

available at www.sciencedirect.comjournal homepage: www.elsevier.com/locate/cosrev

Research paper

The saga of minimum spanning trees

Martin Mareš*

Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha, Czech Republic
 Institute of Theoretical Computer Science,¹ Charles University, Malostranské nám. 25, 118 00 Praha, Czech Republic

ABSTRACT

This article surveys the many facets of the Minimum Spanning Tree problem. We follow the history of the problem since the first polynomial-time solution by Borůvka to the modern algorithms by Chazelle, Pettie and Ramachandran. We study the differences in time complexity dependent on the model of computation chosen and on the availability of random bits. We also briefly touch the dynamic maintenance of the MST and other related problems.

© 2008 Elsevier Inc. All rights reserved.

1. Minimum spanning trees

1.1. The problem

The problem of finding a minimum spanning tree of a weighted graph is one of the best studied problems in the area of combinatorial optimization since its birth. Its colorful history (see [1,2] for the full account) begins in 1926 with the pioneering work of Borůvka [3],² who studied primarily an Euclidean version of the problem related to planning of electrical transmission lines (see [5]), but gave an efficient algorithm for the general version of the problem. As it was well before the dawn of graph theory, the language of his paper was complicated, so we will better state the problem in contemporary terminology:

Problem 1.1.1. Given an undirected graph G with weights $w : E(G) \rightarrow \mathbb{R}$, find its minimum spanning tree, defined as follows:

Definition 1.1.2. For a given graph G with weights $w : E(G) \rightarrow \mathbb{R}$:

- A subgraph $H \subseteq G$ is called a *spanning subgraph* if $V(H) = V(G)$.
- A *spanning tree* of G is any spanning subgraph of G that is a tree.
- For any subgraph $H \subseteq G$ we define its *weight* $w(H) := \sum_{e \in E(H)} w(e)$. When comparing two weights, we will use the terms *lighter* and *heavier* in the obvious sense.
- A *minimum spanning tree (MST)* of G is a spanning tree T such that its weight $w(T)$ is the smallest possible among all the spanning trees of G .
- For a disconnected graph, a (*minimum*) *spanning forest (MSF)* is defined as a union of (minimum) spanning trees of its connected components.

Borůvka's work was further extended by Jarník [6], again in a mostly geometric setting. He has discovered another efficient algorithm. However, when computer science and graph theory started forming in the 1950s and the spanning tree problem was one of the central topics of the flourishing new disciplines, the previous work was not well known and the algorithms had to be rediscovered several times.

* Corresponding address: Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha, Czech Republic.
 E-mail address: mares@kam.mff.cuni.cz.

¹ Partially supported by grant 1M0021620808 of the Czech Ministry of Education.

² See [4] for an English translation with commentary.

In the next 50 years, several significantly faster algorithms were discovered, ranging from the $\mathcal{O}(m\beta(m, n))$ time algorithm by Fredman and Tarjan [7], over algorithms with inverse-Ackermann-type complexity by Chazelle [8] and Pettie [9], to an algorithm by Pettie [10] whose time complexity is provably optimal. Frequently, the most important ingredients were advances in data structures used to represent the graph.

In the upcoming sections, we will explore this colorful universe of MST algorithms. We will meet the canonical works of the classics, the clever ideas of their successors, various approaches to the problem including randomization and solving of important special cases. At several places, we will try to contribute our little stones to this mosaic.

When compared with the earlier surveys on the minimum spanning trees, most notably Graham and Hell [1] and Eisner [11], this work adds many of the recent advances, the dynamic algorithms and also the relationship with computational models. We tried to be self-contained and to include proofs of the main results, including the low-level details where they are important. This paper is based on a part of the author's Ph.D. Thesis [12].

Notation 1.1.3. We have tried to stick to the usual notation except where it was too inconvenient. Most symbols are defined at the place where they are used for the first time. A complete index of symbols with pointers to their definitions is then available in [Appendix](#). This appendix also describes the formalism of multigraphs and of Ackermann's function, both of which are not defined consistently in the common literature.

To avoid piling up too many symbols at places that speak about a single fixed graph, this graph is always called G , its set of vertices and edges are denoted by V and E respectively, and I also use n for the number of its vertices and m for the number of edges. At places where there could be a danger of confusion, more explicit notation is used instead.

1.2. Basic properties

In this section, we will examine the basic properties of spanning trees and prove several important theorems which will serve as a foundation for our MST algorithms. We will mostly follow the theory developed by Tarjan in [13].

For the whole section, we will fix a connected graph G with edge weights w and all other graphs will be spanning subgraphs of G . We will use the same notation for the subgraphs as for the corresponding sets of edges.

