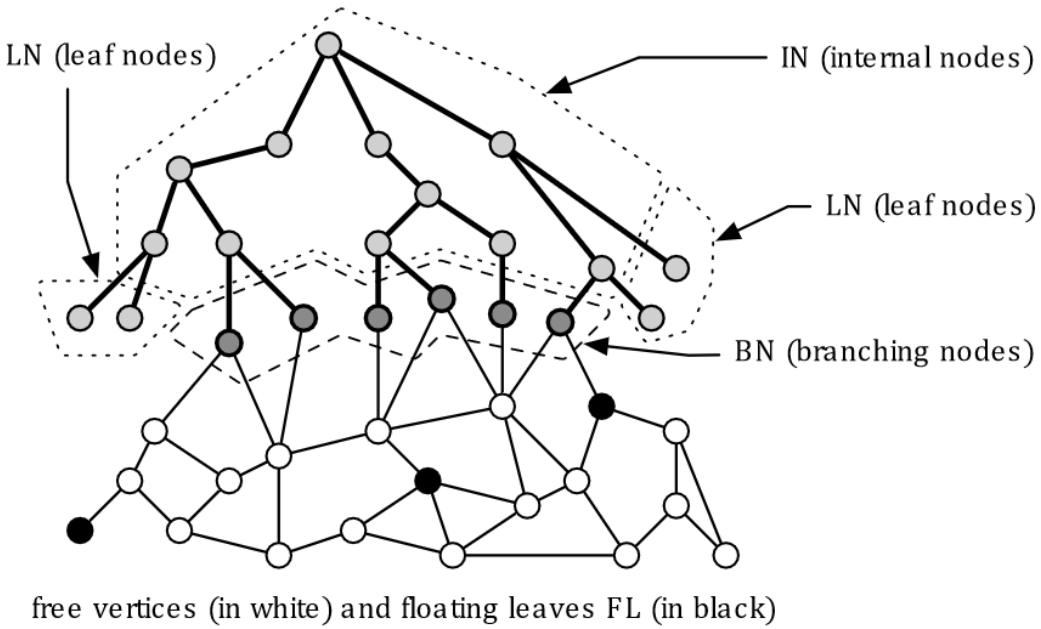


Figure: Internal Nodes

An Exact Exponential Algorithm

Sets

BN (Branching Nodes) Leaf nodes of the current spanning tree that still have some neighbors to process.

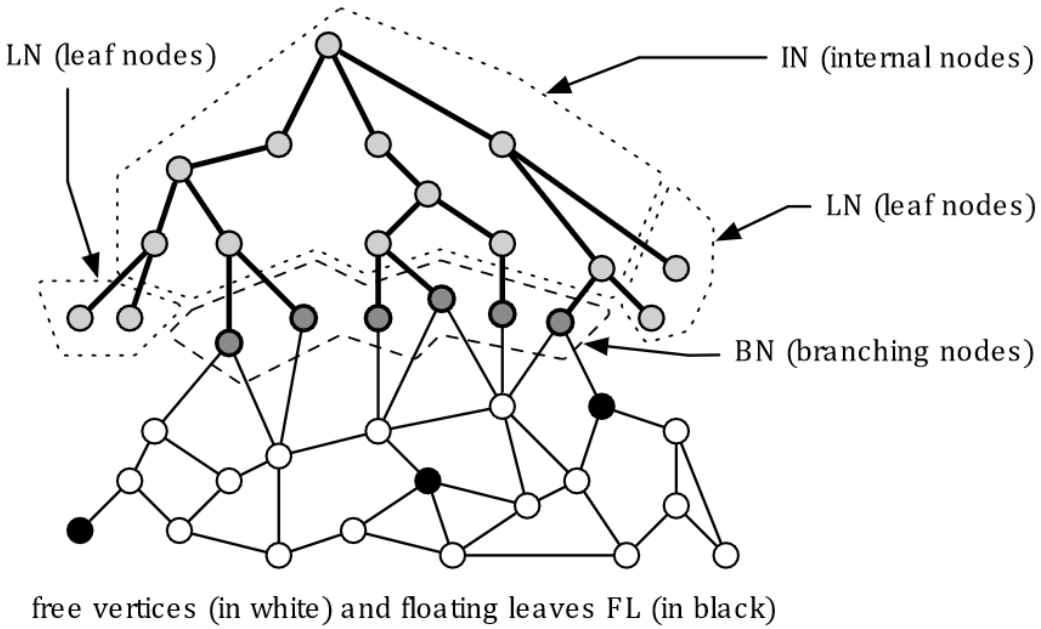


Figure: Internal Nodes

An Exact Exponential Algorithm

Sets

FL (Floating Leaves) Nodes in the original graphs designated to be leaves in the final spanning tree, but has not yet been attached to the current spanning tree

