

Instituto de Engenharia de Sistemas e Computadores de Coimbra
Institute of Systems Engineering and Computers
INESC – Coimbra

João Clímaco, Marta Pascoal and Carlos Gomes da Silva

Some computational improvements on finding the K shortest spanning trees

No. 7

2008

ISSN: 1645-2631

Instituto de Engenharia de Sistemas e Computadores de Coimbra
INESC – Coimbra
Rua Antero de Quental, 199; 3000 - 033 Coimbra; Portugal
www.inescc.pt

SOME COMPUTATIONAL IMPROVEMENTS ON FINDING THE K SHORTEST SPANNING TREES

JOÃO C. N. CLÍMACO^(1,2), MARTA M. B. PASCOAL^{(2,3)*}, and CARLOS GOMES DA SILVA^(2,4)

⁽¹⁾ Faculdade de Economia, Universidade de Coimbra, Avenida Dias da Silva, 165, 3004-512 Coimbra, Portugal

⁽²⁾ Instituto de Engenharia de Sistemas e Computadores – Coimbra, Rua Antero de Quental, 199, 3000-033 Coimbra, Portugal

E-mail: jclimaco@inescc.pt

⁽³⁾ Departamento de Matemática, Universidade de Coimbra, Apartado 3008, 3001-454 Coimbra, Portugal

E-mail: marta@mat.uc.pt

⁽⁴⁾ Escola Superior de Tecnologia e Gestão de Leiria, Instituto Politécnico de Leiria, 1749-016 Leiria, Portugal

E-mail: cgsilva@estg.ipleiria.pt

June 2008

Abstract: A key computational step in the K shortest spanning tree problem is the procedure to obtain the second best. In this paper we describe and implement several methods for the determination of the second shortest tree on a network. The methods use different strategies for reaching the best swap of edges that leads to the second best spanning tree. Several fathoming conditions are considered in order to prevent useless calculations. Computational experiments are presented and results are analysed for randomly generated networks.

Keywords: Spanning trees, ranking algorithms, networks.

1 Introduction

A spanning tree of a given network, $(\mathcal{N}, \mathcal{A})$, is formed by the set of edges that guarantees the connectivity among all network vertices. For this reason the spanning trees determination leads to specially many interesting problems that have been studied for about 50 years now [4, 5]. One of the main research topics considers that a real value is associated with each network edge, c_{ij} , for $\{i, j\} \in \mathcal{A}$, and the goal is to obtain the spanning tree with a minimum total cost, which is given by the summation of all edge costs, $c(T) = \sum_{\{i,j\} \in T} c_{ij}$, with T a spanning tree. The calculation of spanning trees besides the best one, according to non-decreasing order of cost, can be used, for instance, in finding alternative solutions, sensitivity analysis or spanning tree problems involving more than one objective function.

The ranking of spanning trees was studied by Gabow, by Katoh, Ibaraki and Mine and by Eppstein [1, 2, 3], who proposed algorithms with worst-case time complexity of $\mathcal{O}(Km)$, of $\mathcal{O}(Km)$ and of $\mathcal{O}(Km)$, respectively, if K denotes the number of solutions to be listed. The efficiency in

*Corresponding author

terms of time and space and the implementation of these techniques are major aspects to be taken into account, specially for certain types of applications, given the high number of spanning trees that even a small size network may contain (n^{n-2} in the complete case).

So far the computational aspects of the determination of more than one spanning tree have been set aside, as the three papers mentioned above concern only theoretical aspects of the determination of the K minimum cost spanning trees and, to the best of our knowledge, there is no study in the literature about their implementation or empirical evaluation of their performances. This paper focus on determining the second best tree, once the best one is known, discusses several methods for that problem and compares them from a computational point of view. In the next Section the procedure proposed by Gabow is reviewed and methods and further tests for making use of the information obtained along its steps, aiming to prevent useless actions, are introduced. These methods are illustrated by a small example, and in Section 3 computational results on random instances are then reported and discussed. Conclusions and ongoing research are drawn in Section 4.

2 The second shortest spanning tree

As mentioned above, the algorithms by Gabow [2] and by Katoh, Ibaraki and Mine [3] deal with the K best spanning trees problem. These algorithms work in a similar way, by finding the best spanning tree and then obtaining the next solution by means of computing the minimum cost edge exchange, which is an exchange of an edge in the tree with an edge not in the tree which gives the minimum possible increase of the tree cost. The two algorithms differ on the details about how to maintain the partition and how to select the edge exchanges.