First of all, let us show that the weights on edges are not necessary for the definition of the MST. We can formulate an equivalent characterization using an ordering of edges instead.

Definition 1.2.1 (Heavy and Light Edges). Let T be a spanning tree. Then:

- For vertices x and y , let $T[x, y]$ denote the (unique) path in T joining x with y .
- For an edge $e = xy$ we will call $T[e] := T[x, y]$ the path covered by e and the edges of this path edges covered by e .

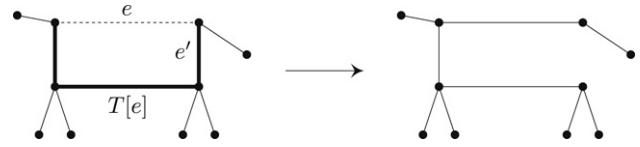


Fig. 1 – An edge exchange as in the proof of Lemma 1.2.3.

- An edge e is called *light with respect to T* (or just *T -light*) if it covers a heavier edge, i.e., if there is an edge $f \in T[e]$ such that $w(f) > w(e)$.
- An edge e is called *T -heavy* if it covers a lighter edge.

Remark 1.2.2. Edges of the tree T cover only themselves and thus they are neither heavy nor light. The same can happen if an edge outside T covers only edges of the same weight, but this will be rare because all edge weights will be usually distinct.

Lemma 1.2.3 (Light Edges). Let T be a spanning tree. If there exists a T -light edge, then T is not minimum.

Proof. If there is a T -light edge e , then there exists an edge $e' \in T[e]$ such that $w(e') > w(e)$. Now $T - e'$ (T with the edge e' removed) is a forest of two trees with endpoints of e located in different components, so adding e to this forest must restore connectivity and $T' := T - e' + e$ is another spanning tree with weight $w(T') = w(T) - w(e') + w(e) < w(T)$. Hence T could not have been minimum. \square

The converse of this lemma is also true and to prove it, we will once again use the technique of transforming trees by exchanges of edges. In the proof of the lemma, we have made use of the fact that whenever we exchange an edge e of a spanning tree for another edge f covered by e , the result is again a spanning tree (see Fig. 1). In fact, it is possible to transform any spanning tree to any other spanning tree by a sequence of exchanges.

Lemma 1.2.4 (Exchange Property for Trees). Let T and T' be spanning trees of a common graph. Then there exists a sequence of edge exchanges that transforms T to T' . More formally, there exists a sequence of spanning trees $T = T_0, T_1, \dots, T_k = T'$ such that $T_{i+1} = T_i - e_i + e'_i$ where $e_i \in T_i$ and $e'_i \in T'$.

Proof. By induction on $d(T, T') := |T \Delta T'|$. When $d(T, T') = 0$, both trees are identical and no exchanges are needed. Otherwise, the trees are different, but as they have the same number of edges, there must exist an edge $e' \in T' \setminus T$. The cycle $T[e'] + e'$ cannot be wholly contained in T' , so there also must exist an edge $e \in T[e'] \setminus T'$. Exchanging e for e' yields a spanning tree $T^* := T - e + e'$ such that $d(T^*, T') = d(T, T') - 2$. Now we can apply the induction hypothesis to T^* and T' to get the rest of the exchange sequence (see Fig. 2). \square

In some cases, a much stronger statement is true:

Lemma 1.2.5 (Monotone Exchanges). Let T be a spanning tree such that there are no T -light edges and T' be an arbitrary spanning tree. Then there exists a sequence of edge exchanges transforming T to T' such that the weight of the tree does not decrease in any step.

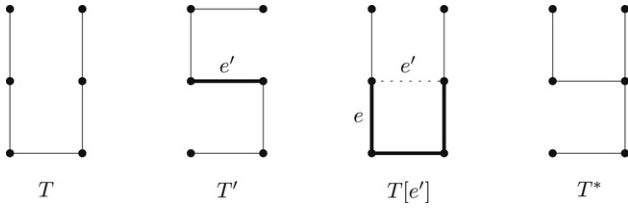


Fig. 2 – One step of the proof of Lemma 1.2.4.

Proof. We improve the argument from the previous proof, refining the induction step. When we exchange $e \in T$ for $e' \in T' \setminus T$ such that $e \in T[e']$, the weight never drops, since e' is not a T -light edge and therefore $w(e') \geq w(e)$, so $w(T^*) = w(T) - w(e) + w(e') \geq w(T)$.

To keep the induction going, we have to make sure that there are still no light edges with respect to T^* . In fact, it is enough to avoid such edges in $T' \setminus T^*$, since these are the only edges considered by the induction steps. To accomplish that, we replace the so far arbitrary choice of $e' \in T' \setminus T$ by picking the lightest such edge.

