CS301 - Fall 2019-2020

A.Öykü Erçin, Berk Tahtacı, Ece Alptekin, Kubilay Kitapçıoğlu, Şansal Bakkal

1- Problem Description

The Maximum Leaf Spanning Tree (MLST) problem, which asks to find for a given graph, a spanning tree with as many leaves as possible. In our task we take input as n-node undirected graph G(V, E); positive integer k ≤ n and it asks that does G have a spanning tree in which at least k nodes have degree 1.There are real time applications for using Maximum Leaf Spanning Tree like some broadcasting problems in network design ask to minimize the number of broadcasting nodes, which must be connected to a single root. This explains that finding a spanning tree with many leaves and few internal nodes. For that reason, this algorithm should run fast, and we don't know any algorithm to solve this polynomial time. When we analyze it we found that this problem can solvable in exponential time and given solution that it can checked if is it correct or not in exponential time because of that this problem is NP-Complete problem.

Proof of NP- Completeness:

The maximum leaf spanning tree(MLSPT) is equivalent to the minimum connected dominating set(MCDS)

decision version of MCDS is in NP. Similar to proof of NP completeness of dominating set, we perform a reduction from vertex cover(VC), i.e., we prove that $VC \leq_p MCDS$:

Given an instance $(G(V, E), k)$ of VC we construct an instance $(G'(V', E'), k)$ of MCDS as follows: G' contains the complete graph on V, and for each edge $(u, v) \in E$, an *edge vertex* x_{uv} is introduced, along with two extra edges (x_{uv}, u) and (x_{uv}, v). Formally. we have $V' = V \cup \{x_{uv} : (u, v) \in E\}$, $E' = E \cup \{(x_{u,v}, u) : (u, v) \in E\}$. The reduction can obviously be done in polynomial time. Note that any induced subgraph in G' is a connected graph.

Let S be a vertex cover of G. For any $v \in V' \setminus S$, if $v \in V$ then by the definition of vertex cover we know that v dominated by S. If $v = v_{xy}$ is an edge vertex of some edge $(x, y) \in E$, because (x, y) is covered by S, then $x \in S$ or $y \in S$, v_{xy} is dominated by S. Thus S is a connected dominating set of G'.

Suppose that S is a connected dominating set in G'. We first observe that for any edge vertex v_{xy} in, it can only be dominated by x or y. For any edge vertex $v_{xy} \in S$, we replace it with x. The replacement does not increase the cardinality of S and mantains S's property of being a dominating set. Now S contains no edge vertex, so every edge vertex is dominated by S, that is to say, every edge in G has at least one endpoint in S. Therefore, S is a vertex cover for G.

2- Algorithm Description

First of all, we will describe the leafy subtrees and leafy forest which we used in our algorithm.

Definition: Let T be a subtree of G.

1. We say T is leafy if V3(T) is not empty and every node in V2(T) is adjacent in T to exactly two nodes in V3(T). A forest F of G is leafy if F is composed of disjoint leafy subtrees of G. We say F is maximally leafy if F is not a subgraph of any other leafy forest of G.

At least ⅓ of the nodes in a leafy subtree T are leaves of T.

Proof:

Since T is leafy, each node in V2(T) must be adjacent in T to exactly two nodes in V3(T). Consider the induced subtree on V2(T) , where each node of degree 2 is replaced by an edge. The number of edges in this subtree is one less than the number of nodes. Therefore |V2(T)| <= |V3(T)|. So;

$$\begin{aligned} |V(T)| &= |V_1(T)| + |V_2(T)| + |\bar{V}_3(T)| \\ &\leq |V_1(T)| + 2|\bar{V}_3(T)| - 1 \\ &\leq 3|V_1(T)| - 5. \end{aligned}$$

Properties of Maximally Leafy Forests:

Let F be a maximally leafy forest of G. Let T1 ,..., Tk be the disjoint leafy subtrees of F. We use the example in Fig. 1 to illustrate each property. The dark lines in the figure are the edges in the maximally leafy forest F, which is composed of three leafy subtrees T1 , T2 , and T3 .

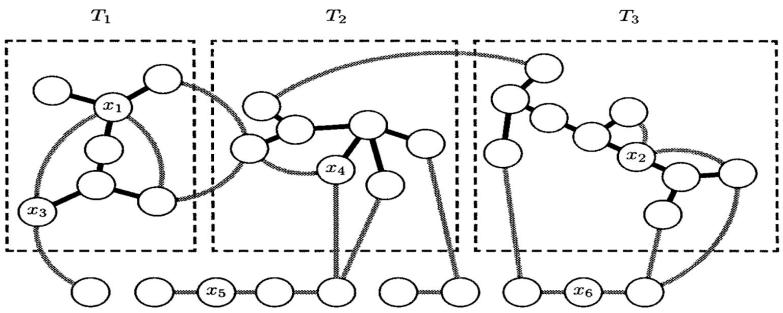


FIG. 1. An example of a maximally leafy forest represented by dark edges. Gray edges are the remaining edges in G.

