Matemática Contemporânea, Vol. 45, 97–105
http://doi.org/10.21711/231766362017/rmc4511
©2017, Sociedade Brasileira de Matemática

On the problem of finding all minimum spanning trees

João Guilherme Martinez **Rosiane de Freitas**
Altigran Silva

Abstract

A minimum spanning tree is a subgraph of an undirected weighted graph that still connects all the vertices, has no cycles and has minimum total weight. Many efficient algorithms are known to solve the problem of determining a single best solution to the problem, such as Prim's and Kruskal's algorithms. However, the problem of enumerating all the minimum spanning trees of a graph is an NP-hard problem, it has great theoretical and practical importance, but there are few algorithms to solve it. In this work, we analyze these algorithms and their properties and we make a comparison with the use of experiments. Such algorithms were analyzed according to their theoretical properties, computational complexity and implementation details, and a comparative analysis was performed using general and specific graph class instances.

1 Introduction

Let G be an undirected weighted graph with n vertices and m edges. We say that T is a spanning tree of G if it is a subgraph from G that con-

2000 AMS Subject Classification: 68R10, 68R15.
Key Words and Phrases: minimum spanning trees, graphs, algorithms.

nects all vertices and does not contain a cycle. T is a minimum spanning tree (MST) if the total weight of its edges is minimal. The problem has optimal substructure and it can be solved by greedy algorithms in almost linear time $O(m+n\log n)$ if used along with efficient data structures such as Fibonacci heaps [FT87]. However, the problem then becomes exponential to enumerate all possible MSTs, as for example in complete equal edge weight graphs. Cayley [Cay89] has proved that the number of trees generated is n^{n-2}. Figure 1 shows an example graph and all of its spanning trees.

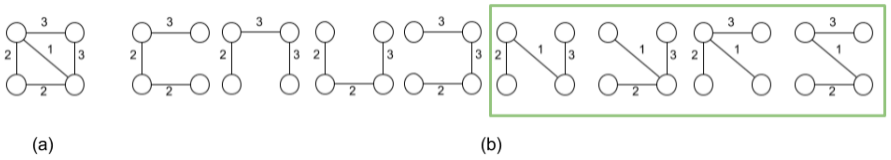

Figure 1: (a) Example G graph. (b) All STs of G with the MSTs framed.

Our work consists of a theoretical and empirical analysis on the main algorithms of the literature to solve the problem of finding all minimum spanning trees. We also verified the behavior of such algorithms in different graph classes.

2 Implemented algorithms

Let us first consider N as the number of spanning trees (ST) of a graph G, and K as the number of minimum spanning trees of the same graph.

The works from Gabow & Myers, Matsui, Kapoor & Ramesh, Shioura & Tamura provide algorithms for enumerating all spanning trees of a graph ([GM78],[Mat97],[KR95],[ST95]). These algorithms could be used as a solution to the MST enumeration problem in such a way that the non-minimum trees could be discarded at the end and only the minimum ones would remain. However, this strategy would demand a high unnecessary computational cost considering that N can be much greater than K. The algorithm proposed by Sörensen & Janssens is slightly different from the others because it outputs the trees in increasing order, which could be a more elegant solution with smaller cost considering that we could stop the enumeration as soon as the cost of the current tree gets bigger than the previous one [SJ05].

Some of the few papers that present algorithms to solve the minimum spanning trees enumeration problems ([Epp95],[YKW10],[Wri00]) have much better time complexity bounds comparing to the ones that generate only spanning trees as we can see in Table 1. These algorithms are described below.

Table 1: Algorithms for enumerating STs and MSTs and their complexities.

Algorithm	**Year**	**Return**	**Time Complexity**
Gabow & Myers	1978	All STs	$O(n + m + N.n)$
Matsui	1997	All STs	$O(n + m + N.n)$
Kapoor & Ramesh	1995	All STs	$O(n + m + N)$
Shioura & Tamura	1995	All STs	$O(n + m + N)$
Sörensen & Janssens	2005	All STs (in order)	$O(N.m \log m + N^2)$
Wright	2000	All MSTs	undefined in original paper
Yamada, Kataoka & Watanabe	2010	All MSTs	$O(K.m \log n)$
Eppstein	1995	All MSTs	$O(m + n \log n + K)$

2.1 Yamada, Kataoka and Watanabe's algorithm

Three algorithms are proposed in the paper from Yamada, Kataoka & Watanabe [YKW10], but in our paper we have implemented only the second one. The algorithm uses a combinatorial optimization framework for solving enumerating problems. It considers two edge sets, F as the fixed edges that must be in the current tree and R as the restricted edges that cannot be in the current tree.

Let T_0 be an MST from G, and $e \in T_0$, deleting e from T_0 splits the tree into two connected components V_1 and V_2, so define $\text{Cut}(e)$ as the set of edges that can substitute e and reconnect V_1 and V_2, and $S(e) := \{\tilde{e} \in$

$\text{Cut}(e) \mid \tilde{e} \neq e, w(\tilde{e}) = w(e)\}$. The algorithm is described in details in the original paper and has time complexity $O(mnK)$.

