Spanning Tree Problem With Constraint on the Number of Leaves

17050{02,17,66,92,94}
Department of CSE, BUET

February 23, 2023

Outline

Spanning Tree

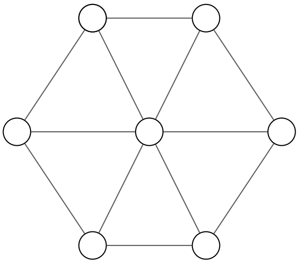


Figure: The Graph W_7

Spanning Tree

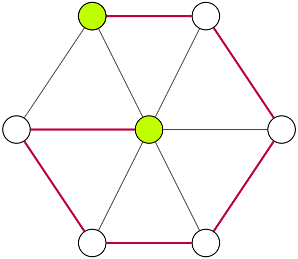


Figure: A Spanning Tree of W_7 with 2 leaves

Spanning Tree

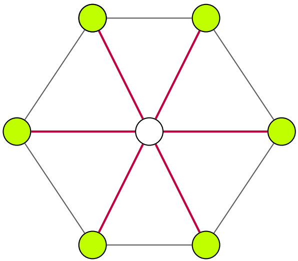


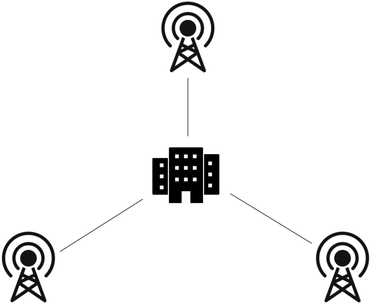
Figure: A Spanning Tree of W_7 with 6 leaves

Outline

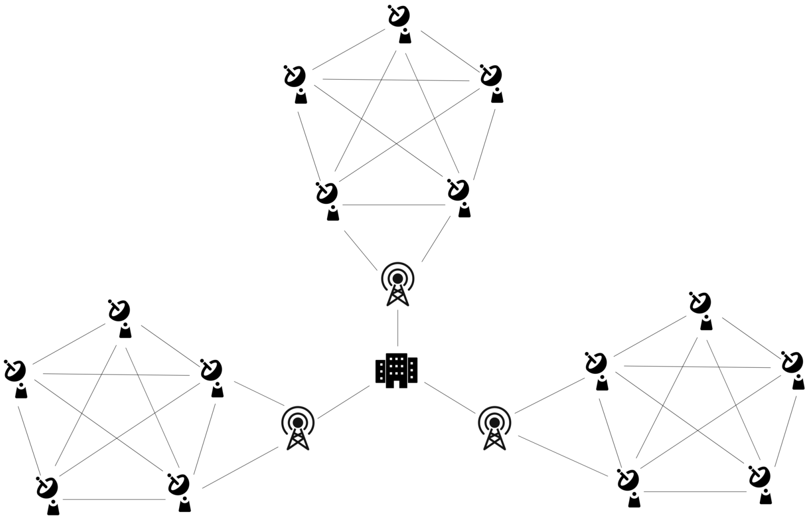
Motivation

Let's take an example.

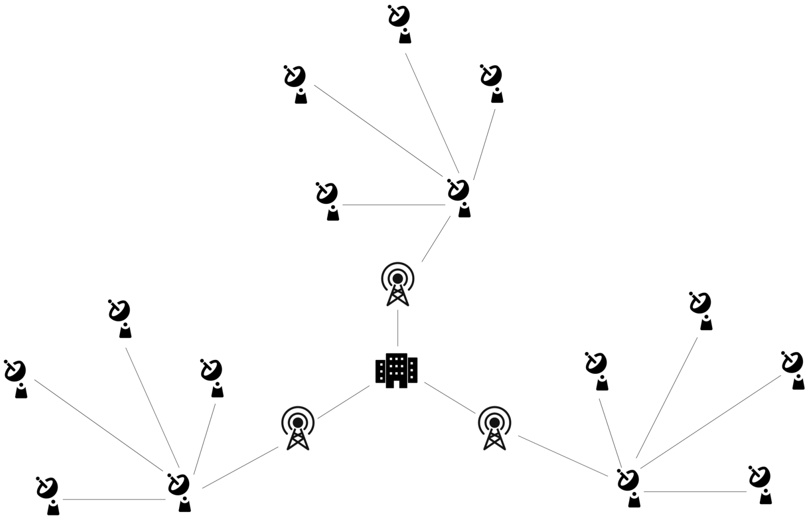
Motivation



Motivation



Motivation



Motivation

This is a Maximum Leaf Spanning Tree problem.

Problems: Optimization Versions

Minimum Leaf Spanning Tree Problem

Given a connected graph G, find a spanning tree of G with the **minimum** number of leaves.