In the following we assume the antecessor, predecessor and cost of any edge in the network to be known. Let T^* be the minimum cost spanning tree. The second best spanning tree results from T^* by swapping one of its edges by another one that doesn't belong to T^* , and choosing the minimum cost of these swaps. Different manners of finding the best swap can be derived, in the following we will discuss some of them, aiming to reduce those repeated scans of the edges in T^* and/or edges in \mathcal{A} but not in T^* .

The problem can be viewed in two different ways, we can either choose which non-tree edge to include in T^* , or else choose the tree edge that should be replaced in T^* . Methods 1 and 2 explore the first idea, while Methods 3 and 4 follow the second. Before describing these methods we present the auxiliary routine given in Procedure 1, which will be used for storing the information relative to an exchange of an edge by a better one.

Procedure 1 *Exchange edges $\{x_1, x_2\}$ and $\{y_1, y_2\}$*

BestSwap $\leftarrow c_{x_1x_2} - c_{y_1y_2}$
NewEdge $\leftarrow \{x_1, x_2\}$
OldEdge $\leftarrow \{y_1, y_2\}$

Moreover we will consider the edges in T^* and the information about the *Father* and *Level* of each vertex to be known. Arrays *Father* and *Level* are obtained by traversing T^* level by level, that is breadth-first, considering any of the vertices as the root of the tree. $Father_i$ represents the vertex that precedes i in T^* according to that traversal.

For example, given the spanning tree $T^* = \{\{1, 2\}, \{2, 4\}, \{1, 3\}, \{3, 5\}, \{4, 6\}\}$, and choosing vertex 1 to be the root of the tree, we have the array *Father* = [0 1 1 2 3 4]. The *Level* array is defined as

$$Level_1 = 0, \quad Level_x = Level_{Father_x} + 1,$$

thus we have *Level* = [0 1 1 2 2 3].

These arrays are used to efficiently find the edges in the circuit created by adding a non-tree edge $\{x, y\}$. The edges in the created circuit are found in the following manner. Firstly, we start by backtracking in the tree from the vertex with the highest level, using the *Father* array until the vertex has the same level, and then, when the vertices have the same level, the antecessor of each vertex is recursively identified until a common antecessor is found. The considered edges belong to the circuit. For instance, suppose that the edge $\{5, 6\}$ is added to T^* . As $Level_5 < Level_6$, the antecessor of vertex 6 is identified: $Father_6 = 4$. Now, $Level_4 = Level_5$. As $Father_4 = 2 \neq Father_5 = 3$, the backtracking process continues. Now, $Father_2 = Father_3 = 1$, and the circuit was identified: $\{\{5, 6\}, \{4, 6\}, \{2, 4\}, \{3, 5\}, \{1, 2\}, \{1, 3\}\}$.

Method 1 Adding a new edge to a spanning tree forms a circuit, the edges of which need to be scanned in order to find out the best exchange for the new one. The steps for a simple way of accomplishing this task are summarised in the following algorithm.

```

BestSwap  $\leftarrow +\infty$ 
For  $\{i, j\} \in \mathcal{A} - T^*$  Do
   $x \leftarrow i, y \leftarrow j$ 
  If  $Level_x > Level_y$  Then Exchange  $x$  and  $y$ 
  While  $Level_x < Level_y$  Do
    If  $BestSwap > c_{y, Father_y} - c_{ij}$  Then Exchange edges  $\{y, Father_y\}$  and  $\{i, j\}$ 
     $y \leftarrow Father_y$ 
  EndWhile
  While  $x \neq y$  Do
    If  $BestSwap > c_{x, Father_x} - c_{ij}$  Then Exchange edges  $\{x, Father_x\}$  and  $\{i, j\}$ 
    If  $BestSwap > c_{y, Father_y} - c_{ij}$  Then Exchange edges  $\{y, Father_y\}$  and  $\{i, j\}$ 
     $x \leftarrow Father_x, y \leftarrow Father_y$ 
  EndWhile
EndFor
 $T' \leftarrow T^* - \{OldEdge\} \cup \{NewEdge\}$ 

```

In this method all the non-tree edges $\{i, j\}$ are scanned and are tested to replace an edge in the single path between vertices i and j .