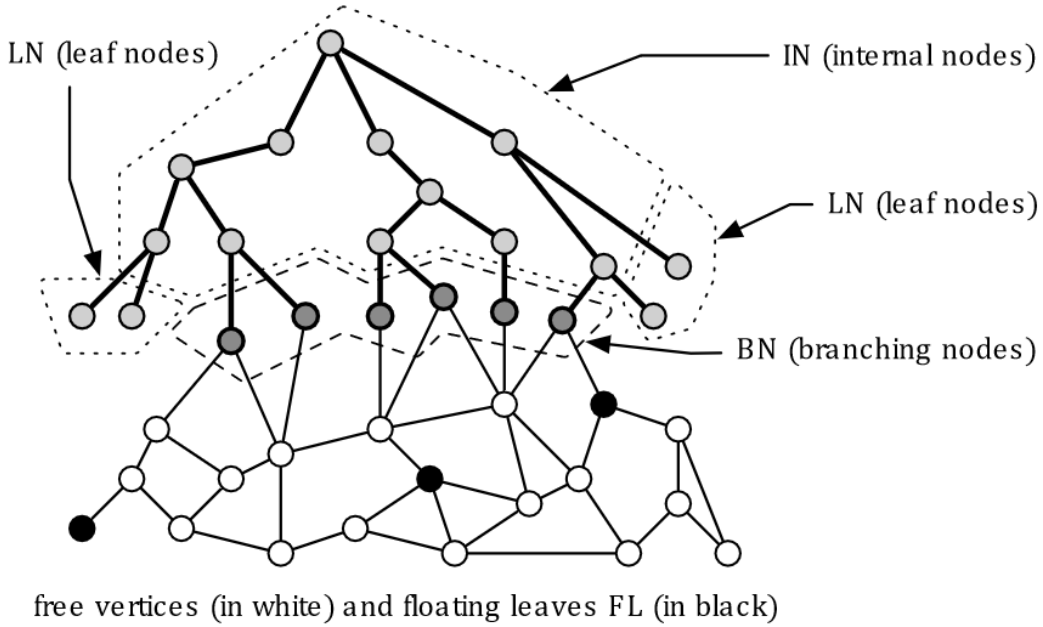


Figure: Internal Nodes

An Exact Exponential Algorithm

Sets

Free (Free Nodes) All other nodes

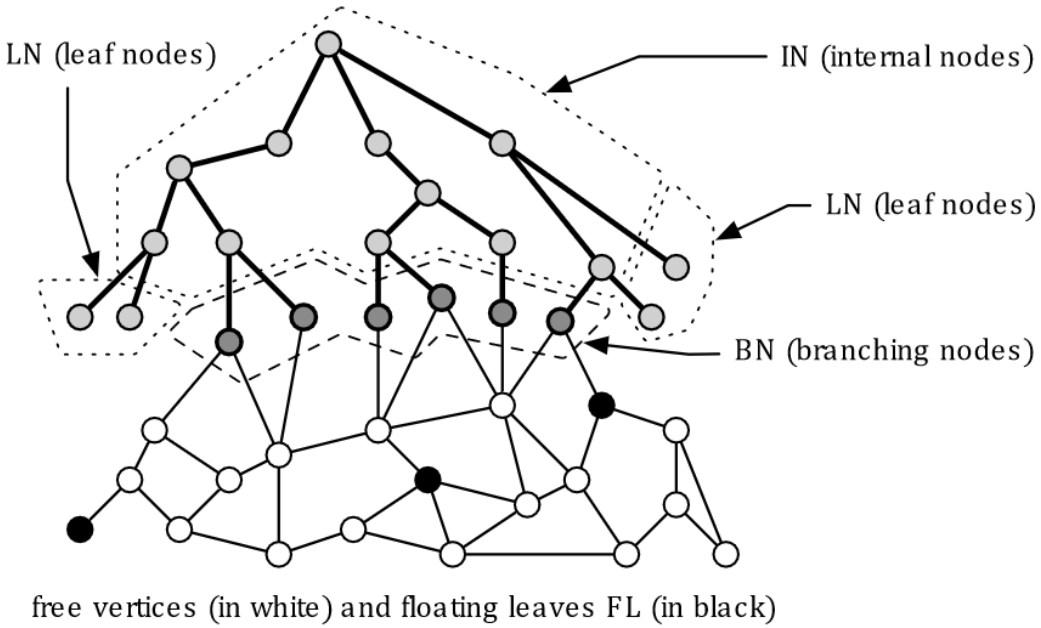


Figure: Internal Nodes

An Exact Exponential Algorithm

The algorithm reduces the input size by applying the following rules recursively.