Maximum Leaf Spanning Tree Problem

Given a connected graph G, find a spanning tree of G with the **maximum** number of leaves.

Problems: Decision Versions

Minimum Leaf Spanning Tree Problem (MinLST)

Given a connected graph G and an integer k, does G have a spanning tree with **at most** k leaves?

Maximum Leaf Spanning Tree Problem (MaxLST)

Given a connected graph G and an integer k, does G have a spanning tree with **at least** k leaves?

Outline

Applications - MaxLST

Forest Fire Detection using MaxLST (Farvaresh et al. (2022))

As a realistic application of MaxLST, a forest fire detection system was developed and successfully simulated for the Iranian Arasbaran forest using a wireless sensor network (WSN).

Applications - MaxLST

Broadcasting Network

In a broadcasting network, sometimes the number of broadcasting nodes is required to be minimized. The solution is to find a spanning tree with many leaves.

Applications - MaxLST

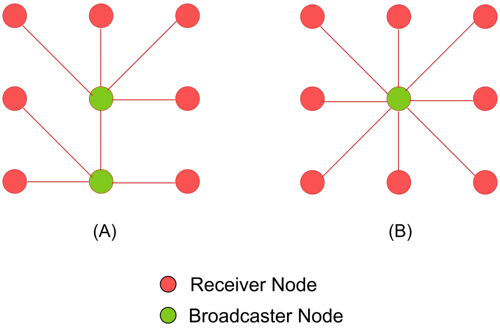


Figure: Maximum number of leaves in a broadcasting network

Applications - MaxLST

Distribution Networks

In any kind of distribution network, Max-LST helps to minimize the delay in the distribution process and identify distributor nodes.

Applications - MaxLST

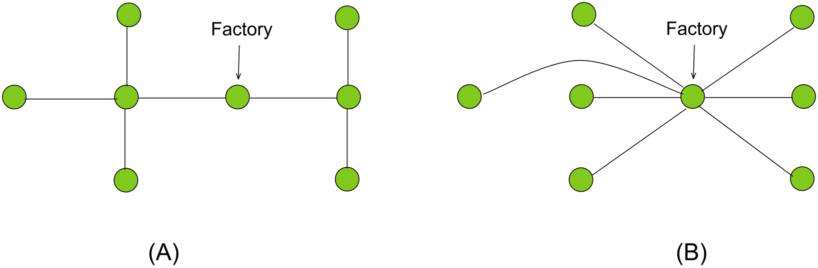


Figure: A distribution network

Applications - MinLST

Security of Private Systems

In private systems, the leaf nodes (network endpoints) are more prone to attacks. In order to secure the system, the number of leaf nodes is minimized, which is done by Min-LST.

Applications - MinLST

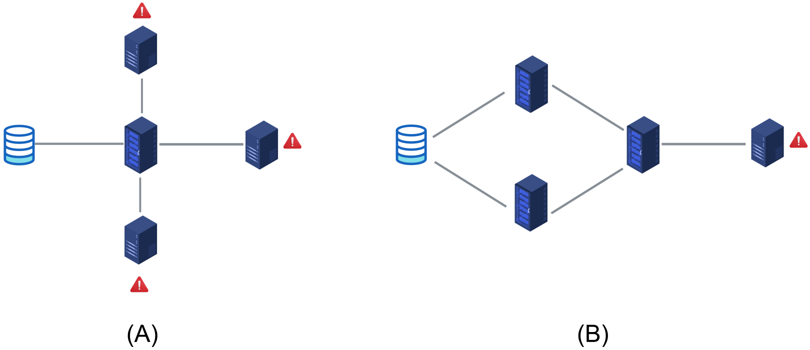


Figure: A private system with vulnerable endpoints

Applications - MinLST

Database Indexing (Demers et al. 1998, Oracle Corp)

In a database system, a graph is built based on the combination of different table columns. A Min-LST for the graph is found and indices are created for the table based on it. The leaves of the tree correspond to the columns that are to be indexed.

