Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in DISCRETE MATHEMATICS
Electronic Notes in Discrete Mathematics 53 (2016) 343–354
www.elsevier.com/locate/endm

An Algorithm for k^{th} Minimum Spanning Tree

Amal P M[1] Ajish Kumar K S[2]

Department of Computer Science & Engineering
College of Engineering
Trivandrum, India

Abstract

Minimum Spanning Tree problem is a well-known problem in graph theory. There are many polynomial time algorithms, for finding minimum spanning tree of an undirected connected graph. The k^{th} minimum spanning tree problem is a variation of minimum spanning tree problem that finds k^{th} minimum spanning tree of an undirected connected graph, for some integer k. An algorithm of time complexity $O(k|E||V|log\ k)$ and space complexity $O(k.|V| + |E|)$ for finding k^{th} minimum spanning tree is proposed in this paper. This algorithm is used to solve degree constrained minimum spanning tree problem, an NP-complete problem, with exponential time complexity.

Keywords: k^{th} minimum spanning tree

1 Introduction

Computation of a Minimum Spanning Tree (MST) is one of the fundamental problems in graph theory. Given a weighted undirected connected graph

[1] Email: `amalamalpm@gmail.com`
[2] Email: `ajish@cet.ac.in`

http://dx.doi.org/10.1016/j.endm.2016.05.029
1571-0653/© 2016 Elsevier B.V. All rights reserved.

G(V,E), the minimum spanning tree of G is a tree spanning G which has the minimum weight among all possible spanning trees.

k^{th} minimum spanning tree of a graph is a spanning tree with k^{th} minimum weight among all the possible spanning trees, for some input parameter k.

Spanning subgraph of a graph $G(V, E)$ is a graph $G'(V, E')$ where $V' = V$ and $E' \subseteq E$. Tree is connected acyclic graph. Spanning tree of a Graph G is spanning subgraph of G, which is a tree. There may exist one or more spanning tree for a connected graph. Weight of a graph is the total weight of edges. A minimum spanning tree of graph $G(V, E)$ is a spanning tree of G with weight less than or equal to the weight of every other spanning tree of G. The spanning tree with 2^{nd} minimum weight is called 2^{nd} minimum spanning tree. Thus for a k^{th} minimum spanning tree there exists at least $k-1$ spanning trees with weight less than or equal to weight of that spanning tree.

Algorithms for k^{th} minimum spanning tree can be used to find solutions for constrained spanning tree problems like degree constrained spanning tree, Steiner tree etc., which are known NP-Hard problems. Degree constrained spanning tree of an undirected connected graph is a spanning tree with minimum weight, whose vertices have degree at most Δ, where Δ is the degree constraint.

Here we are proposing a new algorithm for k^{th} minimum spanning tree of an undirected connected graph. The rest of the paper organized as follows. Section 2 discusses some of the previous algorithms and studies related to minimum spanning trees, k^{th} minimum spanning tree and degree constrained minimum spanning tree. The proposed algorithm to find k^{th} minimum spanning tree is explained in the section 3. An application of k^{th} minimum spanning tree to find degree constrained minimum spanning treeis explained in section 4. Section 5 presents a summary and the conclusions.

2 Literature Review

There are Several polynomial time algorithms for the Minimum Spanning Tree problem using greedy strategy. In 1926 Boruvka proposed an algorithm for minimum spanning tree problem. The time complexity of this algorithm is $O(|E|\ log\ |V|)$.

Prim's algorithm[8] is discovered in 1957. It builds a partial spanning tree starting from an arbitrary vertex and in each iteration it adds a vertex to this partial spanning tree which is nearest to it. This repeats until all the vertices are added to the tree. The running time of Prim's algorithm when using binary heap is $O(|E|\ log\ |V|)$.

In 1956 Joseph Kruskal developed an algorithm [9] for MST Problem, which selects edges of graph in non-decreasing order of weight. It repeatedly adds next nearest edge that doesn't forms a cycle until $|V|-1$ edges are added.

2.1 Degree Constrained Minimal Spanning Tree

The Degree Constrained Minimal Spanning Tree (DCMST) problem is a variation of minimum spanning tree problem with an additional constraint over the degree of each vertex in the spanning tree. In 1979 Garey and Johnson[5] proved that in general the problem of finding degree constraint minimum spanning tree (DCMST) is NP-complete by reducing it to Hamiltonian path problem. There exist several methods to find approximate solutions of this NP-complete problem. Gabow [6] developed a branch exchange algorithm to find an approximate solution. Gavish [7] used a branch and bound heuristic which solves the problem in a graph up to 200 nodes.

