Electronic Colloquium on Computational Complexity, Report No. 339 (2026)

A Deterministic Linear Time Algorithm for Minimum Spanning Tree

Sanjeev Khanna* Aaron Putterman† Junkai Song‡

November 23, 2026

Abstract

We show that there is a deterministic linear time algorithm for the minimum spanning tree problem, thereby settling this problem, and providing the first improvement since the work of Chazelle (JACM 2000). Our work proceeds by showing that the deterministic comparison complexity of the minimum spanning tree problem is linear, whereby a known reduction of Pettie and Ramachandran (JACM 2002) immediately implies a linear time algorithm.

*Courant Institute, New York University, New York, NY, USA. Supported in part by NSF award CCF-2625203 and AFOSR award FA9550-25-1-0107. Email: sanjeev.khanna@nyu.edu.

†School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts, USA. Supported in part by a Jane Street Graduate Research Fellowship, the Simons Investigator Awards of Madhu Sudan and Salil Vadhan and AFOSR award FA9550-25-1-0112. Email: louiealbp@gmail.com.

‡Courant Institute, New York University, New York, NY, USA. Supported in part by NSF award CCF-2625203. Email: junkaisong@nyu.edu.

ISSN 1433-8092

Contents

1 Introduction **1**
- 1.1 A brief history of the MST problem . . . 1
- 1.2 Technical Overview . . . 5
- 1.3 Acknowledgments . . . 10
- 1.4 AI Acknowledgments . . . 10

2 Preliminaries **11**

3 The Certificate Procedure **12**
- 3.1 Overview and Formal Algorithm . . . 12
- 3.2 The Batch Processing . . . 14
- 3.3 Bounding Unsupported Edges . . . 16
- 3.4 Building the Schedule . . . 18
- 3.5 Estimating Preferred Actions . . . 22
- 3.6 Putting the Pieces Together . . . 22

4 The Final Algorithm **25**

1 Introduction

The *minimum spanning tree* (MST) problem is a cornerstone of theoretical computer science and graph algorithms. Given a connected undirected graph $G = (V, E)$ with n vertices, m edges, and a real-valued weight function $w\colon E \to \mathbb{R}$, the MST problem asks for a spanning tree of G with minimum total weight.[1]

The MST problem has a long and storied history, with the first work studying it often being attributed to Borůvka [Bor26a, Bor26b], who encountered the problem during an effort to design an efficient scheme for the electrification of Moravia.[2] This study led to the development of the first efficient algorithm for the MST problem [Bor26a, Bor26b], and to this day it remains one of the most extensively studied problems in the area of combinatorial algorithms. Several textbook algorithms based on natural greedy rules, including Borůvka's algorithm, admit near-linear $O(m \log n)$-time implementations. However, despite its rich history, the most basic question about the MST problem has remained unresolved:

Does the MST problem admit a deterministic algorithm with $O(m)$ worst-case running time?

Naturally, the above question has seen significant attention, and as a consequence, several weaker linear-time bounds were already known. Klein and Tarjan [KT94], and later Karger, Klein, and Tarjan [KKT95], gave a *randomized* algorithm that always returns an MST and runs in linear time both in expectation and with high probability. Fredman and Willard [FW94] gave a deterministic linear-time algorithm for MST with word-sized integer weights on a word RAM. Similarly, deterministic linear-time *verification* of a supplied spanning tree was known by the early 1990s [DRT92].

We study the most general version of the problem, where the algorithm may only compare edge weights in pairs, with each comparison taking constant time, and otherwise treats the weights as opaque keys (thus, the algorithm works for even the most unstructured weighted settings). In this setting, the best previously known explicit worst-case bound for deterministic comparison-based construction was $O(m\alpha(m, n))$, due to Chazelle [Cha00a]. In this context, α is a parameterization of the extremely slowly-growing *inverse-Ackermann* function. In a later work, Pettie and Ramachandran [PR02] established that optimal deterministic running time is governed by $\mathcal{T}^*(m, n)$, which we use to denote the worst-case deterministic *comparison complexity* of minimum spanning forests on simple graphs with m edges and n vertices (see Section 1.1). Consequently, a deterministic linear-time comparison-based MST algorithm exists if and only if $\mathcal{T}^*(m, n) = O(m)$. However, this work of [PR02] left open a tight bound on $\mathcal{T}^*(m, n)$.

In this paper, we settle the above question by proving the following theorem.