Applications - MinLST

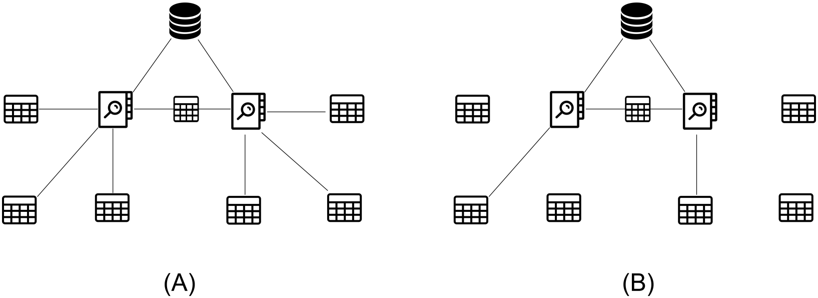


Figure: Database indexing system using MinLST

Outline

MaxLST

1. An $\mathcal{O}^*(1.8966^n)$ exact algorithm by Fernau et al. (2009)
2. A 2-approximation algorithm by Solis-Oba et al. (2017)
3. MaxLST is APX-hard, and thus no PTAS exists unless $\mathcal{P} = \mathcal{NP}$. (Galbiati et al., 1994)
4. An $\mathcal{O}^*(3.4575^k n^{\mathcal{O}(1)})$ FPT algorithm by Raible and Fernau (2010)
5. Genetic Algorithm Based Heuristic Approach by Farvaresh et al. (2022)
6. A modified version of the problem, where a subset of vertices is specified instead of an integer k, is soluble. (Rahman and Kaykobad, 2005)

MinLST

1. No result on approximability exists. (Watel 2020)
2. Heuristic-based solutions exist (e.g. Scatter search by Kardam et al. (2022))
3. An $\mathcal{O}^*(1.8916^n)$ exact algorithm by Fernau et al. (2009) for graph with $\Delta \leq 3$.
4. Algorithm that finds a spanning tree of G with $\geq k$ internal vertices $(2 < k < n-2)$ if it exists in $\mathcal{O}^*(2^{3.5klogk})$.

Outline

Exact Algorithm

- An exact algorithm with running time $O(1.8966^n)$
- Proposed by Fernau et al. 2011 [1]

---

[1] Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, Daniel Raible, Peter Rossmanith, *An exact algorithm for the Maximum Leaf Spanning Tree problem*, Theoretical Computer Science, Volume 412, Issue 45, 2011, Pages 6290-6302, ISSN 0304-3975, `https://doi.org/10.1016/j.tcs.2011.07.011`

Exact Algorithm

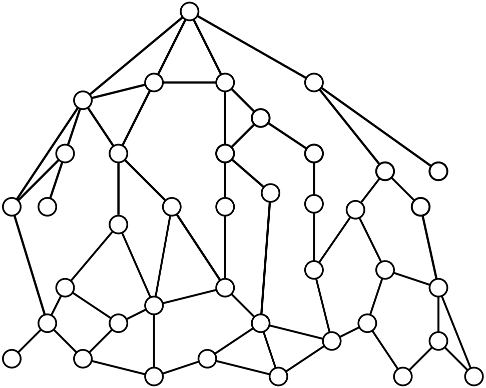


Figure: An example of a graph G

Exact Algorithm

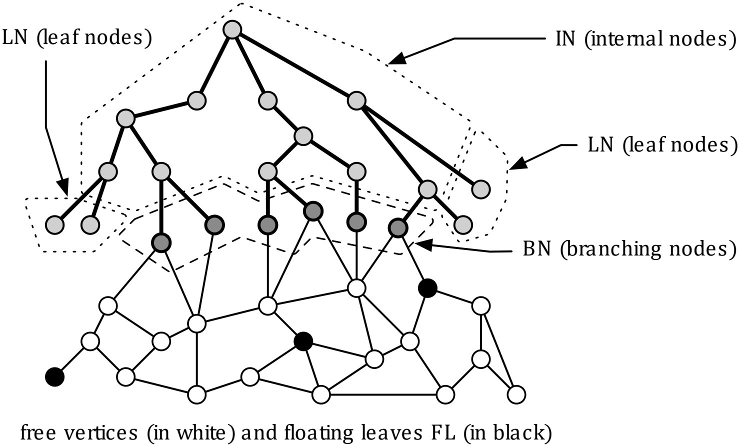


Figure: G with a subtree with corresponding sets of vertices IN, BN, LN (describing the subtree), as well as FL and Free.

Reduction Rules: R1

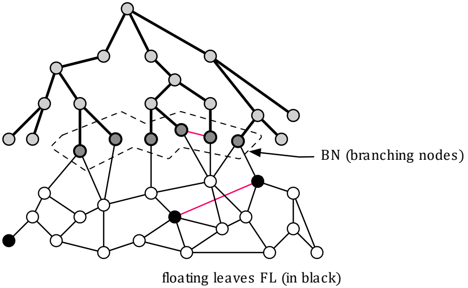


Figure: Reduction Rules: R1

R1

If there exist two adjacent vertices $u, v \in V$ such that $u, v \in \mathrm{FL}$ or $u, v \in \mathrm{BN}$, then remove the edge $\{u, v\}$ from G.

