Bounded Degree Spanning Trees

By: Shriya Mishra

Abstract

The Minimum Spanning Tree is one of the oldest and one of the most basic graph problems in theoretical Computer Science.Extensive researches on the Minimum spanning tree problems had started way back in the 1920's and till today Minimum Spanning Tree problems form the most important areas of research in the field of Computer Science. MSTs have direct applications in the designing of Computer networks, telecommunication networks, transportation networks, water supply networks etc., and recent breakthroughs have been made in the networking of wireless sensors. To serve for the requirements and to improve the performance of these algorithms in its applications many forms of restrictions can be made to the Minimum spanning tree problem. The Bounded Degree Minimum Spanning tree(BDMST) and the Minimum Leaf Spanning Tree(Min-LST) are two of the many restrictions that have been discussed in this paper. In this paper algorithms for the Minimum Spanning Tree as well as the algorithms for the BDMST and the Min-LST (that give a near optimal solution) have been illustrated.

Introduction

Let us consider a telephone company that wants to connect a group of n cities $\{v_1, v_1, \ldots, v_n\}$. Let $c(v_i, v_j)$ be the cost of laying cables between a pair of cities v_i and v_j. The company should connect each of the cities in such a way that there every city v_i can communicate with every other city v_j.

The company can establish a connection between cities in many ways. To establish a connection between cities the company can connect each city every other city, for such a case, if there are n cities there will be $\frac{n(n-1)}{2}$ connections and this is the maximum number of connections possible. Let us take another case where the company can connect the cities with $(n-1)$ connections and this is the minimum number of connections that are required to connect n cities. Obviously, the company would choose the second option as it is cost effective and the company would have to spend an amount that is much lesser than amount that would be spent when there are $\frac{n(n-1)}{2}$ connections.

In Graph Theory, the problem can be viewed as a graph with n vertices and all the vertices in the graph have to be connected. The vertices v of the graph G represent the cities the edges e represent the cable connections between cities. The cost of laying a cable is represented by the cost of each edge c. The n vertices $\{v_1, v_1, \ldots, v_n\}$ are connected by $n-1$ edges and such a graph is called a *tree* more precisely, *A Spanning Tree.*

Tree

A graph $G = (V, E)$ is called a tree if it is connected and is acyclic. The following theorem gives a characterization of a tree.

Theorem 1. *[**?**] A graph $G = (V, E)$ is called a tree if and only if*

(a) It has $|V| - 1$ edges.

(b) $G = (V, E)$ is connected and $|E| = |V| - 1$.

(c) $G = (V, E)$ is acyclic and $|E| = |V| - 1$.

(d) Every edge in the graph is a cut- edge.

(e) $V \cup \{e\}$ always forms only one cycle $\forall e \notin E$

Spanning Tree

Given a connected graph $G = (V, E)$, a subgraph $T = (V', E')$ is called a *spanning tree* of G if T is acyclic, connected and $V = V'$. Here, V represents the set of all vertices and E represents a set of all edges in G.

Minimum Spanning Tree :

Let us again consider the problem discussed earlier. The company reduces the total expenditure on the connections by choosing $(n-1)$ connections. The expenditure can be further reduced by choosing those $(n-1)$ edges that have minimum cost. A spanning tree with minimum cost is called a **minimum spanning tree**. More formally, given a graph $G = (V, E)$ and an edge weight function $c : E \to \mathbb{R}^+$, in minimum spanning tree problem, it is required to find a spanning tree $T = (V, E_T)$ of G with minimum total edge weight $c(T) = \sum_{e \in E_T} c(e)$.

Theorem 2. *Let $G = (V, E)$ be a connected graph with edge weight function $c : E \to \mathbb{R}^+$. Let T be a spanning tree of G. Then the following statements are equivalent.*

(a) T is a minimum spanning tree.

(b) If $e = (x, y) \in E(G) - E(T)$ no edge in the path of (x, y) has greater cost than e.