Theorem 1.1. *There is a uniform deterministic algorithm that, on every connected undirected multigraph with n vertices and m edges with real weights, finds an MST in $O(m)$ worst-case time, accessing edge weights only through pairwise comparisons.*

1.1 A brief history of the MST problem

We now give a brief overview of prior work on the MST problem, starting with the seminal work of [Bor26a, Bor26b]. Borůvka's 1926 algorithm works in phases: starting with a graph $G = (V, E)$,

[1] In general, for a disconnected graph, one asks for a *minimum spanning forest* (MSF), which simply contains an MST of each connected component.

[2] Presumably, the vertices were villages, and edge weights represented the cost of connecting the power grid between two towns.

each vertex selects its incident edge with the smallest weight. The algorithm then *commits* to keeping these edges in the MST, and effectively *contracts* on these committed edges, treating the created components as their own vertices. In this way, every phase at least halves the number of vertices, so after $O(\log n)$ phases, each using $O(m)$ comparisons, one obtains an MST. In 1930, Jarník [Jar30] described a tree-growing alternative. Indeed, this work proposed starting from a single vertex and repeatedly adding the cheapest edge leaving the current tree. This algorithm was later rediscovered by Prim [Pri57] and by Dijkstra [Dij59] and is now commonly called Prim's algorithm. Kruskal [Kru56] developed yet another alternative. In his work, he proposes scanning the edges in *increasing order of weight* and keeping each edge that closes no cycle. Naturally, Kruskal's algorithm relies on sorting the underlying edges (in addition to maintaining a connectivity data structure) and so also requires an $O(\log n)$ overhead. For a more complete discussion of these early algorithms and their rediscoveries, we encourage the reader to the work of Graham and Hell [GH85] trace these developments. Similarly, Nešetřil, Milková, and Nešetřilová [NMN01] give English translations of both of Borůvka's 1926 papers.

Year	**Algorithm**	**Running-time guarantee**
1926	Borůvka [Bor26a, Bor26b]	$O(m \log n)$
1930 / 1957	Jarník [Jar30] / Prim [Pri57]	$O(m + n \log n)$ with Fibonacci heaps
1956	Kruskal [Kru56]	$O(m \log n)$
1975	Yao [Yao75]	$O(m \log \log n)$
1975	Johnson [Joh75]	$O(m \log_d n)$
1976	Cheriton and Tarjan [CT76]	$O(m \log(\log_d n))$
1984 / 1987	Fredman and Tarjan [FT87]	$O(m\beta(m, n))$
1984 / 1986	Gabow, Galil, and Spencer [GGS84], later with Tarjan [GGST86]	$O(m \log \beta(m, n))$
1997	Chazelle [Cha97]	$O(m\alpha(m, n) \log \alpha(m, n))$
2000	Chazelle [Cha00a]	$O(m\alpha(m, n))$
2000 / 2002	Pettie and Ramachandran [PR02]	$O(T^*(m, n))$, optimal
Other settings		
1990 / 1994	Fredman and Willard [FW94]	$O(m)$ for word-sized integer weights on a word RAM
1994 / 1995	Klein and Tarjan [KT94], then Karger, Klein, and Tarjan [KKT95]	$O(m)$ in expectation and with high probability

Table 1: Selected MST algorithms for connected simple graphs where the first block gives deterministic comparison-based worst-case bounds. In the above, we use $d := 2 + m/n$, and $\beta(m, n) := \min\{\, i \geq 1 : \log^{(i)} n \leq m/n \,\}$.

All of the above-mentioned algorithms rest on two dual principles. Assume, as we may by breaking ties consistently, that the edge weights are *distinct*, so that the MST is *unique*. The *cut property* states that the lightest edge crossing any cut belongs to the MST; the *cycle property* states that the heaviest edge on any cycle does not.

Tarjan [Tar83] cast these two conditions above as coloring rules. On a cut crossed by no blue edge, color the lightest uncolored crossing edge blue. On a cycle containing no red edge, color the heaviest uncolored edge red. Performing a any sequence of moves in keeping with the above conditions, once every edge is colored, the blue edges necessarily form the MST.

Beating sorting. Since the MST depends only on the *relative order* of the weights, the natural cost measure of finding the MST is the number of comparisons (together with the amount of work needed to decide which comparisons to make). On simple graphs, each of the aforementioned classical algorithms runs in $O(m \log n)$ time. Kruskal's algorithm sorts the edges in $O(m \log m) = O(m \log n)$ time, Prim's algorithm performs $O(m)$ operations on a binary heap of at most n vertices, and Borůvka's algorithm may need $\Theta(\log n)$ linear-time phases.