Reduction Rules: R1

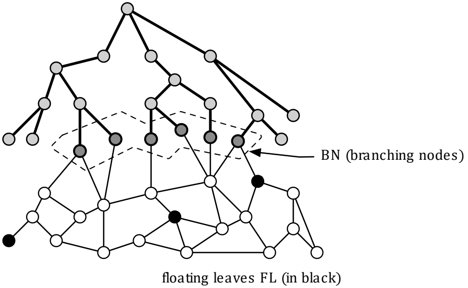


Figure: Reduction Rules: R1

R1

If there exist two adjacent vertices $u, v \in V$ such that $u, v \in \mathrm{FL}$ or $u, v \in \mathrm{BN}$, then remove the edge $\{u, v\}$ from G.

Reduction Rules: R2

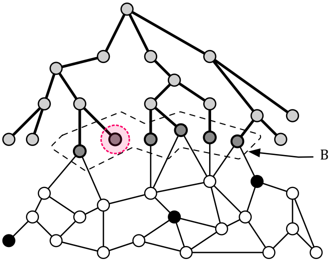


Figure: Reduction Rules: R2

R2

If there exists a node $v \in \text{BN}$ with $d(v) = 0$, then move v into LN.

Reduction Rules: R3

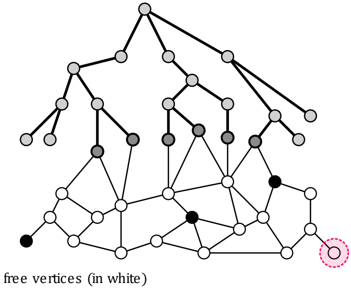


Figure: Reduction Rules: R3

R3

If there exists a free vertex v with $d(v) = 1$, then move v into FL.

Reduction Rules: R4

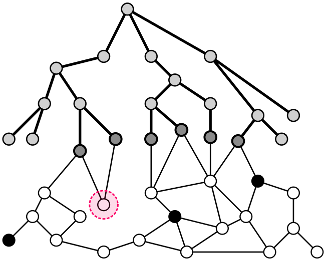


Figure: Reduction Rules: R4

R4

If there exists a free vertex v with no neighbors in $\mathrm{Free} \cup \mathrm{FL}$, then move v into FL.

Reduction Rules: R5

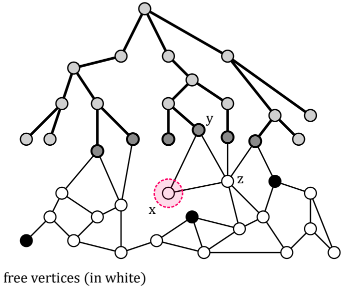


Figure: Reduction Rules: R5

R5

If there exists a triangle $\{x, y, z\}$ in G with x a free vertex and $d(x) = 2$, then move x into FL.

Reduction Rules: R6

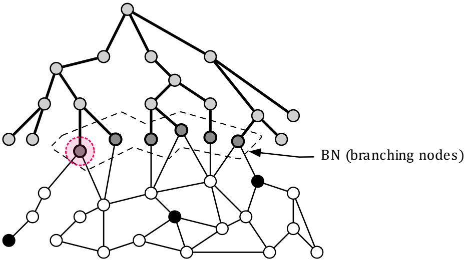


Figure: Reduction Rules: R6

R6

If there exists a node $u \in$ BN which is a cut vertex in G, then apply rule $u \to$ IN.

Reduction Rules: R7

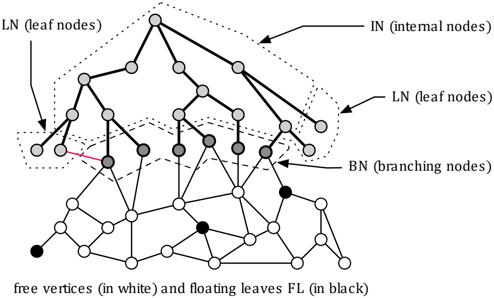


Figure: Reduction Rules: R7

R7

If there exist two adjacent vertices $u, v \in V$ such that $u \in \mathrm{LN}$ and $v \in V \setminus \mathrm{IN}$, then remove the edge $\{u, v\}$ from G.

Reduction Rules: R7

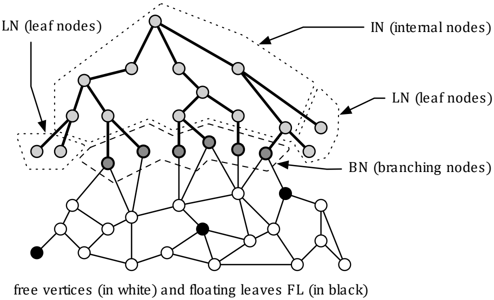


Figure: Reduction Rules: R7

R7

If there exist two adjacent vertices $u, v \in V$ such that $u \in \text{LN}$ and $v \in V \setminus \text{IN}$, then remove the edge $\{u, v\}$ from G.