(c) If $e \in E(T)$ then e is the edge with the minimum weight among all the edges $\delta(V(c))$ where C is the connecting component and $\delta(V(C))$ is the set of all the edges in the cut set.

Algorithms for minimum spanning tree

There are two algorithms that are used to find out the the minimum spanning tree of a given graph $G = (V, E)$, they are Kruskal's algorithm and Prim's algorithm. Kruskal's algorithm is based on the Theorem 1(b), and Prims algorithm is based on Theorem 1(c). Kruskal's algorithm starts from the graph without any edge and includes an edge at a time of least cost, step by step, while maitaining the acyclicity property. Whereas, Prim's algorithm starts from a graph without any edge and expands one particular acyclic component by including one edge at a time of minimum cost. The pseudocode of these two algorithms are described as given bellow.

Kruskal's Algorithm

```
Algorithm 1: The Kruskal's Algorithm
Input: An undirected graph $G = (V, E)$;
Output: Minimum Spanning Tree;
Sort the edges of $G$ in a non-decreasing order;
Let this order be $e_1, e_2, \ldots, e_{|E|}$;
$E_T = \phi$;
for $i = 1$ to $|E|$ do
    if $(E_T \cup \{e\})$ is acyclic then
        $E_T = E_T \cup \{e\}$;
    end
end
return $(T = (V, E_T)$
```

Time complexity of Kruskal's algorithm

1. Sorting the edges in an increasing order of their weights can be done in $O(E \log V)$ time.
2. Checking for the acyclicity of the tree can be done by using BFS (Breadth First Search) or DFS(Depth First search) which takes $O(E)$ time. The Loop runs for $O(V)$ time, hence the total time complexity is $O(EV)$.

The Time Complexity of Kruskal's is $= O(E \log V) + O(EV) \Rightarrow O(EV)$.

Improving the time complexity

The time complexity of the Kruskal's algorithm can be improved using the Union-Find Data Structure. In this data structure the vertices of the tree are arranged in a data structure known as a linked list. The following operations are used :

1. Find-set(x) : This operation returns a pointer to the representative containing x.
2. Union(x,y) : This operation appends the list x to the list y.

Algorithm Analysis with the above data structure :

1. Sorting the edges in an increasing order of their weights can be done in $O(E \log V)$ time by using a sorting algorithm(Eg: Merge Sort).

2. Let us assume two trees x,y. The UNION(x,y) function appends the list of x to y.To reduce the number of changes made to the pointers of each list, we append the list with the smaller length to the list with larger length. Now we analyse the changes:

Changes	size(minimum length after appending)
1	2
2	4
3	8
$\vdots$	$\vdots$
$\log V$	V

   Explanation : Let us consider a list l_1 and l_2 where length of $l_1 \leq l_2$, the minimum length of each of the lists should be atleast 1. When we combine the two lists using the UNION(X,Y) the length of the list becomes atleast 2. When the new list is combined with another list the length becomes atleast 4, if this list is further combined with another list, the length becomes atleast 8. This procedure is continued till the length of the final list becomes V (indicating that all the vertices are covered).The length of the list becomes V after a minimum of $\log V$ combinations.

   The above Analysis is called AMORTISED ANALYSIS.

   So the total number of changes made by the UNION(X,Y) is $O(\log V)$. The FIND-SET(X) takes constant time.As the For loop runs for E times, the total time taken by the loop will be $O(E \log V)$.

THE TIME COMPLEXITY OF KRUSKAL'S IS $=O(E \log V) + O(E \log V) \Rightarrow O(E \log V)$.

Correctness of Kruskal's Algorithm

Let us consider a graph G with T as its minimum spanning tree obtained by applying the Kruskal's Algorithm to the graph G. In the Kruskal's Algorithm a spanning tree is built by adding one edge at a time. Let us say that the edges of T were added in the order of $\{e_1, e_2, e_3 \dots e_{n-1}\}$. Suppose T is not a minimum spanning tree instead we choose T^* to be a minimum spanning tree that follows T for the longest time. T^* has the edges $\{e_1, e_2, e_3 \dots e_k\} where k < n-1$ (we assume that no minimum spanning tree has all the edges $\{e_1, e_2, e_3 \dots e_{k+1}\}$).