1. If there exist two adjacent vertices u, v in V such that u, v in FL u, v in BN, then remove the edge $\{u, v\}$ from G.
2. If there exists a node v in BN with $d(v) = 0$, then move v into LN.
3. If there exists a free vertex v with $d(v) = 1$, then move v into FL.
4. If there exists a free vertex v with no neighbors in $Free \cup$ FL, then move v into FL.
5. If there exists a triangle $\{x, y, z\}$ in G with x a free vertex and $d(x) = 2$, then move x into FL.
6. If there exists a node $u \in$ BN which is a cut vertex in G, then apply rule $u \rightarrow$ IN.
7. If there exist two adjacent vertices $u, v \in V$ such that $u \in$ LN and $v \in V \setminus$ IN, then remove the edge $\{u, v\}$ from G.

An Exact Exponential Algorithm

- The algorithm checks if all the node is either IN or
- Whenever the Algorithm decides that some node $x \in V$ becomes an internal node, all of its neighbors are directly attached to the tree.
- Nodes of T belonging to LN will always remain leaves in subsequent calls.
- But the status of the nodes in BN might change

2-approximation Algorithm

- Let G be a simple graph
- $V(G)$ (respectively, $E(G)$) denote the vertex (respectively, edge) set of G.
- $d_G(u)$ with $u \in V(G)$ denote the number of neighbors of u in G.

Notations

- Let T be a spanning tree of G
- For a vertex u of T, let $V_T(u)$ consist of the vertices $v \in V(G) \setminus V(T)$ with $uv \in E(G)$.
- $E_T(u)$ consists of the edges uv of G with $v \in V_T(u)$.
- If $|V_T(u)| = 1$, then let $v_T(u)$ denote the vertex in $V_T(u)$.

Notations

- $W_2(T)$ consists of each vertex u of T with $|V_T(u)| \geq 2$.
- $W_1(T)$ consists of each vertex u of T with $|V_T(u)| = 1$ and $|V_{T \cup E_T(u)}(v_T(u))| \geq 2$.
- $W_0(T)$ consists of each vertex u of T with $|V_T(u)| = 1$ and $|V_{T \cup E_T(u)}(v_T(u))| \leq 1$.

Algorithm

Algorithm tree(G):
Let T be the initial tree of G consisting of a vertex a of G with $d_G(a) \geq 2$.
Repeat the following steps until $V(T) = V(G)$:
If $W_2(T) \neq \varnothing$, then
let u be an arbitrary vertex in $W_2(T)$
else if $W_1(T) \neq \varnothing$, then
let u be an arbitrary vertex in $W_1(T)$
else
let u be the vertex in $W_0(T)$ that joins $V(T)$ most recently.
Expand T at u by letting $T = T \cup E_T(u)$.
Return T.

Figure:

Algorithm

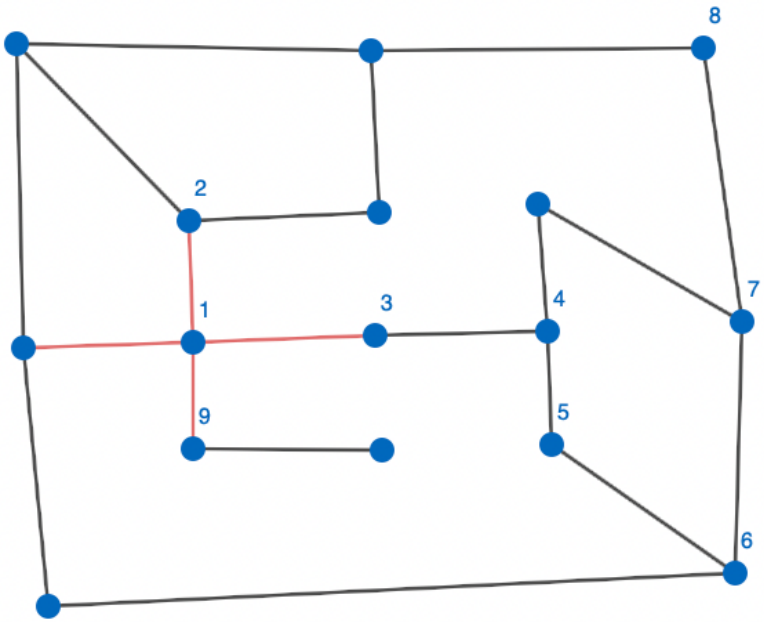


Figure:

Algorithm

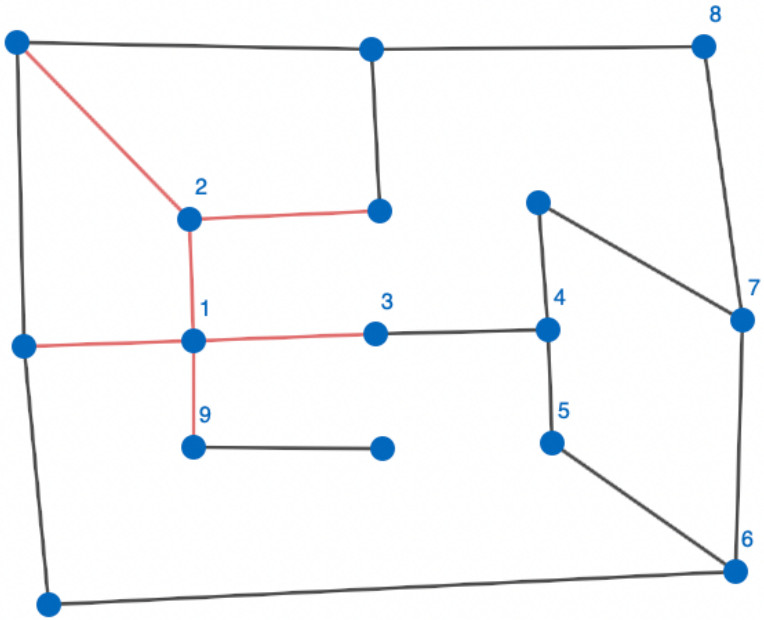


Figure:

Algorithm

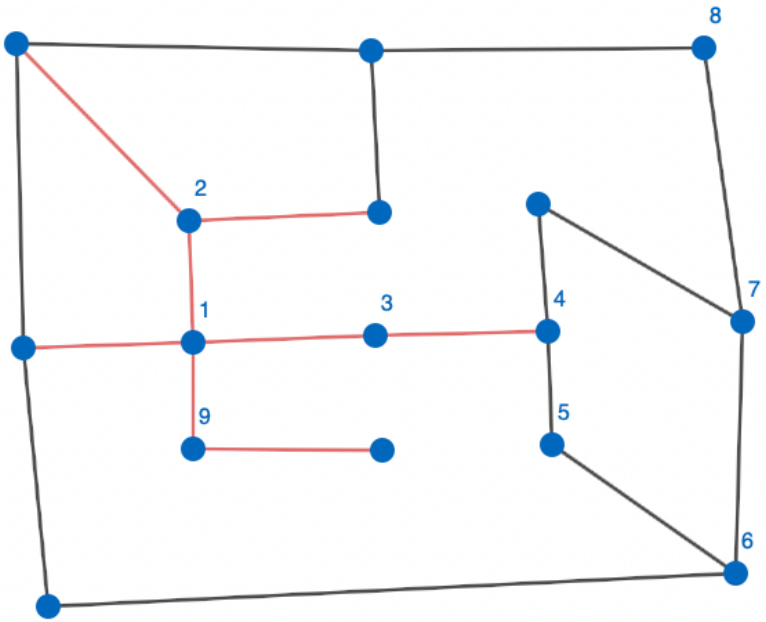


Figure:

Algorithm

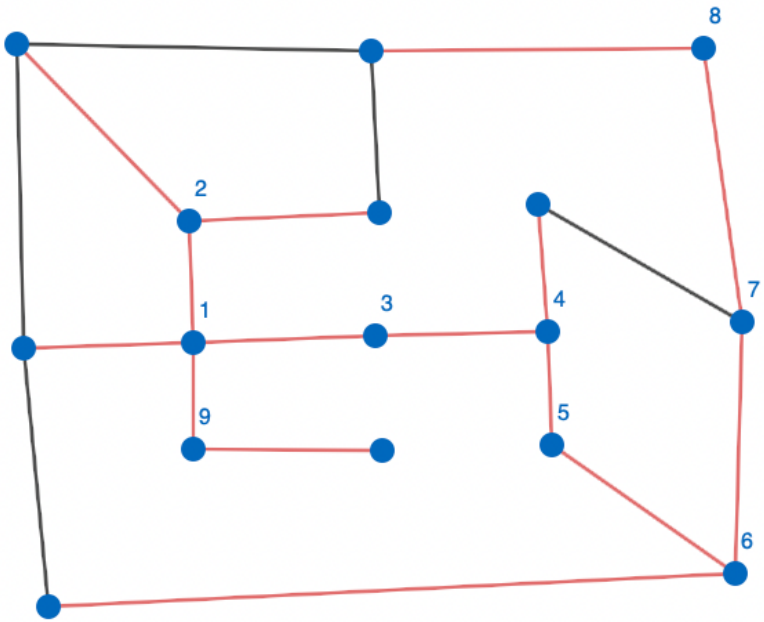


Figure:

Implementation

- We maintain 3 linked lists corresponding to $W2$, $W1$, $W0$.
- We will also maintain an array which will maintain the current state of each vertex(i.e. whether the vertex is an internal node, a leaf or is not added yet)
- When picking up from a linked list, we will pick up its last element and check whether its state matches its container.
- When a vertex is expanded to, we will update the state of all of its neighbours and itself.
- This procedure results in linear time and memory.

References I

[1] Christian Brüggemann and Tomasz Radzik. “Development and Experimental Comparison of Exact and Heuristic Algorithms for the Maximum-Leaf Spanning Tree Problem Technical Report TR-09-08”. In: ().

[2] Jean Daligault et al. “FPT algorithms and kernels for the Directed k-Leaf problem”. In: *Journal of Computer and System Sciences* 76.2 (2010), pp. 144–152. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2009.06.005. URL: https://www.sciencedirect.com/science/article/pii/S0022000009000592.

References II

[3] Henning Fernau et al. “An exact algorithm for the Maximum Leaf Spanning Tree problem”. In: *Theoretical Computer Science* 412.45 (2011), pp. 6290–6302. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2011.07.011. URL: https://www.sciencedirect.com/science/article/pii/S0304397511006219.

[4] Fedor Fomin, Fabrizio Grandoni, and Dieter Kratsch. “Solving Connected Dominating Set Faster Than 2n”. In: *Algorithmica (New York)* 52 (Oct. 2008), pp. 153–166. DOI: 10.1007/s00453-007-9145-z.

[5] Giulia Galbiati, Francesco Maffioli, and Angelo Morzenti. “A short note on the approximability of the maximum leaves spanning tree problem”. In: *Information Processing Letters* 52.1 (1994), pp. 45–49.

References III

[6] Amir Masoud Hosseinmardi, Hamid Farvaresh, and Isa Nakhai Kamalabadi. “A new formulation and algorithm for the maximum leaf spanning tree problem with an application in forest fire detection”. In: *Engineering Optimization* (2022).

[7] Joachim Kneis, Alexander Langer, and Peter Rossmanith. “A New Algorithm for Finding Trees with Many Leaves”. In: *Algorithmica* 61 (2008), pp. 882–897. URL: https://api.semanticscholar.org/CorpusID:2579003.

[8] I-Cheng Liao and Hsueh-I Lu. “A Simple 2-Approximation for Maximum-Leaf Spanning Tree”. In: *International Journal of Foundations of Computer Science* (2023), pp. 1–11.

References IV

[9] Hsueh-I Lu and R Ravi. “Approximating Maximum Leaf Spanning Trees in Almost Linear Time”. In: *Journal of Algorithms* 29.1 (1998), pp. 132–141. ISSN: 0196-6774. DOI: https://doi.org/10.1006/jagm.1998.0944. URL: https://www.sciencedirect.com/science/article/pii/S0196677498909440.

[10] Hsueh-I Lu and R Ravi. “The power of local optimization: Approximation algorithms for maximum-leaf spanning tree (DRAFT)”. In: (1996), pp. 533–542.

[11] Roberto Solis-Oba, Paul Bonsma, and Stefanie Lowski. “A 2-Approximation Algorithm for Finding a Spanning Tree with Maximum Number of Leaves”. In: *Algorithmica* 77.2 (2017), pp. 374–388. ISSN: 1432-0541. DOI: 10.1007/s00453-015-0080-0. URL: https://doi.org/10.1007/s00453-015-0080-0.

Acknowledgements

- Sheikh Azizul Hakim, Lecturer, CSE, BUET

The End

Questions? Comments?