Generating all the spanning trees in a graph

Gaganjeet Reen
ICM2015003

Vinay Surya Prakash
IHM2015004

Swapnesh Narayan
ISM2015002

Leetesh Meena
ISM2014008

Group - D

***Abstract*— Consider a simple, undirected graph G with given Adjacency Matrices X(G).In this paper we propose an algorithm to find whether or not the graph is Hamiltonian or not . Further we evaluate the proposed algorithms time complexity by the help of a plot of Time vs Number of vertices.**

I. INTRODUCTION

An interconnection of points is known as Graphs , or more formally , A graph **G** is a set of **E** and **V** , where **E** denotes the set of edges and **V** denotes the set of Vertices . Here a vertex v represents a point or node and an edge e is a connection between two vertices. v_i and v_j are known as endpoints of that edge if an edge e_{ij} connects the vertices v_i and v_j.

A) Adjacency Matrix
The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices.If there exists an edge from v_i to v_j ,the corresponding entry in the adjacency matrix is 1,otherwise the entry is 0. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric.

B) Cycle in a graph
A cycle in a graph is a closed walk(alternating sequence of vertices and edges with no edge being repeated) in which each vertex except the terminal vertex appears once.A cycle is also known as a circuit.

C) Tree
A tree is a connected graph without any circuits.There is one and only one path between any two vertices in a tree.A tree with n vertices always has n-1 edges.

D) Spanning Tree
A tree T is said to be a spanning tree of a connected graph G if T is a subgraph of G and T contains all the vertices of G.

E) Branches
The edges of a graph which are also the edges of a spanning tree are said to be the branches of a graph with respect to that spanning tree.

F) Chords
The edges of a graph which are not a part of the edges of a spanning tree are said to be the chords of a graph with respect to that particular spanning tree.

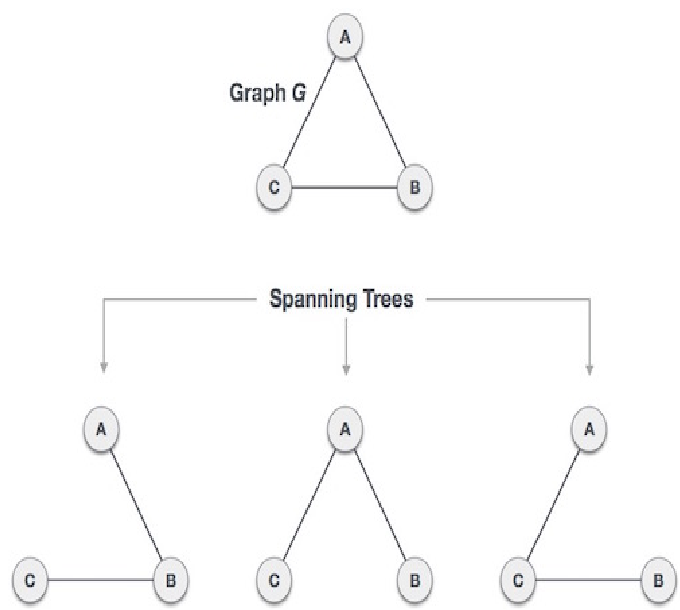

Fig. 1. Example of Spanning trees in a graph

II. MOTIVATION

Efficient polynomial time algorithms are well known for the minimum spanning tree problem. We use these algorithms to find one particular spanning tree and then use different approaches to list all possible spanning trees in a graph. We dicuss two approaches to find all the spanning trees in a graph. These approaches have been discussed in the following section.

III. Methods and Description

We propose two approaches to find all the spanning trees in a graph.

A) **Cyclic Interchange Method**

We first need to find atleast one spanning tree in the graph and we will then build the other spanning trees on top of that. We know that Prim's algorithm exists to find the minimum spanning tree. Working of Prims Algorithm has been discussed in the proposed algorithms section of the paper.

To get all possible spanning trees in this particular graph, we do the following steps :-

- Make all the edges of the graph of the same weight.
- Use Prims Algorithm to get a spanning tree.
- Add any chord to the spanning tree.This forms a fundamental circuit in the tree.
- Remove any branch from the fundamental circuit.
- Repeat the process for all the different branches in the fundamental circuit.
- After generating all the trees with a particular chord in it, we can repeat the process for all the different chords

B) **Backtracking Method**

This proposed method uses the programming paradigm of backtracking.In order to decide whether or not to add an edge to a partially constructed spanning tree, the algorithm checks that the candidate edge:

- Has a higher numeric index than its predecessor
- Does not have both edges in the same connected component of the tree

The first condition prevents duplicates because the edges are always listed in ascending order. The second one ensures that the tree does not contain too many edges, because an edge is superfluous if both of its endpoints are in the same connected component.