1. Let w be a node in V2(Ti). Then we cannot be adjacent in G to any node not in Ti. (Nodes x1 and x2 are two examples of w. Then F would not be maximal since x1,x5 could be added to F.)

2. Let w be a node in Ti . Let w1 and w2 be two distinct nodes adjacent to w in G. If w1 is not in F, then w2 must be in T . (Nodes x3 is an example of w. If x3 had two neighbors not in F, both these edges could be added to F, contradicting its maximality).

3. Let w be a node in F. If w is adjacent to two distinct nodes not in F, then the degree of w in G is 2. (Nodes x5 and x6 are two examples of w. Note that such nodes are not in F. If the degree of say x5 were greater 5 than 2, then x5 and its three neighbors not in F could be added as an additional star in F, contradicting its maximality again.)

Upper Bound

The crux of the proof of the performance guarantee is an upper bound we derive on the maximum number of leaves in any tree relative to the number of leaves in any maximally leafy forest of the graph.

THEOREM 1. *Let F be a maximally leafy forest of G. Let T be a spanning tree of G such that F is a subgraph of T. Then*

$$|V_1(T)| \geq |V_1(\hat{T})|/3$$

for any spanning tree $\hat{T}$ of G.

Lemma1: Let F be a maximally leafy forest of G that is composed of k disjoint leafy subtrees T1,..., T of G k . Then:

$$|V_1(\hat{T})| \leq |V(F)| - k + 1$$

for any spanning tree T of G .

Lemma2: Let F be a forest of G that has k disjoint non singleton subtrees. Let T be a spanning tree of G such that F is a subgraph of T. Then

$$|V_1(T)| \geq |V_1(F)| - 2(k-1).$$

Proof for Theorem 1: Suppose F has k disjoint leafy subtrees T1 ,..., Tk.

$$\begin{aligned} |V(F)| &= \sum_{i=1}^{k} |V(T_i)| \\ &\leq 3|V_1(F)| - 5k. \end{aligned}$$

$$\begin{aligned} |V_1(\hat{T})| &\leq |V(F)| - k + 1 \\ &\leq 3|V_1(F)| - 6k + 1 \\ &\leq 3(|V_1(T)| + 2(k-1)) - 6k + 1 \\ &\leq 3|V_1(T)|. \end{aligned}$$

THE ALGORITHM

The Pseudocode of our algorithm is:

MAXIMUM LEAF SPANNING TREES

```
MaximallyLeafyForest(G)
1   Let F be an empty set.
2   For every node v in G do
3       S(v) := {v}.
4       d(v) := 0.
5   For every node v in G do
6       S' := ∅.
7       d' := 0.
8       For every node u that is adjacent to v in G do
9           If u ∉ S(v) and S(u) ∉ S' then
10              d' := d' + 1.
11              Insert S(u) into S'.
12      If d(v) + d' ≥ 3 then
13          For every S(u) in S' do
14              Add edge uv to F.
15              Union S(v) and S(u).
16              Update d(u) := d(u) + 1 and d(v) := d(v) + 1.
17  Output F as a maximally leafy forest of G.
```

FIG. 2. The procedure for finding a maximally leafy forest F of G.

We give an approximation algorithm for maximum leaf spanning tree in this section. Given a graph G, our algorithm computes a spanning tree T for G by the following two steps.

1. Obtain a maximally leafy forest F for G.
2. Add edges to F to make it a spanning tree T for G.

It follows from Theorem 1 that the approximation ratio of the above algorithm is 3. We show x that our algorithm can be implemented to run in time $O((m+n)\alpha(m,n))$.

We use S(w) to denote the subtree of F that contains node w. The degree of node w in F is kept in d(w). The variable d' is the maximal number of edges adjacent to v that could be added to F without creating

cycles. If uv is one of those d' edges, then S(u) is stores in the set S'. If d' + d(v) is greater than or equal to 3, then we add those d edges to F and union S(v) with those d subtrees S(u).