We add an edge e_{k+1} to $T^*.T^* + e_{k+1}$ definitely produces a cycle C, but the spanning tree T does not have a cycle, it means that there exists some edge e' that does not exist in T, moreover, as T^* is being constructed using the Kruskal's Algorithm the weights of e' and e_{k+1} are compared and we find that $c(e') > c(e_{k+1})$. This means that $T^* + e_{k+1} - e'$ has weight that is lesser that T^* with the edges $\{e_1, e_2, e_3 \dots e_{k+1}\}$ which is a contradiction to the assumption that no minimum spanning tree has all the edges $\{e_1, e_2, e_3 \dots e_{k+1}\}$. Therefore T has to be a minimum spanning tree.

Prim's Algorithm

```
Algorithm 2: Prim's Algorithm
  Input: An undirected graph $G = (V, E)$
  Output: Minimum Spanning Tree
  Select a vertex $v \in V$;
  $E_T = \phi$;
  $P = \{v\}$;
  while $P \neq V$ do
      select an edge $e = (r, s)$ with $r \in P$, $s \in V - P$ and, $e$ of minimum cost
      then update $P = P \cup \{s\}$;
      $E_T = E_T \cup \{e\}$;
  end
  return $(T = (V, E_T)$
```

Time complexity

For Prim's Algorithm we use a data structure called a HEAP. The Heap is actually an array and can be viewed as an almost complete binary tree, in other words, every element in an array corresponds to a node in the heap.

To construct a tree using the Prim's Algorithm we use two operations of the HEAP data structure.

1. BUILD-MIN-HEAP : This operation is used to include the vertices that form an edge in the minimum Spanning tree, in the HEAP array.
2. EXTRACT-MIN : This operation is used to find out the vertex of the light edge incident by another vertex in the cut-set.

ANALYSIS :

1. Initialising takes constant time.
2. Building a max heap using the BUILD-MIN-HEAP operation (to construct P), takes $O(E \log V)$ time .
3. Searching for the vertex of the light edge incident by a vertex in P this is done by the operation EXTRACT-MIN ,this operation takes $O(\log V)$. As the loop runs for $|V| - 1 = E$ times the total running time is $O(E \log V)$.

THE TIME COMPLEXITY OF PRIM'S ALGORITHM IS $=O(E \log V)$.

Correctness of Prim's algorithm

The correctness of the algorithm can be proven by the Theorem 2 (c).The loop ensures that the edge chosen is the minimum of all the edges incident by the vertices in the set P. As vertices are being chosen from the sets P and $V - P$ no cycles will be formed.Hence, the output of the algorithm is a minimum spanning tree.

Linear program for the Minimum spanning tree problem

Let E_T be the edges of the minimum spanning tree T of graph $G = (V, E)$ and let X be the characteristics vector such that $X = \{x_1, x_2, x_3, \ldots\}$.

If $x_e = 1 \rightarrow$ the edge is a part of the minimum spanning tree.
If $x_e = 0 \rightarrow$ the edge is not a part of the minimum spanning tree.
Then the linear programming formulation of the minimum spanning tree would be ,

$$\begin{aligned}
\text{Minimize} \sum_{e \in E} c_e x_e & & \\
\text{Subject to} \sum_{e \in E} x_e(E(V)) = n - 1 & & (1) \\
\sum_{e \in E(S)} x_e(E(S)) \leq |S| - 1, & \quad \forall S \subset V & (2) \\
x_e \in \{0, 1\} & \quad \forall e \in E & (3)
\end{aligned}$$

Explanation :

1. Inequality (1) implies that there are exactly $(n-1)$ edges.
2. If S is spanning tree subgraph such that $S \subset V, S \neq \phi$(i.e., S has atleast one vertex) then $E(S)$ consists of all those edges that have both the end points in S. The Inequality (2) eliminates all the simple cycles formed within the subset S by ensuring that number of edges in S is less than $|S|$.