This method is straightforward when non-preprocessing is required to fathom the scanning of the best swap.

Method 2 Gabow [2] proposed an algorithm based on Method 1 and included a set of tests in order to avoid to scan the edges more than once. Assuming the non-tree edges are chosen by non-decreasing order of costs (which is not a strong assumption if T^* is computed by Kruskal algorithm), then given an edge $\{x, y\} \in T^*$ the first exchange that is found for $\{x, y\}$ is the best, thus it is not necessary to try to exchange $\{x, y\}$ again. Gabow proposes the use of disjoint set operations to mark the edges that have been analysed. In the following we present a more detailed approach to implement computationally that purpose.

```

For  $i \in \mathcal{N}$  Do  $Mark_i \leftarrow False$ 
BestSwap  $\leftarrow +\infty$ 
For  $\{i, j\} \in \mathcal{A} - T^*$  Do
   $x \leftarrow$  first unmarked vertex before  $i, y \leftarrow$  first unmarked vertex before  $j$ 
  If  $Level_x > Level_y$  Then Exchange  $x$  and  $y$ 
  While  $Level_x < Level_y$  Do

```

```

    If  $BestSwap > c_{y, Father_y} - c_{ij}$  Then Exchange edges  $\{y, Father_y\}$  and  $\{i, j\}$ 
    Marky  $\leftarrow$  True,  $y \leftarrow Father_y$ 
EndWhile
While  $x \neq y$  Do
    If  $BestSwap > c_{x, Father_x} - c_{ij}$  Then Exchange edges  $\{x, Father_x\}$  and  $\{i, j\}$ 
    If  $BestSwap > c_{y, Father_y} - c_{ij}$  Then Exchange edges  $\{y, Father_y\}$  and  $\{i, j\}$ 
    Markx  $\leftarrow$  True,  $x \leftarrow Father_x$ 
    Marky  $\leftarrow$  True,  $y \leftarrow Father_y$ 
EndWhile
EndFor
 $T' \leftarrow T^* - \{OldEdge\} \cup \{NewEdge\}$ 

```

In this algorithm the first vertex to analyse should be the one which is closer to the root and has not been scanned yet. This routine is summarised in the following.

Function 1 *First unmarked vertex before i*

```

 $x \leftarrow i$ 
While ( $Father_i \neq 0$  and  $Father_i$  is unmarked) Do  $i \leftarrow Father_i$ 
Return  $i$ 

```

We now focus on an alternative way of dealing with the determination of the second best spanning tree could be to find the best exchange for an edge that belongs to the best tree. First we analyse how the problem can be solved following such a procedure, the procedure is then modified in order to reduce the number of scanned edges as well as the number of performed operations, which is done by setting an order for examining the edges in the tree that can be replaced and the edges that do not belong to the tree that can be inserted. The simplest version is described below as Method 3 and a more selective version is presented as Method 4.

When trying to exchange an edge in T^* , excluding it originates two connected components, and an additional edge is needed in order to find the next spanning tree. The new edge is a bridge between the two components, that is, each of its extreme vertices belongs to different components.

Method 3 For each edge $\{i, j\}$ in T^* the two vertex subsets determined by the connected components obtained after deleting $\{i, j\}$ should be identified. Then the edges that link the two components are scanned in order to analyse the possible exchanges for $\{i, j\}$.

Method 4 Two types of tests can be designed in order to reduce the number of operations, one focused on the number of scanned edges and another focused on the number of vertices that have to be identified every time a new partition of T^* is considered.

- The best exchange is expected to be given by the introduction of a cheap edge and the deletion of an expensive edge. Assume that the edges in T^* are scanned by non-increasing order of cost (like before this should not bring any additional cost), and for each one the outer edges are examined by non-decreasing order of cost. When scans are made by order of cost, the sequence of edge costs to be inserted in T^* increases, therefore the first bridge between the two components of T^* gives the cheapest exchange for the edge in T^* . Moreover, the search for the best exchange for each $\{i, j\}$ in T^* can be halted as soon as $c_{xy} - c_{ij} \leq BestSwap$, for any $\{x, y\}$ not in T^* .

- Another way of reducing the number of operations is oriented towards the identification of the vertices in each connected component obtained by the deletion of one intree edge. Assuming again that edges are analysed by order of cost, only the vertices that can be reached starting at any of the two extremes of the deleted edge by means of unscanned edges need to be marked as the extreme of the new edge to be inserted – Figure 1.

