

# An Algorithm to the Maximum Single-Degree-Constrained Spanning Tree Problem

Peiqian Li

High School Attached to Northeast Normal University

P. R. China

PeiqianLi@gmail.com

November 26, 2009



## Abstract

This article discusses an algorithm to the Maximum Single-Degree-Constrained Spanning Tree problem, a variation on the classical MST (Minimum/Maximum Spanning Tree) problem. We present an efficient algorithm with its time/space complexity analysis.

## 1 Introduction

The classical MST (Minimum/Maximum Spanning Tree) problem has been studied extensively, partly because of its important application in the field of computer network design. However, when many terminal computers have to be connected to the central hub, finding a spanning tree that organizes the terminals into subnetworks can lower the cost of implementing a network. Sometimes, the central hub may only allow a limited number of computers or subnetworks to connect to it directly. In this case, we need to find the best spanning tree with an additional requirement on the degree of one special node, which comes to the Maximum Single-Degree-Constrained Spanning Tree problem: given an undirected graph  $G = (V, E)$ , weights  $w : E \rightarrow R$ , a particular node  $v_0 \in V$  and an integer  $K \geq 2$ , find a maximum-weighted spanning tree with the degree of  $v_0$  not exceeding  $K$ .

Below are comprehensive descriptions of the MSDCST problem.

- **Problem Description** Given an undirected graph  $G = (V, E)$ , a particular node  $v_0 \in V$  and a positive integer  $K$ , find a maximum-weighted spanning tree with the degree of  $v_0$  not exceeding  $K$ .
- **Preconditions** Graph  $G$  is finite, simple and connected. It is guaranteed that there exists at least one spanning tree with the degree of  $v_0$  not exceeding  $K$ .
- **Input**  $G$ ,  $v_0$  and  $K$ .
- **Output** The MSDCST of graph  $G$ .

## 2 Notations

The following list contains somewhat uncommon symbols used in this article.

- For a particular edge  $e$ ,  $weight(e)$  represents its weight. Since we're discussing simple graphs only, there is at most one edge connecting two distinct nodes in a graph. Without ambiguity, we use  $edge(a, b)$  to refer to the edge connecting node  $a$  and  $b$ .
- For a particular node  $v$ ,  $E(v)$  denotes the set of edges that connect node  $v$  to some other node in the graph.
- For two sets of nodes  $V_1$  and  $V_2$ ,  $E(V_1, V_2) = \{edge(a, b) | a \in V_1, b \in V_2\}$ . If  $E(V_1, V_2)$  is used where  $V_1$  or  $V_2$  is a single node instead of a set of nodes, then  $V_1$  or  $V_2$  should be considered a set containing that single node.
- For node  $a$  and  $b$ ,  $path(a, b)$  represents the set of all edges that are on the path between node  $a$  and  $b$  in the graph.
- An  ***$S$ -degree spanning tree*** of an undirected graph is a spanning tree where node  $v_0$  has a degree of  $S$ . The ***maximum  $S$ -degree spanning tree*** is the  $S$ -degree spanning tree with the maximum total weight. Note that the MSDCST we are to find is just the one with the largest total weight among all maximum  $X$ -degree spanning trees where  $X \leq K$ .
- The total weight of the ***maximum  $S$ -degree spanning tree*** is denoted by  $W_S$ .

## 3 Algorithm

Since the MSDCST problem is a stronger version of the classical MST problem, it makes sense to try to throw away the additional requirement first. That can be achieved by temporarily removing node  $v_0$  as well as the edges connect to it. If node  $v_0$  and all its adjacent edges are removed, we'll get one or more connected components.  $P$  denotes the number of connected components formed. Note that there doesn't exist any  $X$ -degree spanning tree where  $X < P$ , since we must use at least  $P$  edges to connect  $v_0$  and every connected component in the spanning tree. The following description shows the general idea of our algorithm.

---

**Algorithm 1** Get-MSDCST( $G, v_0, K$ )
 

---

**Require:** An undirected graph  $G = (V, E)$ , a particular node  $v_0 \in V$  and a positive integer  $K$ . There exists at least one spanning tree of graph  $G$  satisfying that the degree of  $v_0$  does not exceed  $K$ .

**Ensure:** The MSDCST of graph  $G$ .

$P \leftarrow$  the number of connected components formed by removing node  $v_0$

$T_P \leftarrow$  the maximum  $P$ -degree spanning tree

$R \leftarrow T_P$

**for**  $I = P + 1$  to  $K$  **do**

$T_I \leftarrow$  the maximum  $I$ -degree spanning tree {This can be done with the help of  $T_{I-1}$ }

**if** the total weight of  $T_I$  is greater than that of  $R$  **then**

$R \leftarrow T_I$

**end if**

**end for**

**return**  $R$

---

The following figure shows an example of a simple connected graph  $G$  on the left. In this example,  $K = 3$  and node  $v_0$  is the node marked by 1 in the graph. On the right are two connected components formed when node 1 and its adjacent edges are removed.