C) **A Cut-Set based algorithm**

In this approach we first make all the edges of the graph of the same weight.We then use Prim's algorithm to generate a spanning tree T.Deleting e from T divides it into two connected components with vertex sets V1 and V2.We order the edges of a spanning tree in a particular manner and add them to a set.We the perform the following steps :-

- Pick a particular edge from the set of spanning tree edges.
- Create two set F and R with F containing all the edges before the particular edge in the set of spanning tree edges and R contains all the edges which have been removed from the original spanning tree so far including the picked edge.
- We then find the substitute of the particular edge from the cutset defined for this edge based on the original spanning tree.
- The substitute edge selected should not belong to the set F or R.
- We repeat this process recursively for all the spanning trees generated till no new trees can be generated.

.

IV. Proposed Algorithms

Algorithm 1 Prims Algorithm

1: Create set **mstSet** to keep track of all vertices already included in the minimum spanning tree.
2: Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3: **while** mstSet doesn't include all vertices **do**
  - Pick a vertex u which is not there in mstSet and has minimum key value.
  - Include u to mstSet.
  - Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

4: **end while**

Algorithm 2 Generate all possible spanning tree using Backtracking

if number of vertices in the tree becomes equal to the vertices in the given graph **then**
  print the tree
end if
for all the edges e in the graph having the index number greater than the previous added edges **do**

  if (the edge e is the first edge) or(the vertices in the edge e are in different connected component) **then**
    add edge e to the tree
    recur for all possible configurations of adding different edges to the tree
  end if
end for

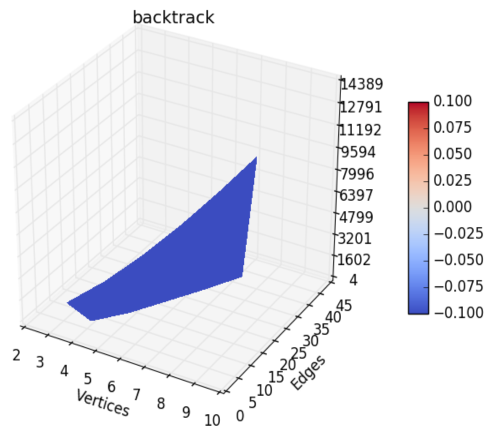

Fig. 2. Time vs vertices vs edges graph for backtracking algorithm)

Algorithm 3 Cut-Set based algorithm

Comment: T is an (F, R)-admissible MST, which is written as $T = F\cup \{e^{k+1},...,e^{n-1}\}$, with $F = \{e^1,...,e^k\}$

for i=k+1,.....n-1 **do**
  Find Cut-Set Cute^i
  Find if a substitute e^{-i} exists for e^i in Cute^i
end for
for i=k+1,....n-1 if substitute exists **do**
  Set T_i = T∪ e^{-i}
e^i and output T_i
  $F = F\cup \{e^{k+1},.....,e^{i-1}$ and $R = R\cup \{e^i\}$
  Repeat Recursively
end for

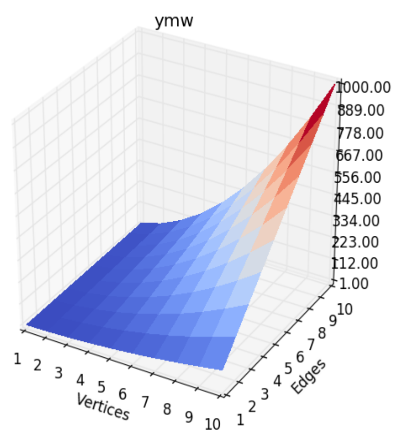

Fig. 3. Time vs vertices vs edges graph for Cut-Set based algorithm

V. IMPLEMENTATION

We have used C++ in the backtracking approach to accept a graph from the user.We accept the graph in the form of an adjacency list.To implement the cut-set based algorithm, we have used python and have used matplotlib library to plot the graphs shown in the results.

VI. RESULTS

Github Repository Link : `https://github.com/piano-man/graph_theory.git`

Analysis and comparison of algorithms :-
The time complexity for the cut-set based algorithm code is $O(Nmn)$ where n, m and N stand for the number of nodes, edges and minimum spanning trees, respectively and where Cut(e^i) can be found in $O(m)$ time and at each subproblem it's repeated atmost n times.Time complexity for the Backtracking algorithm is $O(E!)$ where E is the number of edges which is more than the cut-set based algorithm.