2.2 k^{th} minimum spanning tree

In 1975 Harold N Gabow[1] developed an algorithm for k smallest spanning trees of a connected graph. k may be known in advance or specified as the trees are generated. The algorithm requires time $O(k|E|\alpha(|E|,|V|)+|E|log|E|)$ and space $O(k+|E|)$. Here $|V|$ is the number of vertices, $|E|$ is the number of edges, and α is Tarjans inverse of Ackermanns function which is very slow growing function.

N. Katoh, T. Ibaraki and H. Mines [2] presented an algorithm for finding k minimum spanning trees in an undirected graph with time $O(k|E|+min(|V|^2,|E|\ log\ log\ |V|))$ and space $O(k+|E|)$. It has less time complexity compared to Gabow's algorithm.

David Eppstein[3] improved the algorithms for generate k smallest spanning trees in a general graph and planar graph. The algorithm for general graphs takes time $O(|E|log\ \beta(|E|,|V|)+k^2)$. And the time required to find planar graphs improved to $O(|V|+k^2)$.

A parallel algorithm proposed in [4] for find k spanning trees from an undirected connected graph $G(V,E)$, which uses $O(|V|^2/log\ |V|)$ processors on a CREW PRAM, with time complexity of $O(T(|V|)+k\ log|V|)$. A parallel algorithm for the k-MST problem on CREW PRAM is presented in this by modifying KIM algorithm[2].

3 Algorithm for k^{th} minimum spanning tree

3.1 Basic idea

Let $G(V, E)$ be the given graph, where V is the set of vertices and E is set of edges.

- Find the minimum spanning tree T_1, using Prim's algorithm. Let W_1 be its weight. The running time of Prim's algorithm is $O(|E|\ log\ |V|)$ and it does not sort the edges for finding minimum spanning tree. So it is a better choice for dense graph than Kruskal's method.
- Then the next minimum spanning tree is constructed by a process called 'minimum edge exchange'. Here an edge $(u, v) \in E$, which not in $T_1$is added to T_1.
  - For every spanning tree there will be one path $uv_0v_1...v_lv$ exists between every pair of vertices. So when adding edge (u, v), there creates a cycle $uv_0v_1...v_lvu$. An example in figure 1

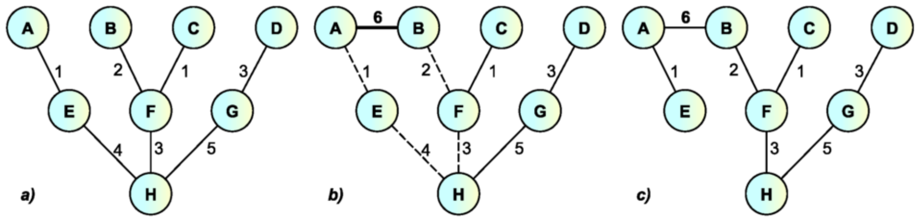


Fig. 1. Steps of edge exchange **a)** A spanning tree **b)** Adding edge (A,B) **c)** New spanning tree after removing edge (H,E)

  - Remove the edge v_iv_j with largest weight smaller than(consider the equal weighted edges too for the first iteration) weight of (u, v) in the cycle $uv_0v_1...v_ivu$.
  - This creates a new spanning tree T_1' that has weight W_1' more than or equal to W_1
- Repeat this edge exchange process with all other edges that are not in T_1
- Find the 2nd minimum spanning tree T_2 from this set of newly generated spanning trees, which have next minimum weight.
- For finding T_3 we have to do edge exchange process in T_1 and T_2.
- Repeat this steps until we get the k^{th} minimum spanning tree T_k.