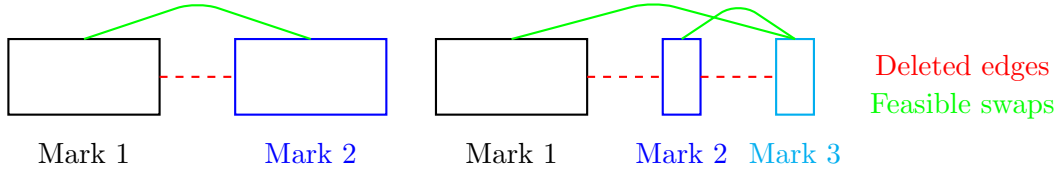


Figure 1: Connected components obtained in 2 edge deletions in T^* and possible exchanges

```

For  $i \in \mathcal{N}$  Do  $Mark_i \leftarrow \text{False}$ 
 $BestSwap \leftarrow +\infty$ 
 $MarkValue \leftarrow 0$ 
For  $\{i, j\} \in T^*$  Do /* by non-increasing order of cost */
     $MarkValue \leftarrow MarkValue + 1$ 
    If  $Level_i < Level_j$  Then  $a \leftarrow i$ 
    Else  $a \leftarrow j$ 
    For  $x$  reachable from  $a$  by unscanned edges in  $T^*$  Do  $Mark_x \leftarrow MarkValue$ 
     $\{x, y\} \leftarrow$  first edge in  $\mathcal{A} - T^*$  such that  $Mark_x$  or  $Mark_y$ , but not both, have  $MarkValue$ 
    If  $BestSwap > c_{xy} - c_{ij}$  Then Exchange edges  $\{x, y\}$  and  $\{i, j\}$ 
EndFor
 $T' \leftarrow T^* - \{OldEdge\} \cup \{NewEdge\}$ 

```

Using sensitivity analysis – see Tarjan [6] – an alternative procedure can be proposed.

Method 5 Let $C(\{x, y\})$ represent the set of edges in T^* in the circuit obtained by adding the non-tree edge $\{x, y\}$ and θ_{ij} be a non-negative constant added to the cost of edge $\{i, j\} \in C(\{x, y\})$. The tree T^* remains optimum while

$$c_{xy} \geq c_{ij} + \theta_{ij}, \text{ for any } \{i, j\} \in C(\{x, y\}).$$

Then the second best tree can be obtained by solving the linear problem:

$$\begin{aligned}
& \max \quad \theta \\
& \text{s. t.} \quad c_{xy} \geq c_{ij} + \theta_{ij}, \quad \{i, j\} \in C(\{x, y\}), \quad \{x, y\} \in \mathcal{A} - T^* \\
& \quad \quad \theta \leq \theta_{ij}, \quad \{i, j\} \in C(\{x, y\}), \quad \{x, y\} \in \mathcal{A} - T^* \\
& \quad \quad \theta \geq 0
\end{aligned}$$

The optimal solution of this linear problem is

$$\theta^* = \min_{\{x, y\} \in \mathcal{A} - T^*} \min_{\{i, j\} \in C(\{x, y\})} \{c_{xy} - c_{ij}\}, \quad (1)$$

which must be equal to the value of $BestSwap$ in the presented methods and enables the identification of the edge that must be inserted and the edge that must be removed from T^* in order to

obtain the second best tree. A possible implementation of expression (1) consists of analysing all the non-tree edges and looking for the best exchange, as is done in the `While` loop in Method 1, therefore the computational experiments below will consider only Method 1.

The above linear problem can also be used to find the K best trees from T^* . Indeed, by iteratively solving the above problem obtained from the original one by removing step-by-step a binding constraint in the previous resolutions, one obtains a rank of the incremental costs on the best tree: $\theta^* = \theta^1 \leq \theta^2 \leq \dots \leq \theta^K$.

Computational improvements The analysis of edges by order of cost in Gabow’s algorithm can be used to bound the additional cost an exchange can provoke. In fact, while trying to insert cheaper edges in the spanning tree, if an exchange for an edge with the same cost as the most expensive in T^* is found, then the best swap has been determined. In Method 2 those nodes for which an exchange has been found are marked, this information can be used to stop the method after all the n nodes in the network have been marked. Still another possible improvement, applicable to any of the methods above, is motivated by the fact that networks with a large number of edges often contain many spanning trees with the same cost. Therefore it’s likely to find a 0 cost swap very early and, whenever that happens, the process of computing the second best spanning tree can be halted.