MST Algorithm using LP :

```
Algorithm 3: Algorithm for MST using LP
Input: An undirected graph $G = (V, E)$
Output: Minimum Spanning Tree
Initialise $F = \phi$;
while $V(G) \neq \phi$ do
    Compute the optimal solution $x$, for the graph $G$ using the
    LP-MST.;
    if $x_e = 0$ then
        Eliminate the edge $e$.
    end
    else
        Choose that edge $e = (u, v)$ where the $deg(v)$ is at most 1.;
        Update,$F = F \cup \{e\}$ and $G = G - \{v\}$.
    end
end
return $F$
```

Explanation :

We choose a set F that will contain all the edges that form a part of the MST and we gradually decrease the vertices of the graph G on the addition of every edge to F. To add an edge to F, we compute its optimal solution x_e for the graph G using the LP-MST. If $x_e = 0$ the edge is ignored, else we add e to the set F.

Bounded Degree Minimum Spanning Tree

So far we have dealt with the the Minimum Spanning Tree where, for a given graph $G = (V, E)$ we find a path of minimum cost.

In the Bounded Degree Minimum Spanning Tree, apart from finding the minimum spanning tree we add restrictions to the number of edges that are to be incident by each vertex, in simpler words, we restrict the degree of each vertex by k, where k is the maximum degree of the vertex v of graph G.

We define a **Bounded degree Minimum Spanning Tree** as an undirected graph $G = (V, E)$ with c_e as the cost of each edge and $B_v, \forall v \in V$ where B_v is the bounded degree of each vertex such that, $deg(v) \leq B_v, \forall v \in$

V, i.e., the degree of each vertex should be at most B_v.

Linear Program for the Bounded Degree MST

[?]

Let E_T be the edges of the minimum spanning tree in graph $G = (V, E)$ and let X be the characteristics vector such that $X = \{x_1, x_2, x_3, \ldots\}$.
if $x_e > 0 \rightarrow$ the edge is a part of the minimum spanning tree.
if $x_e = 0 \rightarrow$ the edge is not a part of the minimum spanning tree.
Let, the bounded degree be B_v.

For the LP of the Bounded Degree Minimum Spanning tree, we just add the degree constraint to the LP of the Minimum spanning Tree Problem discussed earlier.

$$\begin{aligned}
&\text{Minimize} \sum_{e \in E} c_e x_e \\
&\text{Subject to} \sum_{e \in E(V)} x_e(E(V)) = n - 1 && && (4)\\
&\sum_{e \in E(S)} x_e(E(S)) \leq |S| - 1, && \forall S \subset V && (5)\\
&\sum_{e \in \delta(v)} x_e \leq B_v, && \forall v \in W && (6)\\
&0 \leq x_e \leq 1 && \forall e \in E. && (7)
\end{aligned}$$

We assume that the set $W \subseteq V$ contains all the vertices that have a degree bound. We will see later that once $W = \phi$ there are no degree constrains then the whole algorithm is like that of a minimum spanning tree.

BD- MST algorithm using LP

```
Algorithm 4: Algorithm for Bounded Degree-MST using LP
  Input: An undirected graph $G = (V, E)$
  Output: A Spanning Tree with bounded degree $B_v + 2$
  Initialise $F = \phi$.;
  while $V(G) \neq \phi$ do
      Compute the optimal solution $x$ for an edge $e$ in for the graph $G$
      using the LP- BDMST.;
      if $x_e = 0$ then
          the edge $e$ is eliminated.
      end
      if there is a vertex $v$ of the edge $e = (u, v)$ such that $deg(v) = 1$
      then
          we add the edge $e$ to the set $F$ and update $F = F \cup \{e\}$ ,
          $G = G - \{v\}$, $W = W - \{v\}$ and $B_v = B_v - 1$.
      end
      if there is a vertex $degree \leq 3$ then
          Remove the vertex, Update $W = W - \{v\}$ and $B_v = B_v - 1$.
      end
  end
  return $F$.
```