Progress towards linear time MST algorithms has largely happened by first focusing on *dense* instances. Indeed, on dense simple graphs with $m = \Theta(n^2)$, an $O(n^2)$ array implementation of Prim's algorithm already runs in linear time. This was further improved in the mid-1970s, which brought about algorithms that improved on the $O(m \log n)$ comparison bound even for *sparse graphs*. For instance, Yao [Yao75] combined Borůvka's phases with the then-new linear-time selection algorithm [BFP+72], partially sorting the adjacency lists to obtain $O(m \log \log n)$ time. Later work by Cheriton and Tarjan [CT76] obtained the sharper bound $O(m \log(\log_d n))$ (where $d = 2 + m/n$), which is even *linear* when $m = \Omega(n^{1+\varepsilon})$ for any constant $\varepsilon > 0$, together with a linear-time algorithm for planar graphs.

In the 1980's, the seminal work of Fredman and Tarjan [FT87] introduced *Fibonacci heaps*. Using these, Prim's algorithm can be implemented to run in $O(m + n \log n)$ time, which is linear for any $m = \Omega(n \log n)$. In fact, using these heaps, Fredman and Tarjan were able to produce an even faster *phased* algorithm. In each phase of this new algorithm, trees are grown one at a time in a manner similar to Prim's algorithm. Each tree starts from a vertex of the current contracted graph that is not yet in any tree and storing its neighboring vertices in a heap keyed by their cheapest connecting edges. A tree stops growing when its heap exceeds a threshold, when it reaches a tree grown earlier in the phase, or when its heap becomes empty. The algorithm then contracts the resulting trees and repeats this procedure with a larger threshold. By carefully optimizing these thresholds, this yields a deterministic $O(m\beta(m, n))$ time algorithm, where

$$\beta(m, n) = \min\{\, i \geq 1 : \log^{(i)} n \leq m/n \,\},$$

(note that $\log^{(i)}$ denotes i iterations of the base-two logarithm). In particular, this means that $\beta(m, n) \leq \log^* n$, and one can even show that the algorithm runs in *linear time* whenever $m \geq n \log^{(k)} n$ for a fixed positive integer k. Gabow, Galil, and Spencer [GGS84] later refined this approach in 1984 to obtain a deterministic $O(m \log \beta(m, n))$ time algorithm.

Non-greedy algorithms and the $O(m\alpha(m, n))$ bound. Progress beyond this $O(m \log \beta(m, n))$ running time required a fundamentally new idea. This was ultimately supplied in 1997, when Chazelle [Cha97] gave a breakthrough deterministic algorithm running in $O(m\alpha(m, n) \log \alpha(m, n))$ time. Here α is a two-parameter inverse-Ackermann function as in Tarjan's analysis of union–find [Tar75], and is in general, an extremely slowly growing function. Chazelle subsequently improved the bound to $O(m\alpha(m, n))$ in [Cha00a], along with an independent, similar bound developed by Pettie [Pet99] (still building on [Cha97]).

Chazelle's algorithm crucially *departs* from the greedy paradigm. Its main tool is the so-called *soft heap* [Cha00b], a meldable priority queue that can *corrupt* a key by increasing its value. For an error parameter $0 < \varepsilon \leq 1/2$, after N insertions, the guarantee is simply that *at most* εN items currently in the heap have *corrupted* keys. Every operation takes constant amortized time except insertion, which takes $O(\log(1/\varepsilon))$ amortized time. With these soft heaps in hand, the algorithm first computes a spanning tree and then refines it until it becomes an MST. The ultimate algorithm is comparison-based, runs on a pointer machine, and makes no numerical assumptions about the weights, and, until now, remained the best explicit bound on the deterministic complexity of finding an MST.

Verification algorithms. In parallel to these works designing better MST algorithms, another line of work has concerned *verification*: i.e., the task of deciding whether a given spanning tree T is minimum. By the aforementioned cycle property, this effectively amounts to checking that every non-tree edge is heavier than every edge on the T-path between its endpoints. In his work [Tar79a], Tarjan showed how to solve this path-maximum problem in $O(m\alpha(m,n))$ time using path compression on balanced trees. Komlós [Kom85] then refined this, and showed that a *linear* number of comparisons suffices for verification. However, his method did not give a *linear-time* way to decide which comparisons to make. This was then first achieved by Dixon, Rauch, and Tarjan [DRT92], who obtained a linear-time implementation on a unit-cost RAM by combining Komlós's algorithm with table lookup for small subproblems and with Tarjan's earlier method. King [Kin97] later gave a simpler implementation of this algorithm, and ultimately, Buchsbaum et al. [BKRW98, BGK+08] even achieved linear time on a pointer machine.