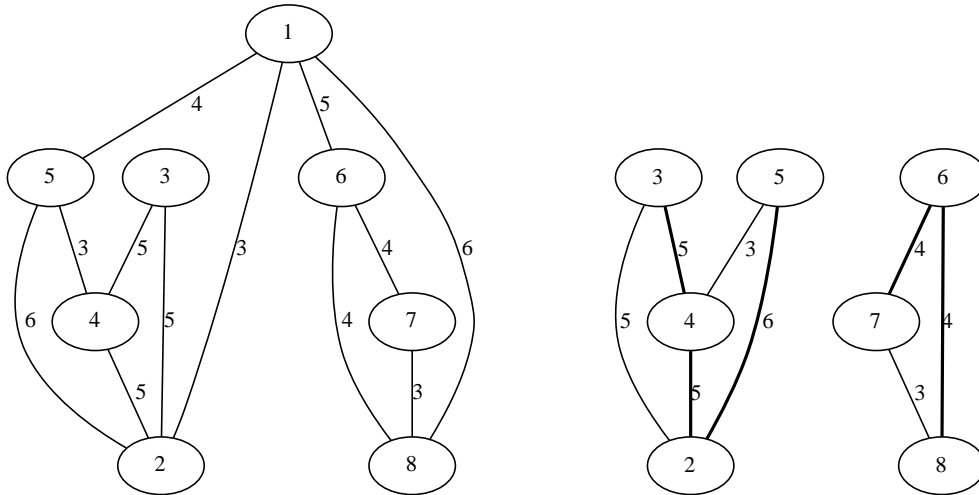


Figure 1: A simple undirected graph  $G$  and two connected components formed when  $v_0$  is removed

After the removal of node  $v_0$ , we are to find the maximum  $P$ -degree spanning tree. Those  $P$  components are independent upon each other, and no degree or any other restriction is laid upon any node in regard to MST. It's easy to obtain the MST for each connected component: Prim and Kruskal algorithms are ideal candidates for this job.

Now we must use edges in  $E(v_0)$  to connect  $v_0$  to every MST we've obtained for each component in order to get a spanning tree for the whole graph. Suppose that the nodes of one connected component form the set  $V_c$ , we can choose any edge  $e \in E(v_0, V_c)$  and use

it to connect  $v_0$  and the corresponding connected component. In order to maximize the total weight of our spanning tree, we must choose the edge with maximum weight wherever possible. *Figure 2* shows the maximum  $P$ -degree spanning tree of the graph shown in *Figure 1*.

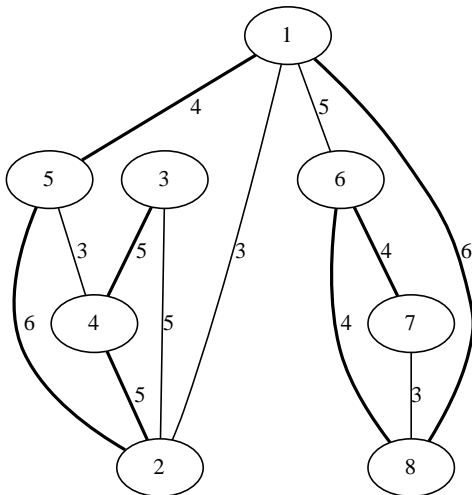


Figure 2: Bold edges form the maximum  $P$ -degree spanning tree of the graph left in *Figure 1*

In the *for-loop* that follows, we take the maximum  $I$ -degree spanning trees where  $P < I \leq K$  into consideration. Each time, since we have already obtained the maximum  $(I - 1)$ -degree spanning tree, we try to obtain the maximum  $I$ -degree spanning tree by using one more edge in  $E(v_0)$  and adjusting the rest of the tree accordingly. No matter which additional edge we choose from  $E(v_0)$ , a circle will form. Thus, another edge of the circle has to be removed. Since removing any edge of that circle would work, we can just delete the one with the least weight in order to maximize the total weight left. We can enumerate the edges in  $E(v_0)$ , choose the one that is not used in the maximum  $(I - 1)$ -degree spanning tree as  $e_{new}$ , and remove the least weighted edge  $e_{remove}$  of the circle formed by adding edge  $e_{new}$  to obtain an  $I$ -degree spanning tree with total weight  $W_{I-1} + weight(e_{new}) - weight(e_{remove})$ . This process is regarded as an **edge-exchange operation**. After attempting all choices of  $e_{new}$  available, we choose the one that maximizes the total weight of the new  $I$ -degree spanning tree. In other words, we perform the **maximum edge-exchange operation** that maximizes  $weight(e_{new}) - weight(e_{remove})$  to get the maximum  $I$ -degree spanning tree. Check if the maximum  $I$ -degree spanning tree is greater than  $R$  and update  $R$  if necessary.  $R$  stores exactly the expected MSDCST when our algorithm completes.