Explanation :

We find the optimum solution x_e for every edge in the graph G. For the BDMST, the optimal solution x_e is always greater than 0 and need not be one for all the edges. If $x_e = 0$ we simply eliminate that edge. If the degree of a vertex is 1 we add it to the set F and this forms the leaf node for the spanning tree. When the value of x_e lies between 0 and 1, it is an observation that the degree of the vertex is at most 3 (i.e., $deg(v) \leq 3$).When we find such a vertex v with at most three edges incident at it we remove the degree constraint on v by removing the vertex from the set W. Once the vertex v is removed all the edges incident on v are included in the set F. In this process, the constraint $B_v = k$ is violated. In the worst case if $B_v = 1$ then the violation of the degree is at most 2.

So, we can conclude that a polynomial time algorithm for a BDMST produces a spanning tree with a bounded degree of $B_v + 2$. The solution produced by the algorithm has a cost almost equal to the cost of the minimum spanning tree that follows the degree constraint, this can be proved by the A+2 approximation algorithm.The above conclusion has been formulated as a theorem by Goemans which can be illustrated as:

Theorem 3. *There exists a polynomial time algorithm for the Bounded Degree Minimum Spanning Tree problem that returns a spanning tree T such that the cost of T, $c(T) \leq opt$ and $degree(v) \leq B_v + 2$ for each vertex $v \in V$, where opt is the cost of the optimal solution satisfying all degree bounds exactly.*

Minimum Spanning tree with restricted number of leaves

In the previous section, we came across the Bounded Degree Minimum Spanning Tree where restrictions were made to the degree of each node i.e., we made restrictions on the number of vertices that can be adjacent to each vertex V of the Graph $G = (V, E)$. In this section we will be restricting the number of leaf nodes of a Graph G.

Designing an algorithm with maximum number of internal nodes

Decreasing the number of leaf nodes is equivalent to increasing the number of internal nodes.Hence, designing an algorithm to find out the maximum number of internal nodes would eventually give us a spanning tree with minimum number of leaf nodes.

Before designing the algorithm we will first make a clear picture of the Depth First Search (DFS). Depth-first search (DFS) is an algorithm for traversing or searching a tree or a graph data structure. One starts at the root(let the root be r), selecting any node and explores as far as possible along each branch before backtracking. The DFS visits each and every node, thus we get a spanning tree, say T.

Now, let us analyse the spanning tree T that was obtained as a result of the DFS. There are nodes that do not have any children, such nodes are called *d-leaves*. If the root r is a leaf, then it is not considered to be a *d-leaf*. The tree T of a DFS gives rise to two cases :

Case 1: if the r is not a leaf node then the set of *d-leaves* obtained, form an independent set of leaves of the graph G.

Case 2: If r is a leaf then there may be a possibility that r is adjacent to another leaf node l.

We design our algorithm based on the above observations. If we obtain a DFS that satisfies *Case:1*, we have already obtained a spanning tree with independent leaves, so we cannot reduce number of leaves of the tree further. If we obtain a tree that satisfies *Case:2*, it implies that the tree has to be restructured to produce a set of independent leaves. From the algorithm illustrated below, we find out a spanning tree with independent leaves and we will further prove that the algorithm gives a solution that is closer to the optimal solution.