Algorithm 1 k_minimum_spanning_trees (G, k)

```
Input: Graph G(V,E) and k
Output: An array(K_MSTS) of length k, which contain k minimum spanning
  trees.
  $k^{th}$ element of array will be $k^{th}$ minimum spanning tree.
  Assumptions:
  Prims(G) return minimum spanning tree of G
  List_of_trees is a data structure which stores a list of trees
  Select_min() select a tree with minimum weight from the list
  Add(t) adds tree t into the list, only if tree t is not in the list. If the list
  contain more than k trees then it removes the tree which have largest weight
  among them.
  Remove(t) removes tree t from the list.

  MST=Prims(G) // Finding the min spanning tree of G
  List_of_trees.Add(MST)
  MST=List_of_trees.select_min() // Select tree with min weight from list
  K_MSTS[1]=MST
  for $i$=2 to $k$ do
    List_of_trees.Remove(MST)
    for each edge E which is not in MST do
      Add E to MST // so there will generate a new cycle
      if i=2 then
        // For first MST we have to consider the equal edges too
        Select edges E' from the cycle which have max weight and weight
          must be less than or equal to E
      else
        Select edges E' from the cycle which have max weight and weight
          must be less than E
      end if
      for each edge in E' do
        MST'=Remove E' from MST
        List_of_trees.Add(MST')
      end for
    end for
    MST=List_of_trees.select_min()
    K_MSTS[i]=MST
  end for
  return K_MSTS
```

3.2 Explanation

Here we are using mainly two data structures.

- K_MSTS: which is an array to store k minimum spanning trees.
- List_of_trees: A list to store k trees which can be next minimum spanning tree.

In the first step of the algorithm finds the minimum spanning tree using Prims algorithm and store that in the List_of_trees. Before starting loop we initially put the minimum spanning tree to the K_MSTS. Each iteration of the loop removes the tree with minimum weight from List_of_trees. Then we make a new set of trees which can be next minimum spanning tree by adding one edge, which is not in current minimum spanning tree, to it. When adding that edge the tree becomes a graph with $|V|$ edges, which means that the new graph contains a new cycle. Select an edge which have cost less than the added edge (consider the equal weighted edges too for the first iteration). Remove that edge and add that spanning tree to the List_of_trees. If there is more than one edge with maximum weight, create separate spanning trees by removing each of them. Repeat this step for each edge which are not added to the current minimum spanning tree. After this take the spanning tree with minimum weight from List_of_trees, which is next minimum spanning tree. Add that tree as next spanning tree in K_MSTS. And make new set of trees using this spanning tree. Repeat this process until the k^{th} minimum spanning tree is obtained.

3.3 Correctness of the algorithm

- Initialization-
  - · When i=2, the loop invariant holds
    Array K_MSTS contain minimum spanning tree T_1
    List List_of_trees contain all the trees, which are obtained minimum edge exchange.
  - · When i=3, the loop invariant holds
    Array K_MSTS contain T_1 and T_2
    List List_of_trees contain all the trees, which are obtained by minimum edge exchange in T_1 and T_2
- Maintenance
  - · In each iteration value of i incrementing by one and loop invariants holds
    Array K_MSTS contain i-1 minimum spanning trees and
    List List_of_trees contain candidates for next minimum spanning tree

  - Then we select the minimum weighted tree T_i from this list (which will contain all the spanning trees which have chance to next minimum spanning tree). We remove it from List_of_trees and add to the K_MSTS as the i^{th} minimum spanning tree.
  - Then update List_of_trees by adding new spanning trees generated from minimum edge exchange process on T_i.
- Termination
  - The loop terminates when i=k+1.
  - So in this case the loop invariant K_MSTS contain k minimum spanning trees.
  - Last tree in K_MSTS is the k^{th} minimum spanning tree, which is the required output of our algorithm.

3.4 Complexity analysis

3.4.1 Time complexity analysis - Worst case

- Prims algorithm has complexity $O(ElogV)$.
- List_of_tree will have at most k trees. Because in any time we want to store only minimum k trees. So if we use ordinary lists add, remove, select_min have complexity $O(k)$.
- To find a cycle from a graph which have only one cycle has complexity $O(V)$. (if we use depth first traversal)
- Finding edge E' for edge exchange is $O(V)$
- Number of E' in one cycle will be at most $|V|-1$, because there is only $|V|-1$ edges.

  $T(G(V,E),k) = |E|log|V| + k(k + (|E|)(|V| + |V|k) + k)$
  $= O(|E|log|V| + k^2 + k^2|E||V|)$
  $= O(k^2|E||V|)$.

  If we use AVL tree for storing List_of_tree, then time complexity of add, remove and search will reduce to $O(log\ k)$. Then time complexity of algorithm will reduce to
  $T(G(V,E),k) = |E|log|V| + k(log\ k + (|E|)(|V| + |V|log\ k) + log\ k)$
  $= O(|E|log|V| + k\ log\ k + k|E||V|log\ k)$
  $= O(k|E||V|log\ k)$.