Algorithm

Data: A graph $G = (V, E), IN, BN, LN, FL \subseteq V$
apply all reduction rules possible;

Algorithm

Data: A graph $G = (V, E), IN, BN, LN, FL \subseteq V$
apply all reduction rules possible;
if *there is some unreachable* v *in* $FL \cup free$ **then**
  return 0

Algorithm

Data: A graph $G = (V, E), IN, BN, LN, FL \subseteq V$
apply all reduction rules possible;
if *there is some unreachable v in* $FL \cup free$ **then**
  return 0
if $V = IN \cup LN$ **then**
  return $|LN|$

Algorithm

Data: A graph $G = (V, E), IN, BN, LN, FL \subseteq V$
apply all reduction rules possible;
if *there is some unreachable* v *in* $FL \cup free$ **then**
 return 0
if $V = IN \cup LN$ **then**
 return $|LN|$
Take the vertex $v \in BN$ of the maximum degree;

Algorithm

Data: A graph $G = (V, E), IN, BN, LN, FL \subseteq V$
apply all reduction rules possible;
if *there is some unreachable* v *in* $FL \cup free$ **then**
  return 0
if $V = IN \cup LN$ **then**
  return $|LN|$
Take the vertex $v \in BN$ of the maximum degree;
if $deg(v) \geq 3$ *or* $(deg(v) == 2$ *and* $N_{FL}(v) \neq \emptyset)$ **then**
  return $\max((v \to LN),\ (v \to IN))$

Algorithm - Continued

if $deg(v) == 2$ **then**
    Let x_1 and x_2 be two neighbors of v such that $deg(x_1) \leq deg(x_2)$;
    if $deg(x_1) == 2$ **then**
        Let $z = N(x_1) - v$;
        if $z \in Free$ **then**
            return $max(\ v \rightarrow LN,$
            $v, x_1 \rightarrow IN,$
            $v \rightarrow IN, x_1 \rightarrow LN\ \ )$
        end
    end

Algorithm - Full

Algorithm $\mathcal{M}$
Input: A graph $G = (V, E)$, IN, BN, LN, FL $\subseteq V$

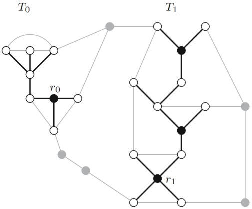
Reduce G according to the reduction rules.
if there is some unreachable $v \in$ Free $\cup$ FL **then** return 0
if $V =$ IN $\cup$ LN **then** return |LN|
Choose a vertex $v \in$ BN of maximum degree.
if $d(v) \geq 3$ **or** $(d(v) = 2$ and $N_{\text{FL}}(v) \neq \emptyset)$ **then**
  $\langle v \to \text{LN} \mid\mid v \to \text{IN} \rangle$ (B1)
else if $d(v) = 2$ **then**
  Let $\{x_1, x_2\} = N_{\text{Free}}(v)$ such that $d(x_1) \leq d(x_2)$.
  if $d(x_1) = 2$ **then**
    Let $\{z\} = N(x_1) \setminus \{v\}$
    if $z \in$ Free **then**
      $\langle v \to \text{LN} \mid\mid v \to \text{IN}, x_1 \to \text{IN} \mid\mid v \to \text{IN}, x_1 \to \text{LN} \rangle$ (B2)
    else if $z \in$ FL **then** $\langle v \to \text{IN} \rangle$
  else if $(N(x_1) \cap N(x_2)) \setminus \text{FL} = \{v\}$ **and** $\forall z \in (N_{\text{FL}}(x_1) \cap N_{\text{FL}}(x_2))$,
    $d(z) \geq 3$ **then** (B3)
    $\langle v \to \text{LN} \mid\mid v \to \text{IN}, x_1 \to \text{IN} \mid\mid v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{IN} \mid\mid$
    $v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{LN}, \quad N_{\text{Free}}(\{x_1, x_2\}) \to \text{FL}, N_{\text{BN}}(\{x_1, x_2\}) \setminus \{v\} \to \text{LN} \rangle$
  else $\langle v \to \text{LN} \mid\mid v \to \text{IN}, x_1 \to \text{IN} \mid\mid v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{IN} \rangle$ (B4)
else if $d(v) = 1$ **then**
  Let $P = (v = v_0, v_1, \ldots, v_k)$ be a maximum path such that
    $d(v_i) = 2, 1 \leq i \leq k, v_1, \ldots, v_k \in$ Free.
  Let $z \in N(v_k) \setminus V(P)$.
  if $z \in$ FL **and** $d(z) = 1$ **then** $\langle v_0, \ldots, v_k \to \text{IN}, z \to \text{LN} \rangle$
  else if $z \in$ FL **and** $d(z) > 1$ **then** $\langle v_0, \ldots, v_{k-1} \to \text{IN}, v_k \to \text{LN} \rangle$
  else if $z \in$ BN **then** $\langle v \to \text{LN} \rangle$
  else if $z \in$ Free **then** $\langle v_0, \ldots, v_k \to \text{IN}, z \to \text{IN} \mid\mid v \to \text{LN} \rangle$ (B5)