2.2 Wright's algorithm

Let T_0 be an arbitrary minimum spanning tree from G, and consider f as an edge in G that is not in T_0. Wright proves that an edge $f = (u, v)$ is part of some MST, if and only if there is an edge e in the path from u to v in T_0 with $w(e) = w(f)$ [Wri00]. This theorem is used in all other MSTs enumeration algorithms.

For the algorithm, the author defines class $[e]$ as the set of edges that can replace e; $i([e])$ as the number of edges in $[e]$ which must be included in every MST; and $c([e])$ as the number of possible choices for $[e]$. For example, if we consider $[e]$ with 4 edges and $i([e]) = 2$ than we will have 6 subsets of size 2 as $c([e]) = \binom{4}{2} = 6$.

The algorithm proposed in the paper works as follows: first it takes an MST T_0 from G as an input and delete all electable edges from G. Then it determines the classes $[e]$ and $i([e])$ by inspection in T_0 and the choices of each class. Finally, it combines theses choices, one from each class, to produce all MSTs from G. Each choice is then evaluated to be suitable by checking if the graph is both acyclic and connected.

The original paper doesn't define the time complexity of the algorithm, but here we make a simple analysis. Lets consider S as the number of subsets formed by the classes and C as the total number of arbitrary combinations of choices, one from each class. We can retrieve all non-MST edges in $O(m)$ time, get the classes of edges in $O(mn)$, form the subsets of classes in $O(S)$ and generate and check each tree in $O(Cn)$. However, we can easily see that $C \gg S$ and $C \geq K$, so replacing S by C we have $O(mn + Cn)$.

2.3 Eppstein's algorithm

The algorithm proposed by Eppstein [Epp95] has the best time complexity so far but it presents much more complex steps. Its main idea is to form an equivalent graph (EG) from G by performing *sliding operations* in a way that all spanning trees of EG are equivalent one to one with all MSTs of G. A sliding operation is defined as follows: consider edges $e = (u, v)$ and $f = (v, w)$ that share a common vertex v, and suppose that $w(e) < w(f)$; to perform a sliding operation we remove $f(v, w)$ and insert $f'(u, w)$ with the same weight as f; this operation is only done if $w(e) < w(f)$.

To form EG, choose a vertex to be the root of T_0 and repeatedly perform sliding operations through edges e and f as long as $e \in T_0$ and u is closer to the root of T_0 than v. Implemented directly this takes $O(mn)$, however, Eppstein presents a technique to achieve $O(m \log n)$ time. To do this, for any edge e from T_0, define the *heavy ancestor* of e to be the first edge on the path from e to the root having weight greater than that of e. If we slide each T_0 edge directly with its *heavy ancestor*, time complexity is reduced [Epp95]. Algorithm 1 presents the pseudocode.

Algorithm 1: Sliding(G,u,ℓ,$root$)

```
mark $u$ as visited;
for each vertex $v$ adjacent to $u$ do
    if $v$ is unvisited then
        if $u = root$ then
            $A[0] = Edge(u, v, cost(u, v))$;
            SLIDING(G,$v$,0,$root$);
            SLIDECHILDREN($u$,$v$,0);
        else
            $\ell' \leftarrow$ BINSEARCH($A$,$cost(u, v)$,0,$\ell$);
            Edge $e = A[\ell']$;
            $A[\ell'] = Edge(u, v, cost(u, v))$;
            SLIDING(G,$v$,$\ell'$,$root$);
            SLIDECHILDREN($u$,$v$,$\ell'$);
            $A[\ell'] = e$;
```

Before the sliding operations, all non-MST edges must be removed from

G. The algorithm proposed by Bazlamacci & Hindi [BH97] can perform this in $O(m)$ time by creating a full branching-tree B_T corresponding to T_0 to make use of the following property: The largest edge weight from x to y in T_0 is the same largest edge weight from x to y in B_T. Along with that, we can use the algorithm from Schieber & Vishkin [SV88] to retrieve the lowest common ancestor (lca) between any two vertices on B_T which allow us to get the heaviest edge by comparing the heaviest one from u to $\text{lca}(u, v)$ and the heaviest one from v to $\text{lca}(u, v)$.

After EG is built, Eppstein suggests the algorithm from Kapoor & Ramesh [KR95] to retrieve all spanning trees from EG, which corresponds to the MSTs of G. However, it is a complex algorithm, so we decided to implement the algorithm from Shioura & Tamura [ST95] that has the same time complexity but much simpler description. Combining all the previous steps, the total time complexity is $O(m + n \log n + K)$.

3 Comparative computational experiments

The algorithms were implemented in C++[1] and the experiments were executed on an Intel i5, 8GB RAM, Ubuntu 16.04 64 bits machine and based on the experiments from Yamada, Kataoka & Watanabe [YKW10]. We used complete and arbitrary random graphs, but in the random ones we used a fixed amount of edges ($m = 3n$). About the edge weight distribution, the experiments were divided in two categories: equal edge weight graphs and random edge weight graphs varying randomly between $[1, 10^L]$ (for $L = 2$ and $L = 3$).