Algorithm

Algorithm 5: Algorithm for the ILST

```
Input: An undirected graph $G = (V, E)$
Output: A Spanning Tree T with independent leaves
Choose any arbitrary node and perform DFS;
Initialise $T = DFS(G)$;
Let $r$ be the root of $T$ ;
if $\underline{deg(r) = 1 \text{ and } l \text{ be a leaf node such that } (r, l) \in E(G)}$ then
    Choose the nearest branching node from $l$, let that node be $x$;
    Choose the nearest node from $x$ in the path of $(x, l)$, let that node be $y$;
    Connect $(r, l)$;
    Disconnect $(x, y)$;
    Add $(r, l)$ to $T$;
end
return $\underline{T.}$
```

The Max-IST produces a spanning tree with maximum number of internal nodes (which in turn gives us the spanning tree with minimum number of leaf nodes) but the tree T obtained may or may not be the optimal solution to our problem. This is because we use the the depth first search algorithm which arbitrarily selects a node and starts exploring all the nodes that ultimately gives us a spanning tree.The spanning tree obtained may or may not be a tree with minimum leaves from all the possible trees of the graph G. Hence, T may or may not be an optimal solution to our problem but we can prove that the optimality of the tree T produced by Max-IST is closer to the optimal solution to the tree T^* (here we assume that T^* is the optimal solution) which would be produced by an algorithm for Min-LST. The above statement has been formulated as a Theorem and before we proceed to prove the statement it is important to have an idea about some properties of a graph that are discussed below.

Cut-Asymmetry : Cut Asymmetry ca of a graph G is defined as $ca(G) = max_{X \subset V, X \neq \phi}(comp_G(X) - comp_G(V - X))$ where $comp_G(X)$ represents the connected components of G in set X.

Lemma 1. *Let T be a tree of the graph G with a minimum of three vertices then, $ca(T) = |V_1(T)| - 1$ where,$|V_1(T)|$ is the number of degree 1 vertices.*

Proof :

The proof of the lemma consists of two parts:

(a) Proving $ca(T) \geq |V_1(T)| - 1$

$$comp_T(V_1(T)) - comp_T(V - V_1(T)) = |V_1(T)| - |V - V_1(T)| \tag{8}$$

if $V_1(T)$ is an independent set of leaves of T then $|V - V_1(T)|$ spans a single sub tree. Hence,$|V_1(T)| - |V - V_1(T)| = |V_1(T)| - 1$

$$\begin{aligned} ca(T) &\geq comp_T(V_1(T)) - comp_T(V - V_1(T)) \\ &= |V_1(T)| - 1 \end{aligned}$$

(b) Proving $ca(T) \leq |V_1(T)| - 1$ let $X \subset V$ $ca(T) = comp_T(X) - comp_T(V - X)$. Let $x = comp_T(X)$, $x' = comp_T(V - X)$ and $d(T)$= degree of the tree T. Let $e_T(X) = |X| - x$ and $e_T(V - X) = n - |X| - x'$,where n= number of vertices and $e_T(x)$ represents the number of edges of X in the tree T.

$$\begin{aligned} d_T(X) &= 2e_T(X) + e_T(X, V - X) \\ &= 2e_T(X) + n - 1 - (e_T(X) + e_T(V - X) \\ &= 2(|X| - x) + n - 1 - (|X| - x + n - |X| - x') \\ &= 2(|X|) - x + x' - 1 \end{aligned} \tag{9}$$

We can observe that every Internal node makes a contribution of atleast 2 to $d_T(X)$. The total contribution of the internal node can be formulated as: $2|X|-|V_1(T)\cap X|$. This is definitely lesser than $d_T(X)$. Based on this we can derive :

$$\begin{aligned} d_T(X) &\geq 2|X| - |V_1(T) \cap X| \\ |V_1(T) \cap X| &\geq 2|X| - d_T(X) \\ |V_1(T)| &\geq 2|X| - d_T(X) \end{aligned}$$

but from *eqn(2)*, $2|X| - d_T(X) = x - x' + 1$

$$\begin{aligned} |V_1(T)| &\geq 2|X| - d_T(X) \\ &\geq x - x' + 1 \\ &\geq comp_T(X) - comp_T(X) + 1 \\ &\geq ca(T) + 1 \end{aligned}$$

Thus, the above lemma is proved.