3.4.2 Space complexity analysis

Algorithm k_minimum_spanning_trees (G,k) uses two additional data structures List_of_trees and K_MSTS to store spanning trees. For each tree we use space $O(|V|)$. Space needed for K_MSTS is $O(k.|V|)$. Space needed for List_of_trees is also $O(k.|V|)$ because these are storing maximum k spanning trees in anytime. If List_of_trees containing k trees and trying to add a new spanning tree to List_of_trees results in removal of the largest tree from that $k+1$ spanning trees, k trees from list and one new tree. But to store input graph we need $O(|E|)$ extra space. So the space complexity of algorithm is $O(k.|V|) + O(k.|V|) + O(|E|) = O(k.|V| + |E|)$.

3.5 Comparison with previous algorithms

Harold N Gabow's algorithm [1] requires time $O(k|E|\alpha(|E|, |V|) + |E|log|E|)$ and space $O(k + |E|)$. KIM algorithm [2] slightly faster than Gabows algorithm, and its time complexity is $O(k|E| + min(|V|^2, |E|\ log\ log\ |V|))$ and space is of the same order. David Eppstein's algorithm[3] for general graphs takes time $O(|E|log\ \beta(|E|, |V|) + k^2)$. Our proposed algorithm has time complexity $O(k|E||V|log\ k)$ which is higher than previous algorithms, and space complexity is $O(k.|V| + |E|)$.

3.6 Working of the algorithm with an example

Given a graph G(V,E) with 7 vertices and 8 edges.
$V = \{A, B, C, D, E, F, G\}$
$E = \{AB, BC, BD, CD, DE, DF, EF, FG\}$

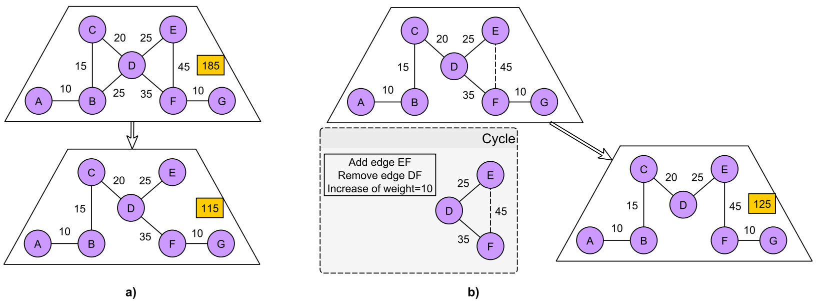


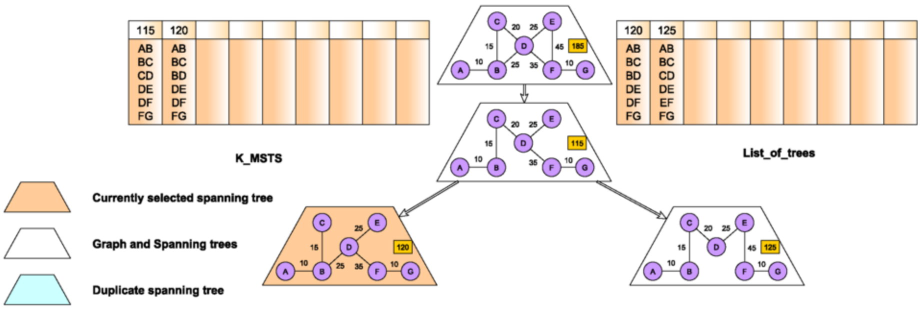
Fig. 2. Adding edge EF to minimum spanning tree

If k is 9, we have to find 9th minimum spanning tree. First find out the minimum spanning tree using Prim's algorithm.

In this example minimum spanning tree has weight 115. Now we are creating next spanning trees from this minimum spanning tree. In this spanning tree only two edges are missing $\{BD, EF\}$. First we add edge EF to the spanning tree. It will create a cycle DEFD, as in figure 2.

We select an edge with highest weight from that cycle which have weight less than newly added edge EF. So we select edge DF. Because it have weight 35 and edge EF have weight 45. We remove this selected edge DF. So we get another spanning tree. It adds weight of 10 to minimum spanning tree. Add this new spanning tree to the list.