3- Algorithm Analysis

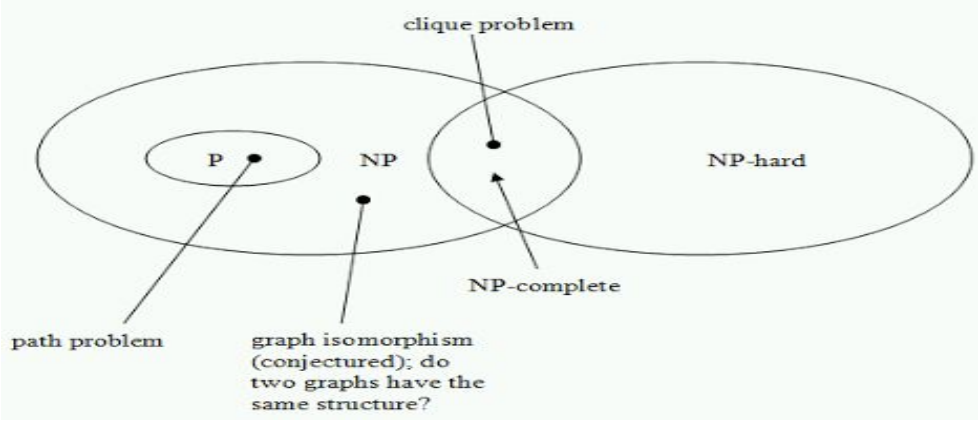
Using the union-find data structure in for set operations, the algorithm runs in time $O((m+n)\alpha(m,n))$. In particular, the first condition in step 9 can be implemented using a find operation on the structure. For the second condition, we first observe that for any set S(u), we may use the representative label in the set structure root of the union-find structure) to denote this set. Also, we observe that for a given node v, if S' contains three or more distinct sets, we proceed to merge these sets into one subsequently (since the condition in step 12 is satisfied). Hence, it suffices to check in the loop in step 8 that for a given node v, the set S' contains at least three distinct sets S(u), subsequently we can combine the check for such candidate edges (v,u) with merging the corresponding sets S(u) and S(v) if the edge passes the test (does not form a cycle). Thus we can assume that the set S' never has more than two elements when we wish to test $S(u) \notin S'$. Consequently, the number of set operations performed is on the order of the degree of v when processing v, giving a total time bound of $O(m\alpha(m,n))$ for this step.

The second step of extending F to a spanning tree can be done by shrinking each leafy subtree in F into a single node and then finding a spanning tree for the corresponding collapsed graph. This can be done in time O(m+n). It follows that the time complexity of our algorithm is $O((m+n)\alpha(m,n))$ which is almost linear in the size of the graph.

Remark: $O(\alpha(n))$ is inverse Ackermann time. The Ackermann function is one of the simplest and earliest-discovered examples of a total computable function that is not primitive recursive. All primitive recursive functions are total and computable, but the Ackermann function illustrates that not all total computable functions are primitive recursive.

4- Experiment Analysis

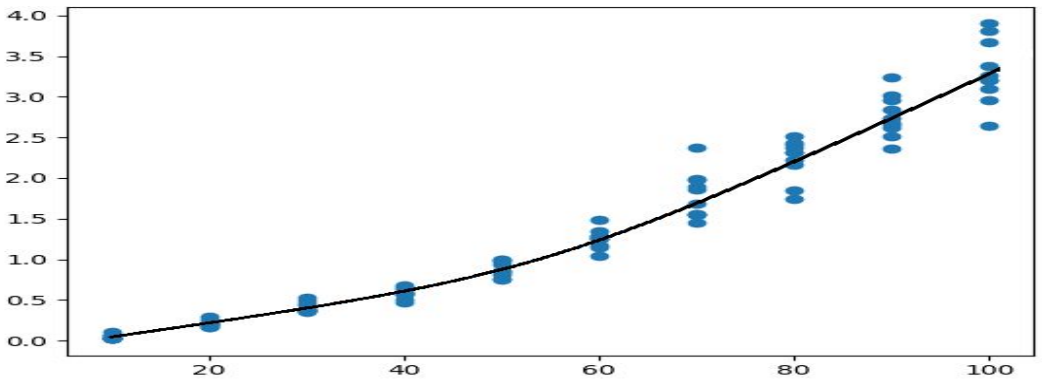
We expect that our algortihm will not run in polynomial time. We reduced our problem to connected dominating set which is a NP complete problem. The reduction takes polynomial time and connected dominating set is also an NP problem therefore our problem is NP- complete.



5- Testing

Performance Testing:

In order to perform our Performance Test, we created input files for each size(10,20,,,,100). With these sizes we prepared 10 different Measurements(N) with Python(inputcreater.py). For calculating running time we used Chrono library, with that we proved our algorithm works on Exponential time when we increase our input size. In order to plot our graph we used Python(PerformanceTestingGraph.py).



We increased correctness of our running time by getting means for each measurement and we generate a line that shows trend line.