Theorem 4. *The algorithm ILST is a 2-Approximation of Max-LST.*

Proof: Let T^* be an optimal spanning tree with maximum number of internal nodes. Let T be the tree given by the Max-LST. We use Lemma 1 to prove the Theorem 4.

$$\begin{aligned} |V_1(T^*)| &= ca(T^*) + 1 \\ &\geq comp_{T^*}(V_1(T)) - (comp_{T^*}(V - V_1(T)) + 1 \\ &\geq |V_1(T)| - |V - V_1(T)| + 1 \\ &\geq 2|V_1(T)| - n + 1 \end{aligned}$$

The above equation is true because $V_1(T)$ is an independent set.

$$\begin{aligned} |V_{\geq 2}(T^*)| &= n - |V_1(T^*)| \\ &\leq n - 2|V_1(T)| + n - 1 \\ &= 2(n - |V_1(T)|) - 1 \\ &\leq 2(n - |V_1(T)|) \\ &= 2|V_{\geq 2}(T)| \end{aligned}$$

From the above equation we can conclude $\frac{V_{\geq 2}(T^*)}{V_{\geq 2}(T)} = 2$ and this proves the Theorem 4

Applications of the Minimum Spanning Tree

The MST is the most important and the most commonly used algorithms in the design and configuration of a network. Of late, the MST finds its application in the *Ad hoc Network* or simply the **MANET**.

A **MANET** is a self- configuring, infrastructureless wireless network in which the participating devices are allowed to move independently in any direction. In such a network of randomness the MSTs give an optimal data centric routing. Let us consider a case where an Ad hoc network consists of a large number of battery-powered sensors, where the sensors are distributed across a region. Large amounts of data packets are exchanged between each of these sensors and the data is transferred at the cost of the energy of these sensors. So, sending large amounts of data with limited power is a challenge. Another challenging aspect of the MANET is its randomness in topology because the devices are mobile. This randomness causes frequent change in paths and routes between devices which increases the possibility of the loss of data packets.

The Minimum Spanning Tree eliminates most of the problems discussed above. It increases the routing efficiency and reduces the energy consumed. The MSTs allow point-to-point configuration and communication which reduces the loss of data packets. A dedicated link between the devices allows peer-to-peer message transfer and this almost eliminates traffic caused due to data congestion. Broadcasting and data aggregation is very efficient as it eliminates data redundancy. It helps in easy fault detection and reconfiguration of a connection as Ad hoc networks are prone to such faults.

The creation of the MST is done locally (because every device is in random motion) by a simple algorithm known as the *Nearest Neighbour Tree Algorithm.* It creates a Minimum Spanning Tree in an Ad hoc setting. The Edge Disjoint Minimum Spanning Tree protocol is used in the MANET which chooses a link with optimum stability just the same way the Prim's algorithm chooses an edge with minimum weight.

Complexity of problems

Introduction

We have an idea about various computational problems like the sorting algorithms, the minimum spanning tree etc.These computational problems can be solved, they have their own algorithms that solve the problem within a given limited time. Apart from these problems there are many others that cannot be solved even with a given unlimited time for eg.Turing Halting problem (Given a program and an input, whether the program will eventually halt when run with that input, or will run forever). There are another set of problems that can be solved but they do not have an algorithm defined till date. Based on how these problems can be solved and whether or not they can be solved in polynomial time these problems are broadly classified into the *P class* and the *NP class.*

The P class

The *P class* is a set of problems that can be solved in polynomial time.More formally, they are the problems that can be solved within $O(n^k)$ where k is some constant and n is the size of the input. These problems can be solved by a deterministic Turing Machine.

The NP class

[**?**] The *NP class* is a set of problems that cannot be solved in polynomial time but can be *verified* within polynomial time. This means that for a given *certificate* we can determine whether the certificate satisfies the problem within a polynomial time.

The problems in this class can be solved by algorithms that can make a right *guess*, and such algorithms are very hard to design.

These problems are considered to be solved only by a non-deterministic *Turing machine.*

Reduction

Two language L_1 and $L_2$are said to be **polynomial-time reducible** , in other words $L1$ *reducible* to L_2if there exists a function $f(x)$ for all x, and

$x \in L_1$ if and only if $f(x)$ belongs to L_2.