Figure: Algorithm $\mathcal{M}$ for Maximum Leaf Spanning Tree

Outline

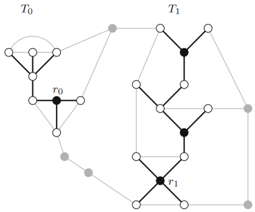
2-Approximation Algorithm

- G is a simple, undirected graph.



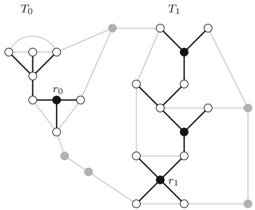
2-Approximation Algorithm

- G is a simple, undirected graph.
- We start with an empty subgraph F, and incrementally expand F until it forms a spanning tree of G.



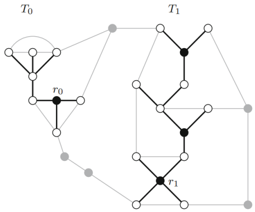
2-Approximation Algorithm

- G is a simple, undirected graph.
- We start with an empty subgraph F, and incrementally expand F until it forms a spanning tree of G.
- F has all the vertices of G.



2-Approximation Algorithm

- G is a simple, undirected graph.
- We start with an empty subgraph F, and incrementally expand F until it forms a spanning tree of G.
- F has all the vertices of G.
- The set of edges of F is a subset of the set of edges of G.



Expanding a Vertex in a Graph

The operation of expanding a vertex $v \in V(G)$ consists of the following steps:

1. If $v \notin V(F)$, add v to F.
2. For every $w \in N_G(v) \setminus V(F)$, add vertex w and edge vw to F.

Note that:

- $N_G(v)$ denotes the neighborhood of v in G, i.e., the set of vertices adjacent to v.
- $V(G) \setminus S$ denotes the set of vertices in G that are not in S.

Expansion Rules

- We use four expansion rules, where the given order defines the priorities.
- for any $i < j$, if Rule i can be applied, then Rule j may not be applied.

Expansion Rules

Rule 1

If F contains a vertex v with at least two neighbors in $\overline{V(F)}$, then expand v.

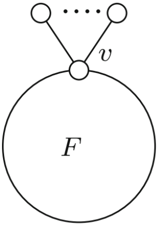


Figure: Rule 1

Expansion Rules

Rule 2

If F contains a vertex v with only one neighbor w in $\overline{V(F)}$, which in turn has at least three neighbors in $\overline{V(F)}$, then first expand v, and next expand w.

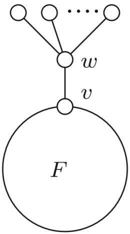


Figure: Rule 2

Expansion Rules

Rule 3

If F contains a vertex v with only one neighbor w in $\overline{V(F)}$, which in turn has two neighbors in $\overline{V(F)}$, then first expand v, and next expand w.

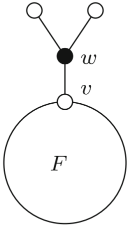


Figure: Rule 3

Expansion Rules

Rule 4

If $\overline{V(F)}$ contains a vertex v with at least three neighbors in $\overline{V(F)}$, then expand v.

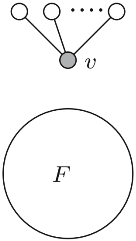


Figure: Rule 4

Expansion Rules

- Only Rule 4 increases the number of components of F.

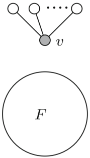


Figure: Rule 4

Expansion Rules

- Only Rule 4 increases the number of components of F.
- The priorities ensure that when Rule 4 is applied, Rules 1, 2, and 3 cannot be applied to any component of F.

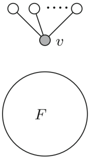


Figure: Rule 4

Expansion Rules

- Only Rule 4 increases the number of components of F.
- The priorities ensure that when Rule 4 is applied, Rules 1, 2, and 3 cannot be applied to any component of F.
- Once a new component is introduced, existing components of F will not be modified anymore.

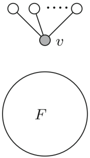


Figure: Rule 4

Finally

- Choose an edge of G between different components of F, and add it to F.