Verification also turned out to be useful for *constructing* MSTs. For any forest $F \subseteq G$, let us call an edge *F-heavy* if it is heavier than every edge on the F-path between its endpoints, and *F-light* otherwise. By the cycle property, F-heavy edges can be discarded, and an adaptation of the verification algorithm identifies all of them in linear time [DRT92, KKT95]. This raised the hope that this *filtering* could be a stepping stone for faster deterministic algorithm. Pettie [Pet06] ruled out one natural approach by proving that *online* verification, in which path-maximum queries arrive one at a time, indeed requires $\Omega(m\alpha(m,n)+n)$ comparisons.

Randomized algorithms. Despite this barrier proved by [Pet06], it is nevertheless the case that combining this *filtering* step with random sampling is the key idea needed to get an $O(m)$ time *randomized* MST algorithm. Karger [Kar93] initiated this approach, whereby he combined random sampling with verification to obtain an MST algorithm running in $O(m + n\log n)$ time with high probability. Klein and Tarjan [KT94] refined this, obtaining an expected linear time algorithm in 1993. A subsequent work of Karger, Klein, and Tarjan [KKT95] provides the same result but with an *even stronger* probability bound. Their algorithm alternates between so-called "Borůvka phases", which shrink the number of vertices, with the new idea of sampling and filtering, which shrink the number of edges. The crucial lemma which underlies their result shows that if each edge is sampled independently with probability $0 < p \leq 1$ and F is the sample's minimum spanning forest, then the expected number of F-light edges of G is at most n/p. This idea of using Borůvka phases to reduce vertices, and "filtering" to reduce edges, will also re-appear in the algorithm we present below (though the filtering will naturally require an entirely different approach to achieve a deterministic guarantee).

Stronger models and structured inputs. Earlier deterministic algorithms also achieved linear time by exploiting the representation of edge weights or the structure of the input graph. Fredman and Willard's algorithm [FW94] runs in linear time on a unit-cost word RAM when each integer weight fits in a machine word. Its atomic heaps exploit word-level operations on the keys. Within the comparison model, deterministic linear-time algorithms were known for planar graphs [CT76], for each fixed proper minor-closed class of simple graphs [Mar04], and for sufficiently dense graphs [FT87], as we discussed above.

An optimal algorithm and its comparison complexity. Pettie and Ramachandran [PR02] made the first explicit link between the number of comparisons required to find an MST and running time. To make this formal, let us fix a simple undirected graph H. A *comparison tree* for H is a decision tree where each internal node compares two edge weights and whose leaves are labeled

with spanning forests of H. We say that it is correct if, for every possible assignment of distinct weights to H, the leaf reached is labeled with the resulting MSF. The depth of the comparison tree is then exactly the worst-case number of comparisons required to find an MSF of H. With this, we can define

$$\mathcal{T}^*(m,n) \;=\; \max_{H} \min_{T} \operatorname{depth}(T),$$

where H ranges over all simple undirected graphs with m edges and n vertices, including disconnected ones, and T ranges over all correct comparison trees for H.

Every deterministic comparison-based MSF algorithm induces a correct comparison tree for each fixed topology H, so on graphs with m edges and n vertices it makes at least $\mathcal{T}^*(m,n)$ comparisons in the worst case. Pettie and Ramachandran gave a uniform pointer-machine algorithm that matches this bound. Indeed, on connected inputs, their algorithm runs in $O(\mathcal{T}^*(m,n))$ time[3] and it constructs the *optimal* comparison trees that it uses, relying on a careful use of soft heaps, Borůvka steps, and brute force computation of optimal comparison trees on small sub-instances.

For $m, n > 2$, the best previously known bounds were

$$\mathcal{T}^*(m,n) = \Omega(m) \qquad \text{and} \qquad \mathcal{T}^*(m,n) = O(m\alpha(m,n)),$$

leaving open whether a linear number of comparisons always suffices.

In this work, we show that it does: $\mathcal{T}^*(m,n) = O(m)$. Together with the algorithm of Pettie and Ramachandran [PR02], this yields Theorem 1.1.

1.2 Technical Overview

Why Edge Filtering is Key As alluded to above, the key new technical ingredient that we introduce is a mechanism for *efficiently finding* many "redundant edges" (i.e., edges which are the heaviest edges in a cycle in the graph). To properly describe how the algorithm works, we first introduce the notion of being *supported*. Indeed, for a graph $G = (V, E)$ and a forest $F \subseteq E$, we say that an edge $e \in E$ is *supported* by F, if the endpoints of e are connected by $F_{\leq w(e)}$. In words, this is simply asking that if one adds the edge e to the forest F, then the resulting cycle uses edges of weight only $\leq w(e)$. With this, we can present the formal statement of our key subroutine, namely, the certificate procedure:

Lemma 1.2 (Certificate Procedure)**.** *Let G be an undirected multigraph with n vertices, m edges, and no isolated vertices. For every fixed constant $\varepsilon > 0$, there exists a deterministic certificate procedure that outputs a forest $F \subseteq E(G)$. The procedure may invoke exact MSF calls on edge sets $E_1, \ldots, E_\ell$, each defined on a subset of $V(G)$. Apart from these calls, the procedure performs $O_\varepsilon(m)$ weight comparisons. Let $U(F) \subseteq E(G)$ denote the set of edges unsupported by F. The procedure guarantees that*

$$|F| + |U(F)| + \sum_{i=1}^{\ell} |E_i| \leq \varepsilon m + C_\varepsilon \sqrt{nm},$$

where C_ε is a constant depending only on ε.

Let us now unpack this lemma: we fix ε to be a small constant (in the final proof, we may even choose it to be $1/8$). The lemma then guarantees that there is a procedure which produces a forest F which in fact supports *many* of the original edges in the graph. Because these original edges

[3]This is why the definition of $\mathcal{T}^*$ allows disconnected graphs. A connected graph with $m = n - 1$ is its own MST, so over connected graphs alone we would have $\mathcal{T}^*(n-1, n) = 0$, whereas reading such an input already takes $\Omega(n)$ time.

are supported, this then provides a proof that (unless they themselves are in the forest), they are *not* present in the MST. Viewing this as a filtering procedure, this means that we can reduce the number of edges in the graph from $|E(G)|$ all the way down to $|F| + |U(F)|$.

This now explains why the left side of the expression in the above lemma is meaningful. Ultimately, we will recurse on a graph with $|F| + |U(F)|$ edges, which is exactly the number of edges left over after filtering. The second piece of the expression on the left is the actual *MST recursive calls* that are used in the computation of the certificate. Thus, the combined size of all recursive calls that are performed is bounded by $|F| + |U(F)| + \sum_{i=1}^{\ell} |E_i| \leq \varepsilon m + C_\varepsilon \sqrt{nm}$.

This expression itself is still not perfect however. On the right hand side, the $\varepsilon \cdot m$ term is quite small, but $C_\varepsilon \sqrt{nm}$ term can be quite large, especially when n approaches m. This is the key place where Borůvka phases come into play. A single Borůvka phase uses $O(m)$ comparisons and *halves* the number of vertices in the graph. Because C_ε is just a constant, we can then apply a *large enough constant* number of Borůvka phases to reduce n, and thereby ensure that $C_\varepsilon \sqrt{nm} \leq m/4$. All together, this will then guarantee that the combined size of all recursive calls satisfies $|F| + |U(F)| + \sum_{i=1}^{\ell} |E_i| \leq m/2$.

In order to finish off the argument, we still need to bound the total number of comparisons that are made. This just requires summing up the number of comparisons across all pieces of the algorithm: (1) the certificate procedure itself makes at most $O_\varepsilon(m)$ weight comparisons, (2) performing a constant number of Borůvka phases requires only $O(m)$ weight comparisons, and finally, (3) by using the work of [Kom85], it turns out that *identifying* the set of unsupported edges (given a forest F) can be done in only $O(m)$ comparisons.

All together, this leads to a recursive relationship of the form $T(m) = T(m/2) + O(m)$, which gives a bound of the form $T(m) = O(m)$, as we desire.

Intuition for the Certificate Procedure With the above in place, the remaining ingredient (and indeed the key roadblock) is to understand *how* we can actually build (and prove the correctness of) the certificate procedure. At a very high level, the procedure operates by repeatedly *projecting* vertices onto smaller and smaller vertex sets. In each such projection, the goal is for a vertex v to be assigned to a vertex $\pi(v)$, such that the weight of the edge connecting v and $\pi(v)$ is small. In this way, for an edge (u, v), if u and v are repeatedly projected, eventually reaching a small enough vertex set where one can find a low-weight path between their respective projections, then this should in fact produce a *cycle* involving (u, v), where (u, v) is the highest-weight edge.

In a bit more detail, this is achieved by maintaining a subset P of the vertices, called *portals*. For each vertex v, the goal is to maintain an incident edge $\sigma(v)$ connecting v to a portal, called the *spoke* of v. Notationally, we let $\pi(v)$ denote the portal at the other endpoint of the spoke. It is also possible that v may not have a spoke, in this case we write $\sigma(v) = \pi(v) = \perp$. Now, for an edge $e = (u, v)$ whose endpoints both have spokes, the algorithm may *project* e to its portals $\pi(u)$ and $\pi(v)$. In the process, the algorithm represents the edge (u, v) with an edge of the same weight between $\pi(u)$ and $\pi(v)$.