*Figure 3* shows the MSDCST of the graph in *Figure 1*.

## 4 Efficient Implementation

To achieve the expected efficiency, we have to implement this algorithm with carefully chosen data structures.

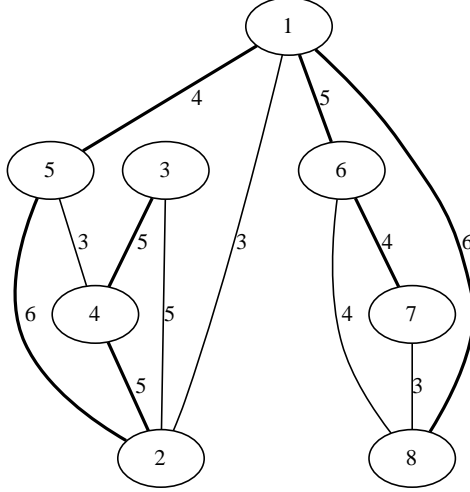


Figure 3: Bold edges form the MSDCST of the graph left in *Figure 1*

Let's focus on the implementation of the **maximum edge-exchange operation**, which has a significant impact on the overall efficiency of the algorithm. The straightforward implementation doesn't run very fast: just iterate through all edge-exchange operations available and find the maximum one. Trying every edge in  $E(v_0)$  for  $e_{new}$  would cost  $O(|V|)$  time, and iterate through all edges of the cycle to find the least weighted edge would also cost  $O(|V|)$  time. Thus, each operation costs  $O(|V|^2)$  time. Since we need to do this for  $(K - P)$  times, the total time taken is  $O(K|V|^2)$ .

Surely we can do better. If  $v_0$  is regarded as the root of the current spanning tree, then we get a rooted tree, which has a more orderly structure. Let  $Min_i = \min\{weight(edge(a, b)) | edge(a, b) \in path(v_0, i), a \neq v_0, b \neq v_0\}$ , that is,  $Min_i$  denotes the minimum weight of edges that are not adjacent to  $v_0$  from  $path(v_0, i)$ . Calculating initial values of  $Min$  for the original maximum  $P$ -degree spanning tree can be achieved in time  $O(|V| + |E|)$ : traverse the spanning tree in Depth-First-Search order and calculate values of  $Min$  whenever we encounter a node. In a single **maximum edge-exchange operation**, we iterate through all candidates for  $e_{new}$ . Each time we choose an edge, say  $edge(v_0, i)$ , the circle formed by adding  $edge(v_0, i)$  consists of all edges from the original  $path(v_0, i)$  and the newly added  $edge(v_0, i)$ . Obviously, the weight of  $e_{remove}$  is precisely  $Min(i)$ . By defining  $Min(i)$  the minimum weight of all edges on the path except the edge directly connected to  $v_0$ , we make sure that we don't remove the edge adjacent to  $v_0$ , therefore ensuring that we indeed increase the degree of  $v_0$  by one. After that, we need to update the values of  $Min$  since the spanning tree changes after each operation. Since  $edge(v_0, i)$  has been added and an edge from  $path(v_0, i)$  has been removed, only the values of  $Min(v)$ , where  $v$  belongs to this subtree, are changed. Consequently, we only need to travel across this very subtree and modify the values accordingly. This maintenance process also costs  $O(|V| + |E|)$  time. Hence, the total time complexity for this step is  $O(K(|V| + |E|))$ .

Overall, this algorithm has a time complexity of  $O(|E|\log|E| + K(|V| + |E|))$  and a space complexity of  $O(|V| + |E|)$ .

## References

- [1] Fred Glover and Darwin Klingman. Finding minimum spanning trees with a fixed number of links at a node. *Combinatorial Programming: Methods and Applications. Proceedings of the NATO Advanced Study Institute*, pages 191-201. D. Reidel Publishing Co., 1974.
- [2] A. Volgenant. A Lagrangian approach to the degree-constrained minimum spanning tree problem. *European Journal of Operational Research*, 39:325-331, 1989.
- [3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, 2/e. MIT Press and McGraw-Hill, 2001.