VII. CODE SNIPPETS

```
unsigned int connected_components(const edge *edges, unsigned int n,
unsigned int order,
        int **components)
{
    unsigned int i;
    unsigned int component = 0;
    *components = (int * )malloc(order * sizeof(int));
    if (components == NULL) {
        return 0;
    }
    for (i = 0; i < order; i++) {
        (*components)[i] = -1;
    }

    for (i = 0; i < order; i++) {
        if ((*components)[i] == -1) {
            connected_components_recursive(edges, n, *components, order,
i, component);
            component++;
        }
    }
    return component;
}
```

Fig. 4. Backtracking Algorithm

```
void connected_components_recursive(const edge *edges, unsigned int n,
        int *components, unsigned int order, unsigned int vertex,
        unsigned int component)
{
    unsigned int i;
    /* Put this vertex in the current component */
    components[vertex] = component;
    for (i = 0; i < n; i++) {
        if (edges[i].first == vertex || edges[i].second == vertex) {
            /* Adjacent */
            const unsigned int neighbour = edges[i].first == vertex ?
                edges[i].second : edges[i].first;
            if (components[neighbour] == -1) {
                /* Not yet visited */
                connected_components_recursive(edges, n, components,
order, neighbour, component);
            }
        }
    }
}
```

Fig. 5. Backtracking Algorithm

```
static unsigned int different_components(const edge *tree, unsigned int
t, unsigned int order,
        unsigned int v1, unsigned int v2)
{
    int *components;
    unsigned int different;
    connected_components(tree, t, order, &components);
    different = components[v1] != components[v2];
    free(components);
    return different;
}
```

Fig. 6. Backtracking Algorithm

```
def ymw(F,R,G,edgeList,V,recCount):
    if(recCount > 20):
        return
    recCount += 1
    if(len(F) > len(V) - 1):
        return
    print("Graph is ",G)
    cutSets = {}
    k = len(F)
    if( k == 0 ):
        k = -1
    n = len(V)
    for count,edge in enumerate(G):
        cutSet = findCutSet(edge,G,edgeList,V)
        cutSets[edge] = cutSet
```

Fig. 7. Cut-Set based Algorithm

```
def findCutSet(e,T,edgeList,V):
    ###print("Number of Edges of Graph",3)
    G = []
    for edge in T:
        if edge != e:
            G.append(edge)
    ##print("Graph after cutting,",G)
    s2 = []
    s1 = [1]
    v = 1
    ###print("Number of Edges of Graph",len(G))
    s1.extend(floodFill(G,[],1))
    s1 = set(s1)
    ###print(s1)
    for edge in G:
        if edge[0] not in s1:
            s2.append(edge[0])
        if edge[1] not in s1:
            s2.append(edge[1])
    s2 = set(s2)
    #print("s1",s1)

    if(len(s2) == 0):
        for x in V:
            if x not in s1:
                s2.add(x)
                break
```

Fig. 8. Cut-Set based Algorithm

VIII. APPLICATIONS

The problem of finding the Spanning trees in a graph is of immense use in real life.Some of these applications are :-

- **Networking :-**Finding Spanning trees could prove to be really advantageous in the field of networking to ensure efficient delivery and unique delivery of packets .
- **Mailing Services :-**Finding least cost Spanning

tree could be really advantageous for efficient delivery of mail.

- **Cluster analysis :-** k clustering problem can be viewed as finding an MST and deleting the k-1 most expensive edges.

IX. DISCUSSION

In this paper we used the given algorithms to find all the spanning trees in a graph. While the first approach works, the backtracking approach gives us a more effecient solution and the cut-set based algorithm gives an even more efficient solution.The cut-set algorithm is popularly known as the ymw algorithm as it was proposed by Yamada,Katakoa and Watanabe.The algorithm traditionally lists all the minimum spanning trees in a graph but because we make all the edges of the same weight, it effectively generates all the spanning trees.

X. CONCLUSIONS

We can find all the spanning trees in a graph using the methods suggested in the paper.While approach 1 and approach 2 are brute force algorithms, approach 3 works rather efficiently.

REFERENCES

[1] Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science.

[2] http://www.nda.ac.jp/ yamada/paper/enum-mst.pdf

[3] http://www.martinbroadhurst.com/spanning-trees-of-a-graph-in-c.html

[4] https://en.wikipedia.org/wiki/Cycle_(graph_theory)

[5] https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

[6] http://www.mate.unlp.edu.ar/ liliana/lawclique_2016/07.pdf