Theorem 5. *If $L_1 \neq L_2$ and if $L_2 \in P$ then $L_1 \in P$.*

Proof :
Let us consider that A_1 solves L_1, A_2 solves L_2 where A_1 and A_2 are algorithms for languages L_1 and L_2 respectively. The F be the reduction algorithm that computes the reduction function f.

When we use the algorithm A_1, it takes x as the input. Then the reduction function F is used to compute the reduction function $f(x)$ which is given as an input to the algorithm A_2.
if, A_2 gives a *yes*, $f(x) \in L_2 \to x \in L_1$. So the overall output of A_1 is true.
else, A_2 gives a *no*, $f(x) \notin L_2 \to x \notin L_1$. So the overall output of A_1 is false.

Correctness : A_1 runs in polynomial time because both F and A_2 run in polynomial time.

NP-completeness

A Language L is said to be NP-complete if

1. $L \in NP$.
2. $L' \leq_p L$ for all $L' \in NP$.

Problems which satisfies only property 2 are said to be **NP-hard**.

Hamiltonian path

A Hamiltonian path is a path in an undirected graph $G = (V, E)$ that visits each vertex exactly once.

Theorem 6. *A Hamiltonian path for a graph $G = (V, E)$ is NP- complete.*

NP-completeness of a Bounded Degree Spanning tree

Theorem 7. *The Bounded Degree Minimum Spanning Tree is NP hard.*

Proof: Let us consider a Bounded Degree Minimum Spanning Tree which satisfies the constraint $deg(v) \leq k, \forall v \in V$. Let us consider a case where $k = 2$, which means that the vertices in graph $G = (V, E)$ have a maximum degree of 2. To prevent the formation of cycles there must be exactly 2 vertices with degree at most 1.

The above problem is equivalent to the problem of searching a Hamiltonian Path in a graph G. As the Hamiltonian Path reduces to the above problem, we can conclude that the Bounded Degree Minimum Spanning tree is NP- hard(as the Hamiltonian path is NP-hard).

In the earlier sections we have found a polytime algorithm which find a $(k + 2)$-bounded spanning tree if a k-bounded spanning tree exists and the cost of the spanning tree is close to the cost of the MST with bounded degree exactly k.

NP-completeness of a Minimum leaf spanning tree

Theorem 8. *The Minimum Leaf Spanning tree is NP-Complete.*

Proof:

Let us consider a Spanning tree T of graph G with a maximum of two leaves(this is the minimum number of leaf nodes possible for a tree). This implies that all the other nodes form the internal nodes of the spanning tree. If we construct a path between the two leaf nodes, it will cover all the the nodes of the graph G, thus forming a Hamiltonian Path.As the Hamiltonian path can be reduced to the the problem discussed above we can say that the Minimum Leaf Spanning Tree is NP-complete because the Hamiltonian path is NP-complete according to Theorem 6.

Conclusion and Future Work:

In this paper we have discussed about MST, BDMST and the Min-LST. The MST for a Graph G is given by the Prim's and Kruskal's Algorithm and these algorithms are polynomial time solvable. We have also seen that the problems like the BDMST and the Min-LST problems are NP complete even then it is possible to design a polynomial time algorithm that give us a solution that is nearly optimal. For example, we have seen that the BDMST algorithm gives us a solution which violates the optimal solution by at most

2 we have also seen that the algorithm for the Min-LST gives us a solution that is nearly optimal.

As a future work we would like to find out an algorithm that would help us find a solution to the BDMST problem that would help us find a solution which would violate the optimal solution by 1 instead of 2. Our future work will also focus on designing an algorithm that would help us find a solution to the Min-LST, whose solution would be :$\frac{V_{\geq 2}(T^*)}{V_{\geq 2}(T)} < 2-\epsilon$ so that our solution is even more closer than the optimal solution.