Let us consider an edge $f \in T' \setminus T^*$. We want to show that f is not T^* -light, i.e., that it is heavier than all edges on $T^*[f]$. The path $T^*[f]$ is either identical to the original path $T[f]$ (if $e \notin T[f]$) or to $T[f] \Delta C$, where C is the cycle $T[e'] + e'$. The former case is trivial, in the latter we have $w(f) \geq w(e')$ due to the choice of e' and all other edges on C are lighter than e' as e' was not T -light. \square

This lemma immediately implies that Lemma 1.2.3 works in both directions:

Theorem 1.2.6 (Minimality of Spanning Trees). A spanning tree T is minimum iff there is no T -light edge.

Proof. If T is minimum, then by Lemma 1.2.3 there are no T -light edges. Conversely, when T is a spanning tree without T -light edges and T_{\min} is an arbitrary minimum spanning tree, then according to the Monotone exchange lemma (Lemma 1.2.5) there exists a non-decreasing sequence of exchanges transforming T to T_{\min} , so $w(T) \leq w(T_{\min})$ and thus T is also minimum. \square

In general, a single graph can have many minimum spanning trees (for example a complete graph on n vertices with unit edge weights has n^{n-2} minimum spanning trees according to Cayley’s formula [14]). However, as the following theorem shows, this is possible only if the weight function is not injective.

Theorem 1.2.7 (Uniqueness of MST). If all edge weights are distinct, then the minimum spanning tree is unique.

Proof. Consider two minimum spanning trees T_1 and T_2 . According to the previous theorem, there are no light edges with respect to either of them, so by the Monotone exchange lemma (1.2.5) there exists a sequence of non-decreasing edge exchanges going from T_1 to T_2 . As all edge weights are all distinct, these edge exchanges must be in fact strictly increasing. On the other hand, we know that $w(T_1) = w(T_2)$, so the exchange sequence must be empty and indeed T_1 and T_2 must be identical. \square

Notation 1.2.8. When G is a graph with distinct edge weights, we will use $\text{mst}(G)$ to denote its unique minimum spanning tree.

Also the following trivial lemma will be often invaluable:

Lemma 1.2.9 (Edge Removal). Let G be a graph with distinct edge weights and $e \in G \setminus \text{mst}(G)$. Then $\text{mst}(G - e) = \text{mst}(G)$.

Proof. The tree $T = \text{mst}(G)$ is also a MST of $G - e$, because every T -light edge in $G - e$ is also T -light in G . Then we apply the uniqueness of the MST of $G - e$. \square

1.2.10. Comparison oracles

To simplify the description of MST algorithms, we will assume that the weights of all edges are distinct and that instead of numeric weights we are given a **comparison oracle**. The oracle is a function that answers questions of type “Is $w(e) < w(f)$?” in constant time. This will conveniently shield us from problems with representation of real numbers in algorithms and in the few cases where we need a more concrete input, we will explicitly state so.

In case the weights are not distinct, we can easily break ties by comparing some unique identifiers of edges. According to our characterization of minimum spanning trees, the unique MST of the new graph will still be a MST of the original graph. Sometimes, we could be interested in finding all solutions, but as this is an uncommon problem, we will postpone it until Section 5.5. For the time being, we will always assume distinct weights.

Observation 1.2.11. If all edge weights are distinct and T is an arbitrary spanning tree, then every edge of G is either T -heavy, or T -light, or contained in T .

1.2.12. Monotone isomorphism

Another useful consequence of the Minimality theorem is that whenever two graphs are isomorphic and the isomorphism preserves the relative order of weights, the isomorphism applies to their MSTs as well:

Definition 1.2.13. A *monotone isomorphism* between two weighted graphs $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$ is a bijection $\pi : V_1 \rightarrow V_2$ such that for each $u, v \in V_1 : uv \in E_1 \Leftrightarrow \pi(u)\pi(v) \in E_2$ and for each $e, f \in E_1 : w_1(e) < w_1(f) \Leftrightarrow w_2(\pi[e]) < w_2(\pi[f])$.

Lemma 1.2.14 (MST of Isomorphic Graphs). Let G_1 and G_2 be two weighted graphs with distinct edge weights and π a monotone isomorphism between them. Then $\text{mst}(G_2) = \pi[\text{mst}(G_1)]$.