Crucially, ensuring that (u, v) is the largest weight edge in the cycle *requires* that (u, v) is heavier than both of its spokes. For this reason, we let $p(e) = \max\{w(\sigma(u)), w(\sigma(v))\}$ be the larger of the two spoke weights, and we say the projection is *valid* if $p(e) \leq w(e)$. Because the algorithm repeatedly projects edges, the certificate procedure itself will be a *recursive* algorithm. At each level of the recursion, the input to the certificate procedure is an undirected multigraph G. Each edge $e \in E(G)$ has a weight $w(e)$ and a *projection record*, which stores, for each preceding recursion level, the pair of spokes used to project e at that level. We say that e is *admissible* if every spoke in its projection record has weight at most $w(e)$, or equivalently, if all projections of e at preceding

recursion levels are valid.

With this intuition established, there are still several questions we must answer like (1) how do we decide where to project vertices? (2) how do we preserve bounds on the number of inadmissible edges? (3) how do we ensure that we do not use too many comparisons?

The Certification Algorithm Unfortunately, there is no single, modular description where each of the above questions is answered separately. Instead, we must introduce the entire certification algorithm, and then separately reason about its (frequently interfering) properties. We now describe this algorithm, and encourage the reader to separately view the formal pseudocode of the algorithm in Algorithm 1 after understanding the essence of its behavior.

The algorithm takes as input a graph G with n vertices and m edges, and initializes its forest set $F = \varnothing$, its portal set $P = \varnothing$, and its set $R = \varnothing$ of rejected (inadmissible edges). Similarly, every vertex is given a starting empty spoke $\sigma(v) = \perp$. The algorithm is then broken into t^2 *disjoint rounds*, which are then organized into t blocks of t rounds (t itself is a parameter we will choose later, which relates to the recursion depth). The intuition is that the edges $E(G)$ will now be partitioned among the t^2 rounds, with each edge being assigned to a single round (we call this the *schedule* of the algorithm). This schedule will designate *when the edge* e is permitted to act – a priori it may seem excessive to keep track of the algorithm on such a fine-grained level, but designing a linear comparison algorithm unfortunately requires extensive care of exactly when and where edges are permitted to have their weights queried. Importantly, we cannot afford to have many edges re-appearing in different recursive calls, thereby increasing the comparison complexity.

To this end, we now further restrict each edge. In the round where it is selected to act, it has only two possible actions that it can take:

1. The find action searches for a valid projection.
2. The project action projects e to $(\pi(u), \pi(v))$ using the current spokes, *regardless* of whether the projection is valid. Crucially however, those edges taking the project action will enter a child recursive instance (unlike the edges taking the find action).

It is worth noting that the choice between doing a find action vs. a project action is *completely deterministic*. Whenever an edge selects the find action, it is added to the current *batch* B of edges. Otherwise, if an edge selects the project action, we project the edge along its spokes, and *recursively* invoke the certificate algorithm.

Finally, at the end of every block of t rounds, the cumulative *batch* of edges B that has been collected so far is processed. This produces a set P' of new portals which is combined with P, a set D of so-called "declared" edges which are used to update the forest F, along with some potentially rejected edges R', which are added to the rejected set.

With this context established, we can now more concretely ask the above questions, which we present below. The remainder of the technical overview then describes the intuition for how each of these questions is answered. Note that several other details (and natural questions) arise in the design of this algorithm, but for the sake of brevity and clarity in this overview, we postpone their discussion to the actual algorithm analysis (see Section 3):

1. How do we decide which action an edge should take?
2. Why does the final forest produced by this procedure not have too many unsupported edges?
3. How do we select the schedule for the algorithm? Recall that we are not permitted to use randomness here!

Deciding Edge Actions We now must first discuss *which of the two actions* an edge should take when it is scheduled to act in a given round. This is captured by our "preferred action" framework, which is essentially just a deterministic set of rules which governs what action is preferred by an edge in a given round.

Recall that at the start of each round, every vertex is already assigned a spoke (or potentially $\perp$), and additionally, that there is a candidate forest F that the algorithm has computed so far. We then automatically say that the *preferred action* of an edge $e = (u, v)$ is find if either endpoint has no spoke or if both spokes lead to the same portal (if they are the same portal, then this is intuitively already making a cycle). Otherwise, when the spokes lead to distinct portals, we have a more careful tradeoff. We designate the preferred action to be find if $F_{\leq p(e)}$ connects u and v, and project otherwise (recall that here $p(e)$ is the weight of the heavier spoke). This already determines the rule for how an edge should ideally act.