We conclude with a short example of the application of Methods 1, 2 and 4.

Example Let $(\mathcal{N}, \mathcal{A})$ be the network in Figure 1, and the minimum cost spanning tree T^* , marked by solid lines.

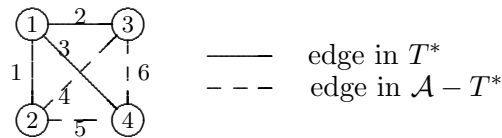


Figure 2: Network

Assuming none of the extra tests have been added to the methods above, Method 1 looks for edges to switch with the non-tree edges $\{2,3\}$, $\{2,4\}$ and $\{3,4\}$.

OldEdge	NewEdge	Additional cost	NewEdge	Additional cost
$\{2,3\}$	$\{1,2\}$	3	$\{1,3\}$	2
$\{2,4\}$	$\{1,2\}$	4	$\{1,4\}$	2
$\{3,4\}$	$\{1,3\}$	3	$\{1,4\}$	4

The best option is then to remove $\{1,3\}$ and insert $\{2,3\}$. Method 2 avoids trying to swap $\{2,4\}$ by $\{1,2\}$ and $\{3,4\}$ by $\{1,3\}$ and $\{1,4\}$. It produces the same solution as before. Finally, Method 4 looks for exchanges of the tree edges $\{1,2\}$, $\{1,3\}$ and $\{1,4\}$.

OldEdge	NewEdge	Additional cost
$\{1,4\}$	$\{2,4\}$	2
$\{1,3\}$	$\{2,3\}$	2
$\{1,2\}$	$\{2,3\}$	3

deciding for the exchange between $\{1,4\}$ and $\{2,4\}$.

3 Computational experiments

In this section the results of computational experiments carried out with the methods described in the previous section are presented. Instances of $n \in \{500, 1000, 5000, 10000\}$ vertices and average degree $d \in \{5, 10, 20\}$ (where $m = dn$) were considered. Graphs were randomly generated and integer costs were uniformly calculated between 0 and 10000. These instances are available at <http://www.mat.uc.pt/~marta/Research/Downloads/2MST.tgz>. The tests ran on a Dual Core AMD Opteron with a 1 GHz processor, 1 MB of cache and 4 Gb of RAM, running over SUSE Linux 9.3

	$d = 5$	$d = 10$	$d = 20$	$d = 5$	$d = 10$	$d = 20$
M1	33979	73056	161996	86011	182951	396180
M2	30645	65163	144897	79108	166589	360950
M1B	24059	32913	40281	66628	92722	111978
M2B	18629	32912	21768	43609	53117	44591
M4	6	6	7	7	8	9

Table 1: Number of attempts to exchange edges in random networks: $n = 500$ on the left, $n = 1000$ on the right

The methods described in Section 2 have been coded in C language and their performance has been compared. Those implementations are about methods 1, 2 and 4, M1, M2 and M4, respectively. An additional test to halt the procedure M1 and M2, when the incremental cost of the new spanning tree is small “enough” to give the best exchange, was considered. With this test, the methods are named M1B and M2B, respectively.

	$d = 5$	$d = 10$	$d = 20$	$d = 5$	$d = 10$	$d = 20$
M1	0.0021	0.0036	0.0087	0.0032	0.0077	0.0200
M2	0.0019	0.0055	0.0084	0.0048	0.0093	0.0207
M1B	0.0015	0.0019	0.0025	0.0056	0.0051	0.0079
M2B	0.0012	0.0015	0.0005	0.0029	0.0041	0.0027
M4	0.0144	0.0132	0.0132	0.0705	0.0674	0.0648

Table 2: CPU times (seconds) in random networks: $n = 500$ on the left, $n = 1000$ on the right

Some of the stopping conditions proposed in the last section were not implemented. We tried to compare the performance of the methods in terms of execution time for examining T^* and computing T' , impact of inserting the described improvements as well as the number of edges that are considered as possible candidates to an exchange, and its reaction to the increase in the size of instances.