We calculated estimated standard error for each size by using our dataset(M = 1.3482496480000001,

Size 10: 0.04825958, Size 20: 0.21226630000000002,Size 30: 0.4129336, Size 40: 0.582524

Size 50: 0.8751539999999999, Size 60: 1.2536589999999999, Size 70: 1.789387

Size 80: 2.232869, Size 90: 2.7634840000000005, Size 100: 3.31196

Sd for size 10 : 0.027948488239529362, Sd for size 20 : 0.03791158475555694

Sd for size 30 : 0.06075876008207468, Sd for size 40 : 0.06418017458686132

Sd for size 50 : 0.09326025326889144, Sd for size 60 : 0.11836898387387351

Sd for size 70 : 0.28366685585775125, Sd for size 80 : 0.2514368550484727

Sd for size 90 : 0.25772158251363686, Sd for size 100 : 0.3906797805535031,

Calculate Estimated Standard Error for 10 input: 0.002794848823952936

Calculate Estimated Standard Error for 20 input: 0.003791158475555694

Calculate Estimated Standard Error for 30 input: 0.006075876008207469

Calculate Estimated Standard Error for 40 input: 0.006418017458686131

Calculate Estimated Standard Error for 50 input: 0.009326025326889144

Calculate Estimated Standard Error for 60 input: 0.011836898387387352

Calculate Estimated Standard Error for 70 input: 0.028366685585775125

Calculate Estimated Standard Error for 80 input: 0.025143685504847267

Calculate Estimated Standard Error for 90 input: 0.025772158251363687

Calculate Estimated Standard Error for 100 input: 0.03906797805535031,

(%95 Confidence Level and t = 1.984)

Confidence interval level for size 10: 0.0427145993327738,0.05380456006672263

Confidence interval level for size 20: 0.2047446415844975,0.21978795841550253

Confidence interval level for size 30: 0.4008790619997164,0.4249881380002836

Confidence interval level for size 40: 0.5697906533619668,0.5952573466380333

Confidence interval level for size 50: 0.8566511657514518,0.8936568342485479

Confidence interval level for size 60: 1.2301745935994233,1.2771434064005764

Confidence interval level for size 70: 1.7331074957978223,1.8456665042021778

Confidence interval level for size 80: 2.182983927958383,2.282754072041617

Confidence interval level for size 90: 2.712352038029295,2.814615961970706

Confidence interval level for size 100: 3.234449131538185,3.389470868461815

(%80 Confidence Level and t = 1.290)

Confidence interval level for size 10: 0.044654225017100715,0.05186493498289929

Confidence interval level for size 20: 0.2073757056653317,0.21715689443346686

Confidence interval level for size 30: 0.4050957199494124,0.4207714800505876

Confidence interval level for size 40: 0.574244757478295,0.5908032425217051

Confidence interval level for size 50: 0.8631234273283129,0.8871845726716868

Confidence interval level for size 60: 1.23838940108027,1.2689285989197296

Confidence interval level for size 70: 1.7527939755943502,1.8259800244056499

Confidence interval level for size 80: 2.200433645698747,2.2653043543012528

Confidence interval level for size 90: 2.7302379158557413,2.7967300841442597

Confidence interval level for size 100: 3.2615623083085983,3.3623576916914018)

Black Box Testing:

In Black Box testing we used some extreme inputs for to see our algorithm's correctness.

5 Node empty edges. (5 0 0 0 0 0) -> " "

1 Item only (1 2 3 0) -> " "

(1 0) -> " "

(1 1 0) -> " "

5 Item (5 3 4 0 1 0 0 5 3 0 2 0) -> Edges: (2 5)(1 4)(1 3)(1 2) Root: (1 2)

100 Item (100,19,86,84,73,62,69,69,79,40,69….) -> (7 71)(7 14)(4 77)(4 70)(4 57)(3 96)…

50 Item(Not connected) (50 0 0 0 0 ….) -> “ “

6- Discussion

Our performance and functional tests results show that our claim is true. We used %95 confidence level to obtain confidence intervals for each size. The results show that when input size increases, the error rate decreases and it shows the correctness our algorithm.

7- References

1. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM 56(5) (2009)
2. Fellows, M.R., McCartin, C., Rosamond, F.A., Stege, U.: Coordinatized kernels and catalytic reductions: An improved FPT algorithm for max leaf spanning tree and other problems. In: Kapoor, S., Prasad, S. (eds.) FST TCS 2000. LNCS, vol. 1974, pp. 240–251. Springer, Heidelberg (2000)
3. Fomin, F.V., Grandoni, F., Kratsch, D.: Solving connected dominating set faster than 2n. Algorithmica 52(2), 153–166 (2008)
4. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM 56(5) (2009)
5. G. Galbiati, F. Maffioli, and A. Morzenti, A short note on the approximability of the maximum leaves spanning tree problem, *Inform*. *Process*. *Lett*. 1994., 45-49.