We note that there is already one important subtlety which arises: just because it is *theoretically possible* to determine an edge's preferred action, this does not imply that there is an efficient comparison algorithm for computing it! This ends up becoming a material challenge in the design of the algorithm, and thus we must ultimately resort to *estimating* the preferred action of each edge. The estimate will end up being always correct when either endpoint has no spoke or if both spokes lead to the same portal (as this does not require comparisons), and otherwise is *mostly correct* at deciding between find and project. We omit an extensive discussion of this estimation procedure here, but it relies on a careful bound on the *number of possible preference vectors* (i.e., a complete enumeration of all possible joint preferences that can appear in a given round), along with a choice of a universal set of edges which can be used to distinguish two preference vectors that are far apart. Roughly speaking, this reduces the task of figuring out the preferences of *all edges* to figuring out the preferred actions for a *smaller set* of edges, while still mostly retaining correctness on the other edges.

Bounding Unsupported Edges For now, let us ignore this preceding detail about estimating the true preferred action of each edge in each round, and instead assume we have access to them. In this ideal situation, we must still show that *choosing preferred actions* actually leads to a good outcome in the Certificate algorithm.

Towards establishing this, we must first introduce the notion of good and bad rounds:

Definition 1.3. A round is *bad* for e if either of the following holds:

- It is the last round in its block with a project preference.
- The preferred action is find, and this block is the *last block* containing any find preference for this edge.

In more detail, within a single block, the preferences of an edge always look like PPPPFFF where P corresponds to project and F to find (in the sense that they are monotonically changing from P's to F's). The final P in any such block is said to be bad. Between blocks, preferences can transition back to Ps – i.e., something like

..... PPPPFFF PPFFFFF PPPPPPP

is possible. In this example, all the F's from the second to last block are bad. The crucial fact we end up showing is the following:

Lemma 1.4. *Consider an instance* Certificate(G) *and an edge* $e \in E(G)$*. Suppose that, for an edge* e*, the edge* e *and all of its projections throughout the recursion are* never scheduled to act in a bad round*. Then,* e *is supported by the final forest* F *returned by the algorithm.*

In other words, whenever an edge is rejected or otherwise not supported by the returned forest F from a call to Certificate(G), the only way that this is possible is that the edge or one of its projections was scheduled to act in a round where its preferred action was bad. Fortunately for us, we also now have a concrete understanding of how many bad rounds there are for a single edge. Out of the t^2 total rounds, there are only t bad choices of a bad round for preferred action P (one in each block) and only t bad choices for preferred action F (all rounds in the final block), leading to a total of $2t$ bad rounds. This only corresponds to a $2/t$ fraction of rounds being bad, which is sufficient for our final bound that there are not too many unsupported edges.

Finding a Good Schedule Nevertheless, the preceding discussion raises a separate concern. Even though in *expectation*, the number of bad rounds for each edge is small, how do we find a deterministic choice of schedule which *realizes* this behavior? This question is really at the heart of designing our linear time MST algorithm. The main difficulty is a subtle circular dependence: the preferred actions of each edge *depends* on the schedule, which in turn determines which rounds are good or bad. To reason about this dependence, we introduce an object called the *history*.

Definition 1.5 (History)**.** Consider a call Certificate(G) on a graph G with n vertices and m edges. We index the t^2 rounds by $i \in [t^2]$. We define the history to be

$$H = (\sigma_i, F_i, a_i)_{i \in [t^2]},$$

where:

- σ_i is a vector of length n, where $\sigma_i(v) \in E(G) \cup \perp$ specifies the spoke of vertex v at the start of round i.
- $F_i \subseteq E(G)$ is the forest F maintained by the algorithm at the start of round i.
- a_i is a vector of length m, where $a_i(e) \in \{\mathsf{find}, \mathsf{project}\}$ specifies the *true preferred action* of edge e at the start of round i.

Note that the history completely determines which rounds are good or bad. Our analysis will begin by showing that the number of histories is *not too large*. This then enables us to show that there exists a *single deterministic schedule* that ensures not too many edges are assigned to bad rounds.

The bound on the number of histories, though not complicated, is subtle. The number of choices for σ_i and F_i is easy to bound. Each vertex in σ_i has $\deg(v) + 1$ choices for its spoke, and each forest F_i has $\leq \binom{m+n-1}{n-1}$ choices. Taking the product over all choices of $i \in [t^2]$ and all vertices, this gives a simple bound of $e^{O(t^2\sqrt{mn})}$ possibilities for σ_i, F_i. It remains still to bound the number of possible *preference vectors* a_i.