	$d = 5$	$d = 10$	$d = 20$	$d = 5$	$d = 10$	$d = 20$
M1	673889	1455790	3220163	1664243	3599654	7526001
M2	637875	1370859	3037503	1591661	3428673	7157795
M1B	597862	887080	1041349	1532348	2415068	2841103
M2B	143003	70276	8478	101842	7447	3864
M4	9	11	13	12	13	17

Table 3: Number of attempts to exchange edges in random networks: $n = 5000$ on the left, $n = 10000$ on the right

For many of the generated networks the number of edges exceeded the costs range, making it likely that two solutions with the same cost exist. Checking the occurrence of a 0 cost exchange would then halt the procedure too soon to evaluate the algorithms themselves, therefore this test was omitted in the codes. We also excluded counting the number of scanned vertices by Methods 1 and 2, to get an idea of the influence of n and m on the results.

	$d = 5$	$d = 10$	$d = 20$	$d = 5$	$d = 10$	$d = 20$
M1	0.0352	0.0745	0.1713	0.0901	0.1915	0.4077
M2	0.0367	0.0809	0.1780	0.0960	0.2112	0.4456
M1B	0.0369	0.0491	0.0567	0.0891	0.1341	0.1620
M2B	0.0081	0.0044	0.0005	0.0068	0.0003	0.0003
M4	2.4863	2.4144	2.3543	12.2542	11.5604	10.9699

Table 4: CPU times (seconds) in random networks: $n = 5000$ on the left, $n = 10000$ on the right

The tables present average results over 30 networks of the same size and concern the scan of best tree, T^* , and the computation of the second best, T' . The set of edges in the network is assumed to be sorted in advance. Notice that this is not a demand of Method 1, but even considering sorting time the relative comparison of the methods was the same as the presented above.

Concerning the number of times a cheaper exchange is tried - Tables 1 and 3 - the results matched the expectations, as it decreases as versions include more tests to avoid repeated operations, namely when marking the edges for which the best exchange is already know - M2 - and then also analysing the cost of the exchanges - MB1 and MB2.

The observed CPU times - Tables 2 and 4 - did not behave according to those numbers and, in fact, M1 outperformed M2 in most of the cases, which is probably due to the increase of tests that need to be done in order to avoid swaps. This was also the case for M4. Although the number of exchanges was much smaller for this case each one implies identifying the vertices of a partition and then looking for the first edge in set $\mathcal{A} - T^*$ that satisfies certain conditions. This is done $n - 1$ times, while for the remaining methods it takes at most $m(n - 1)$ iterations, however finding the right edge in $\mathcal{A} - T^*$ is usually more expensive than comparing it with some of the tree edges, therefore even though many operations have been avoided in these procedure we have concluded that trying to insert new edges in the best tree to be more efficient than trying to replace the intree edges.

It is also worth noticing the fact that method M2B is not sensitive to the increase on the number of edges. Indeed, with the fathom condition imposed in the method it is only required the analysis of a small fraction of edges with costs greater than the greatest cost of T^* . As mentioned before the existence of several edges with the same cost also accelerate the interruption of the method.

To end with, it should be added that the inclusion of the cost exchange bound always led to better results and that amongst all the implemented versions the most promising one was MB2.

4 Conclusions

In the paper several methods were compared concerning the CPU times and number of attempts to obtain the second best spanning tree. The data structures used to represent the networks and cycle detection with the insertion of a new edge in T^* , revealed to be critical for the performance of the methods.

The methods rely on some well-known properties of the spanning tree namely the path optimality condition and the cut optimality condition. Additional tests were considered to avoid non-promising operations. It was found that some naive searches (like Method 1) for the second

best spanning tree outperform more sophisticated procedures. This is a balance to always have into account.

Although the proposed procedures are adequate to be inserted in the classical K shortest spanning tree algorithms, we hope our proposals can help in the development of new improvements concerning the global procedure.

References

- [1] D. Eppstein. Finding the k smallest spanning trees. In *Proc. 2nd Scandinavian Worksh. Algorithm Theory*, number 447 in Lecture Notes in Computer Science, pages 38–47. Springer-Verlag, July 1990.
- [2] H. N. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM Journal on Computing*, 6:139–150, March 1977.
- [3] N. Katoh, T. Ibaraki, and H. Mine. An algorithm for finding K minimum spanning trees. *SIAM Journal on Computing*, 10(2):211–247, 1981.
- [4] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
- [5] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
- [6] R. E. Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *Information Processing Letters*, 14(1):30–33, 1982.