Similarly make new spanning tree by adding other edge BD. when adding BD it will generate a new cycle BCDB. We select edge CD to remove. So this edge exchange will add weight 5 to the graph.



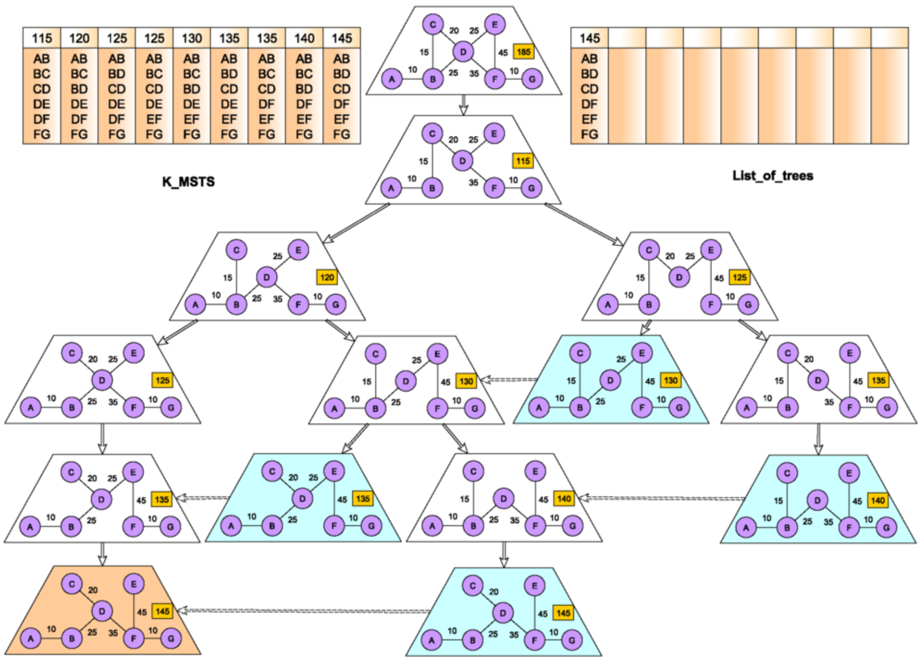
K_MSTS

115	120
AB	AB
BC	BC
CD	BD
DE	DE
DF	DF
FG	FG

List_of_trees

120	125
AB	AB
BC	BC
BD	CD
DE	DE
DF	EF
FG	FG

Fig. 3. 2nd minimum spanning tree



K_MSTS

115	120	125	125	130	135	135	140	145
AB	AB	AB	AB	AB	AB	AB	AB	AB
BC	BC	BD	BC	BC	BD	BC	BC	BD
CD	BD	CD	CD	BD	CD	CD	BD	CD
DE	DE	DE	DE	DE	DE	DF	DF	DF
DF	DF	DF	EF	EF	EF	EF	EF	EF
FG	FG	FG	FG	FG	FG	FG	FG	FG

List_of_trees

145
AB
BD
CD
DF
EF
FG

Fig. 4. 9th minimum spanning tree

We add this two new spanning trees to the list of spanning trees. When adding to the list it checks whether that tree is already added to the list or not. If it is already added we neglect that tree.

Select the spanning tree with minimum weight from the list. That should be the 2^{nd} minimum spanning tree. We make another set of spanning trees by using edge exchange of this spanning tree. Add this new set to the List of trees and remove the selected spanning tree from the list.

In each stage it select a spanning tree from list with minimum weight. Using edge exchange it adds a new set of spanning trees to the list. Continue this process until we get 9^{th} spanning tree. The different stages of algorithm is illustrated in figures 3 and 4.

4 An application of algorithm - Degree constrained minimum spanning tree

Degree constrained minimum spanning tree is the minimum weighted spanning tree of given undirected connected graph, where the degree of each vertex should be less than or equal to given value Δ. Degree constrained minimum spanning treeproblem is NP-complete because for a complete graph with n vertices, there exists at most n^{n-2} spanning trees and we have to find a minimum spanning tree among this, which satisfies the degree constraint.

We use algorithm k_minimum_spanning_trees (G, k) to find k^{th} minimum spanning tree of G. In this algorithm the input is an undirected connected graph G and maximum degree Δ. Output is a spanning tree with minimum weight whose vertices have degree at most Δ. Assumed that there exists a solution for the input graph G.