Algorithm

Data: A connected simple undirected graph G with maximum degree at least 3.
Result: A spanning tree T^* of G.
// First phase:
F := the empty graph.
while *one of the Expansion Rules 1-4 can be applied to* F **do**
  Apply the expansion rule with the highest priority to F.
end
$F^* := F$
// Second phase:
while F *is not spanning* **do**
  Choose a vertex $v \in V(F)$ with $N_G(v) \setminus V(F) \neq \emptyset$, and expand v.
end
while F *is not connected* **do**
  Choose an edge of G between different components of F, and add it to F.
end
$T^* := F$
return T^*

Outline

Exact Algorithm

A branching algorithm whose time complexity is measured by **measure and conquer**

Exact Algorithm

A branching algorithm whose time complexity is measured by **measure and conquer**

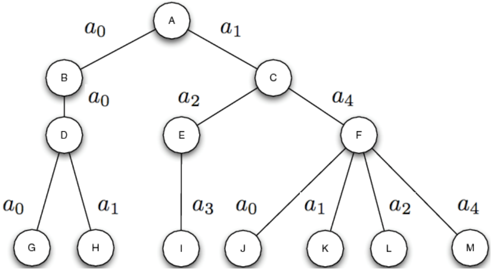


Figure: A search tree

Exact Algorithm

Measure

$\mu(I) = \sum_{i=1}^{n-1} \epsilon_i^{BN}|BN_i| + \sum_{i=2}^{n-1} \epsilon_i^{FL}|FL_i| + \sum_{i=2}^{n-1} \epsilon_i^{Free}|Free_i|$

For $X \in \{BN, FL, Free\}$

- X_i denoted the vertices in X of degree i.
- $\epsilon_i^X \in [0, 1)$ are predefined constants.

Exact Algorithm

Data: A graph $G = (V, E), IN, BN, LN, FL \subseteq V$
apply all reduction rules possible;
if *there is some unreachable* v *in* $FL \cup free$ **then**
  return 0
if $V = IN \cup LN$ **then**
  return $|LN|$
Take the vertex $v \in BN$ of the maximum degree;
if $deg(v) \geq 3$ *or* $(deg(v) == 2$ *and* $N_{FL}(v) \neq \emptyset)$ **then**
  return $\max((v \to LN),\ (v \to IN))$

Change of Measure: $v \to LN$

- v becomes a leaf node

Change of Measure: $v \to LN$

- v becomes a leaf node
- Each vertex in $N_{Free \cup FL}(v)$ loses degree by 1 by (R7).

Change of Measure: $v \to LN$

- v becomes a leaf node
- Each vertex in $N_{Free \cup FL}(v)$ loses degree by 1 by (R7).
- A decrease inmeasure by at least

$$\Delta_1 = \epsilon^{BN}_{deg(v)} + \sum_{x \in N_{Free}(v)} (\epsilon^{Free}_{deg(x)} - \epsilon^{Free}_{deg(x)-1}) + \sum_{x \in N_{Free}(v)} (\epsilon^{FL}_{deg(x)} - \epsilon^{FL}_{deg(x)-1})$$

Change of Measure: $v \to IN$

- v becomes an internal node

Change of Measure: $v \to IN$

- v becomes an internal node
- Each vertex in $N_{Free}(v)$ becomes branching nodes

Change of Measure: $v \rightarrow IN$

- v becomes an internal node
- Each vertex in $N_{Free}(v)$ becomes branching nodes
- A decrease inmeasure by at least

$$\begin{aligned}\Delta_2 = \epsilon^{BN}_{deg(v)} &+ \sum_{x \in N_{Free}(v)} (\epsilon^{Free}_{deg(x)} - \epsilon^{BN}_{deg(x)-1}) \\ &+ \sum_{x \in N_{Free}(v)} (\epsilon^{FL}_{deg(x)})\end{aligned}$$

Measure and Conquer - Continued

$$T(x) \leq T(x - \Delta_1) + T(x - \Delta_2)$$

A case-by-case analysis shows, for the worst case,

$$(\Delta_1, \Delta_2) = (1.538324, 0.730838)$$

and we find

$$T(x) = o(1.8966^n)$$

.

Exact Algorithm - Runtime

Theorem

The given algorithm solves MaxLST in $\mathcal{O}(1.8966^n)$ time.

Exact Algorithm - Runtime

Theorem

The given algorithm solves MaxLST in $\mathcal{O}(1.8966^n)$ time.