Proof. The isomorphism π maps spanning trees to spanning trees bijectively and it preserves the relation of covering. Since it is monotone, it preserves the property of being a light edge (an edge $e \in E(G_1)$ is T -light \Leftrightarrow the edge $\pi[e] \in E(G_2)$ is $\pi[T]$ -light). Therefore by the Minimality theorem (1.2.6), T is the MST of G_1 if and only if $\pi[T]$ is the MST of G_2 . \square

1.3. The Red–Blue meta-algorithm

Most MST algorithms can be described as special cases of the following procedure (again following Tarjan [13]):

Algorithm 1.3.1 (*Red–Blue Meta-Algorithm*).

Input: A graph G with an edge comparison oracle (see 1.2.10)

1. At the beginning, all edges are colored black.
2. Apply rules as long as possible:
3. Either pick a cut C such that its lightest edge is not blue and color this edge blue, (*Blue rule*)
4. or pick a cycle C such that its heaviest edge is not red and color this edge red. (*Red rule*)

Output: Minimum spanning tree of G consisting of edges colored blue.

1.3.2

This procedure is not a proper algorithm, since it does not specify how to choose the rule to apply. We will however prove that no matter how the rules are applied, the procedure always stops and it gives the correct result. Also, it will turn out that each of the classical MST algorithms can be described as a specific way of choosing the rules in this procedure, which justifies the name meta-algorithm.

Notation 1.3.3. We will denote the unique minimum spanning tree of the input graph by T_{\min} . We intend to prove that this is also the output of the procedure.

1.3.4. Correctness

Let us prove that the meta-algorithm is correct. First we show that the edges colored blue in any step of the procedure always belong to T_{\min} and that the edges colored red are guaranteed to be outside T_{\min} . Then we demonstrate that the procedure always stops. Some parts of the proof will turn out to be useful in the upcoming sections, so we will state them in a slightly more general way.

Lemma 1.3.5 (*Blue Lemma, also known as the Cut Rule*). *The lightest edge of every cut is contained in the MST.*

Proof. By contradiction. Let e be the lightest edge of a cut C . If $e \notin T_{\min}$, then there must exist an edge $e' \in T_{\min}$ that is contained in C (take any pair of vertices separated by C : the path in T_{\min} joining these vertices must cross C at least once). Exchanging e for e' in T_{\min} yields an even lighter spanning tree since $w(e) < w(e')$ (see Fig. 3). \square

Lemma 1.3.6 (*Red Lemma, also known as the Cycle Rule*). *An edge e is not contained in the MST iff it is the heaviest on some cycle.*

Proof. The implication from the left to the right follows directly from the Minimality theorem: if $e \notin T_{\min}$, then e is T_{\min} -heavy and so it is the heaviest edge on the cycle $T_{\min}[e] + e$.

We will prove the other implication again by contradiction. Suppose that e is the heaviest edge of a cycle C and that $e \in T_{\min}$. Removing e causes T_{\min} to split into two components, let us call them T_x and T_y . Some vertices of C now lie in T_x , the others in T_y , so there must exist in edge $e' \neq e$ such that

Fig. 3 – Proof of the Blue (left) and Red (right) lemma.

Fig. 4 – Configurations in the proof of the Black lemma.

its endpoints lie in different components. Since $w(e') < w(e)$, exchanging e for e' yields a spanning tree lighter than T_{\min} (see Fig. 3). \square

Lemma 1.3.7 (*Black Lemma*). *As long as there exists a black edge, at least one rule can be applied.*

Proof. Assume that $e = xy$ is a black edge. Let us define M as the set of vertices reachable from x using only blue edges. If y lies in M , then e together with some blue path between x and y forms a cycle and e must be the heaviest edge on this cycle. This holds because all blue edges have been already proven to be in T_{\min} and there can be no T_{\min} -light edges. In this case, we can apply the Red rule.

On the other hand, if $y \notin M$, then the cut formed by all edges between M and $V \setminus M$ contains no blue edges, therefore we can use the Blue rule (see Fig. 4). \square

Notation 1.3.8. We will use $\delta(M)$ to denote the cut separating M from its complement. That is, $\delta(M) = \{uv \in E \mid u \in M, v \notin M\}$. We will also abbreviate $\delta(\{v\})$ as $\delta(v)$.

Theorem 1.3.9 (*Red–Blue Correctness*). *For any selection of rules, the Red–Blue procedure stops and the blue edges form the minimum spanning tree of the input graph.*

Proof. To prove that the procedure stops, let us notice that no edge is ever recolored, so we must run out of black edges after at most m steps. Recoloring to the same color is avoided by the conditions built in the rules, recoloring to a different color would mean that the edge would be both inside and outside T_{\min} due to our Red and Blue lemmata.

When no further rules can be applied, the Black lemma guarantees that all edges are colored, so by the Blue lemma all blue edges are in T_{\min} and by the Red lemma all other (red) edges are outside T_{\min} . Thus the blue edges are exactly T_{\min} . \square