4.1 Explanation

To find degree constrained minimum spanning treewe generate the spanning trees in the order of weight, and for each tree we check whether it satisfies the degree constraint. Spanning trees are generated until Δ-degree constrained spanning tree is generated.

The working of this algorithm is illustrated in figure 5. The input graph is figure 5.a and value of $\Delta$is 2. Algorithm generate spanning tree in non-decreasing order of weight. In this case 4^{th} minimum spanning tree has maximum degree 2 and all other minimum spanning trees before it has maximum degree > 2. So 4^{th} minimum spanning tree in figure 5.e is the degree constrained minimum spanning treeof G with Δ=2.

Algorithm 2 Degree constrained minimum spanning treeusing k^{th} minimum spanning tree (G, Δ)

Input: Graph G and maximum degree Δ.
Output: Degree constrained minimum spanning treeT

```
Assumption:
*k_minimum_spanning_trees* return $k^{th}$ minimum spanning tree.
*Max_Degree(T)* return degree of vertex which have highest degree in T.

i=1
T=k_minimum_spanning_trees (G, i)
while Max_Degree(T) > $\Delta$ do
  i=i+1
  T=k_minimum_spanning_trees (G, i)
end while
return T
```

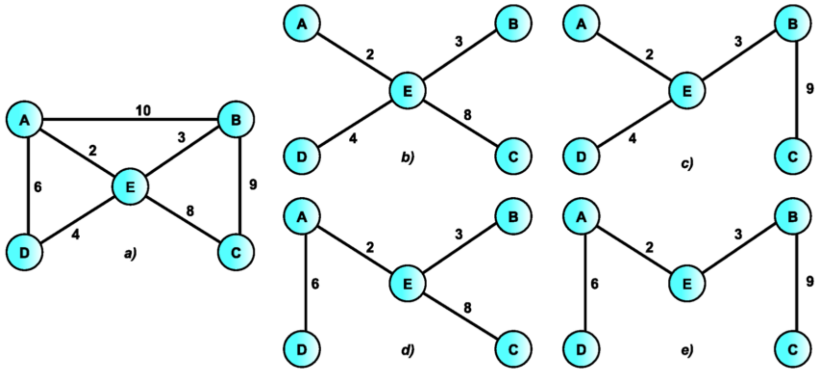


Fig. 5. **a)** Graph G **b)** 1st minimum spanning tree **c)** 2nd minimum spanning tree **d)** 3rd minimum spanning tree **e)** 4th minimum spanning tree which satisfies the constraint $\Delta <= 2$

In the worst case this algorithm has exponential time complexity because it runs until all the spanning trees of G are generated. This algorithm gives the exact solution to this NP-complete problem.

5 Conclusion

The proposed algorithm finds k^{th} minimum spanning tree in polynomial complexity $O(k^2|E||V|)$ in the worst case. When using AVL tree to store k trees its time complexity reduces to $O(k|E||V|log\ k)$ in worst case. This algorithm has a space complexity of $O(k.|V| + |E|)$. An algorithm to solve degree constraint minimum spanning tree using k^{th} minimum spanning tree is also presented here. But in worst case it would have exponential time complexity for finding exact solutions of degree constrained minimum spanning tree, which is an NP-complete problem.

References

[1] H.N. Gabow, Two Algorithms for Generating Weighted Spanning Trees in Order, *SIAM J. Comput.*, **6** (1977), 139–150.

[2] N. Katoh, T. Ibaraki, and H. Mine, An Algorithm for Finding k Minimum Spanning Trees, *SIAM J. Comput.*, **10** (1981), 247–255.

[3] David Eppstein, *Finding the k Smallest Spanning Trees*, Department of Information and Computer Science,University of California, Irvine, CA 92717

[4] Jun Ma, Kazuo Iwama and Qian-Ping Gu, *A Parallel Algorithm for k-Minimum Spanning Trees,IEEE 1997*

[5] M.R Garey and D.S.Johnson, *Computers and Intractability, A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.

[6] H.N. Gabow, A good algorithm for smallest spanning trees with a degree constraint, *Networks*, **8** (1978), 201–208.

[7] B. Gavish, Topological design of centralized computer networks- formulations and algorithms, *Networks*, **12** (1982), 355–377.

[8] R. C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal*, **36** (1957), 1389-1401.

[9] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, *Proceedings of the American Mathematical Society*, **7** (1956), 48-50.