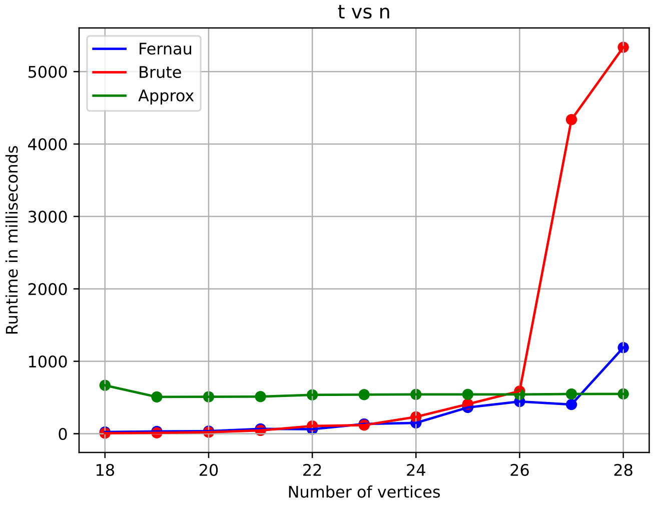
Theorem

The given algorithm solves MaxLST in $\Omega(1.4422^n)$ time.

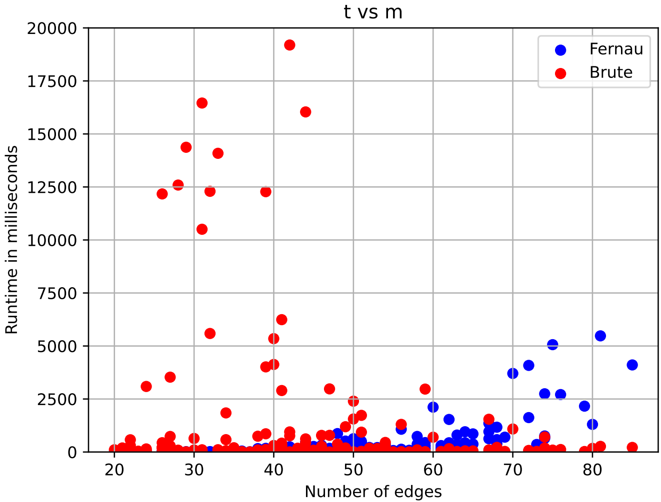
2-Approximation Algorithm

Linear time implementation exists

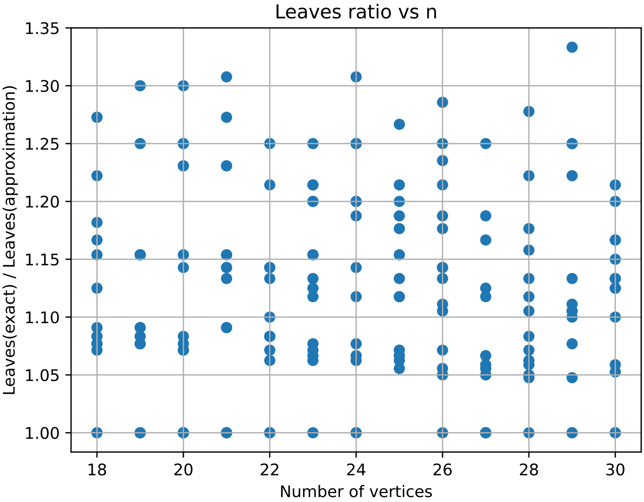
Runtime vs n


t vs n
Fernau
Brute
Approx
Runtime in milliseconds
Number of vertices
0
1000
2000
3000
4000
5000
18
20
22
24
26
28

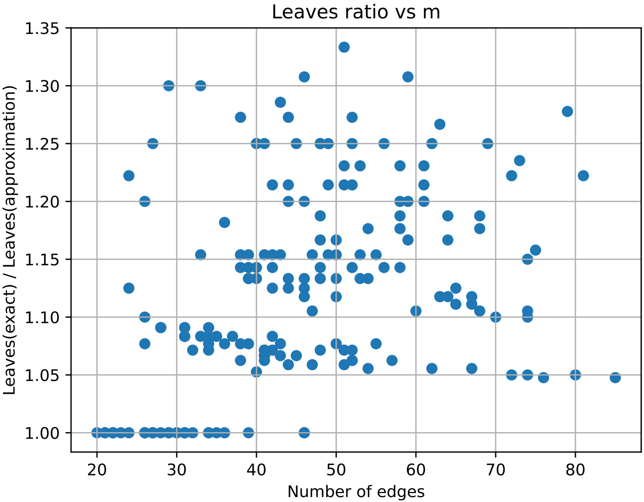
Runtime vs m


t vs m
20000
17500
15000
12500
10000
7500
5000
2500
0
Runtime in milliseconds
20
30
40
50
60
70
80
Number of edges
Fernau
Brute

Leaves ratio vs n



Leaves ratio vs m



Outline

Possible Future Direction

1. The runtime analysis of the exact algorithm by Fernau et al needs to be improved.

Possible Future Direction

1. The runtime analysis of the exact algorithm by Fernau et al needs to be improved.
2. While a plethora of studies exists on MaxLST, there has been no result on exact or approximation algorithms for MinLST.

Thank You