4 Concluding remarks

By the analysis of the experimental results we could confirm that Eppstein's algorithm [Epp95] has the best performance for all graph classes, although it is more complex than the others. It is possible to notice that

[1]Code available at: http://home.ufam.edu.br/joao/msts.zip

for equal weight edge graphs, Wright's algorithm [Wri00] is slower than the one from Yamada, Kataoka & Watanabe [YKW10], most likely because of the high number of combination choices made. We have also confirmed Cayley's formula for complete graphs with equal weight edges, as shown in Table 2a.

Table 2: Comparative results using the execution time of the algorithms for some graph classes in seconds.

Graph	**MSTs**	**Wright**	**YKW**	**Eppstein**
K5	125	0,003	0,002	0,000
K6	1269	0,054	0,029	0,006
K7	16807	1,188	0,434	0,082
K8	262144	31,002	7,5482	1,335
K9	4782969	960,126	152,972	23,957
K10	100000000	*	3541,48	502,631

(a) Results for complete graphs with the same weight. * huge execution time.

L	**Vertices**	**MSTs**	**Wright**	**YKW**	**Eppstein**
2	200	16	0,078	0,087	0,011
	400	48	0,350	1,720	0,026
	600	2048	6,467	76,091	0,042
	800	6144	23,276	230,59	0,06
3	200	1	0,065	0,051	0,011
	400	1	0,269	0,051	0,011
	600	2	0,614	0,755	0,042
	800	2	1,116	1,429	0,061

(b) Results for arbitrary graphs with $m = 3n$ with random weight edges.

L	**Graph**	**MSTs**	**Wright**	**YKW**	**Eppstein**
2	K20	4	0,004	0,001	0,001
	K40	120	0,046	0,056	0,006
	K60	176580	69,633	209,633	0,077
	K80	5971968	*	4218,73	0,167
3	K20	1	0,003	0,000	0,001
	K40	1	0,030	0,003	0,006
	K60	12	0,102	0,039	0,014
	K80	2	0,234	0,016	0,024
	K100	6	0,473	0,050	0,039
	K120	8	0,816	0,169	0,057
	K140	256	1,432	3,992	0,081
	K160	1152	2,681	8,282	0,104
	K180	208	2,902	1,940	0,133
	K200	1152	4,830	16,314	0,167

(c) Results for complete graphs with random weight edges. * huge execution time.

Finding all MSTs is not trivial as the number of solutions (K) can be exponential. As ongoing work, we are working in an algorithm that mixes characteristics of Eppstein's and YKM's algorithms, in order to guarantee a competitive performance but having the most complex step simplified.

References

[BH97] C. F. Bazlamacci and K. S. Hindi. Verifying minimum spanning trees in linear time. In *Symposium on Operations Research*, pages 139–144, Heidelberg, September 1997. Springer-Verlag.

[Cay89] Arthur Cayley. A theorem on trees. *Quart. J. Math*, pages 376–378, 1889.

[Epp95] David Eppstein. Representing all minimum spanning trees with applications to counting and generation. Technical Report 95-50, Univ. of California, Irvine, Dept. of Information and Computer Science, Irvine, CA, 92697-3425, USA, 1995.

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, July 1987.

[GM78] Harold N. Gabow and Eugene W. Myers. Finding All Spanning Trees of Directed and Undirected Graphs. *SIAM Journal on Computing*, 7(3):280–287, August 1978.

[KR95] Sanjiv Kapoor and H. Ramesh. Algorithms for enumerating all spanning trees ofundirected and weighted graphs. *SIAM J. Comput.*, 24(2):247–265, April 1995.

[Mat97] T. Matsui. A Flexible Algorithm for Generating All the Spanning Trees in Undirected Graphs. *Algorithmica*, 18(4):530–543, August 1997.

[SJ05] Kenneth Sorensen and Gerrit K. Janssens. An algorithm to generate all spanning trees of a graph in order of increasing cost. *Pesquisa Operacional*, 25:219 – 229, 08 2005.

[ST95] Akiyoshi Shioura and Akihisa Tamura. Efficiently scanning all spanning trees of an undirected graph. *J. Operation Research Society Japan*, 38:331–344, 1995.

[SV88] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. In *AWOC*, volume 319 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 1988.

[Wri00] Perrin Wright. Counting and constructing minimal spanning trees, 2000.

[YKW10] Takeo Yamada, Seiji Kataoka, and Kohtaro Watanabe. Listing all the minimum spanning trees in an undirected graph. *Int. J. Comput. Math.*, 87(14):3175–3185, 2010.

João Guilherme Martinez
PPGI/IComp-UFAM
Manaus - Amazonas, Brazil
joaogam@icomp.ufam.edu.br

Rosiane de Freitas
PPGI/IComp - UFAM
Manaus - Amazonas, Brazil
rosiane@icomp.ufam.edu.br

Altigran Silva
PPGI/IComp - UFAM
Manaus - Amazonas, Brazil
alti@icomp.ufam.edu.br