For this bound, let us fix the spokes σ_i and the forest F_i at the start of round i. For each edge $e = (u, v) \in E(G)$, recall that its preferred action is find if (1) either endpoint has no spoke, (2) $\pi(u) = \pi(v)$, or (3) $F_{\leq p(e)}$ connects u, v, otherwise, it is project. The first two cases are completely determined by the spokes (and so do not contribute to a larger number of histories), so it remains to analyze case (3).

Consider a ground set T of size at most $2n$ consisting of all the spokes and edges of F_i. The total order is completely determined by edge weights. For each edge $e = (u, v) \in E(G)$, let $S_e \subseteq T$

consist of the spokes $\sigma_i(u), \sigma_i(v)$ and all edges on the u to v path in F_i, if such a path exists. The preferred action of e is find precisely when the maximum element of the set S_e is a spoke. Indeed, this occurs precisely when the heaviest spoke is heavier than every edge on the u to v path in F_i, which is equivalent to u, v being connected in $F_{\leq p(e)}$. Conversely, if the maximum weight edge in S_e is a path edge, then u and v are not connected in $F_{\leq p(e)}$. Note that here we are use the fact that there is a *unique* path between u and v in F (if one exists).

The takeaway from the above is that the preferred actions are entirely determined by the *maxima* of the sets S_e. A known result (first appearing due to Fredman in [GYY80]) then states that in a universe of size N, and M subsets of $[N]$, the number of different "maximizing patterns" (and thus, the number of different preference vectors) is at most $e^{O(\sqrt{NM})}$. Taking the bound across all t^2 rounds then bounds the number of histories by $e^{O(t^2\sqrt{mn})}$.

This is the first place where the parameter guarantees from the certificate procedure come into focus, with this $\sqrt{mn}$ being the same factor that appears. Indeed, with this bound on the number of histories, a careful probabilistic argument shows that we can *deterministically* (without any comparison queries) decide on a schedule which will not schedule more than $O(m/t + t^2\sqrt{mn})$ many edges in bad rounds.

This marks one of the crucial insights in the paper, and indeed the main bound which allows us to reach linear time. We do remark that there are several hurdles which we have overlooked (for instance, bounding the recursive complexity of calling the Certificate procedure on portal graphs, or bounding the number of queries required to determine which edges are admissible and not, and careful accounting of where rejected edges are sent). We encourage readers to visit Section 3 to see the complete algorithm pieced together.

1.3 Acknowledgments

We thank Vijaya Ramachandran for drawing our attention to her joint work with Seth Pettie [PR02] during the first author's visit to the University of Texas at Austin in spring 2026. It was then that we began working on the problem of designing a deterministic linear-time MST algorithm.

1.4 AI Acknowledgments

A. P. would like to thank OpenAI for providing the credits to Astra 6.2 which were used in generating this result.

Indeed, the main ideas in this paper were entirely generated by AI – specifically 6.2 Astra, on a mixture of Ultra and Max effort – with minimal human intellectual intervention (see below). The authors then worked to first carefully verify, and then to simplify and clarify the treatment, resulting in the present manuscript.

The authors had originally tried to query Astra 6 and 6.2 with a direct prompt of achieving deterministic linear comparison complexity for this problem, with no success. The authors then retried with the simpler task of proving any bound of the form $o(m\alpha)$, which would be an improvement over the work of Chazelle [Cha00a]. A team of 15 agents (managed across several Ultra and Max effort settings) succeeded in proving such an improvement after 15 hours, providing a bound of the form $O\left(m \cdot \frac{\alpha}{\log(\alpha)^{1/3}}\right)$. This gave the authors confidence that $O(m)$ is indeed the right answer. Before that, there was some doubt that linear time was the correct answer, largely due to $\Omega(m\alpha)$ lower bounds for related union-find data structures (see [Tar79b]), and variants of the MST problem [Pet06]. After this, the authors continuously prompted the model to iteratively improve the comparison complexity, resulting in a sequence of small improvements. This took several forms. The first improvement was a bound of the form $O(m \log(\alpha))$, which was subsequently followed by

$O(m \log^*(\alpha))$, then $O(m\alpha(\alpha))$, and then $O(m\Gamma(\alpha))$. Here, Γ can be seen as the "inverse of a hyper-Ackermannian function". See [Sch16], where the function Γ is essentially the inverse of F_{ω^ω}. In particular, Γ is smaller than *any constant number* of iterated inverse-Ackermann functions. This was then followed by bounds of the form $O(m \log(\Gamma(\alpha)))$, then $O(m \log^*(\Gamma(\alpha)))$, and even the absurdly small $O(m \cdot \Gamma(\Gamma(\alpha)))$. The authors then sent off one final batch of runs which used the entire history of developed algorithms until this point, and asked for a single linear time algorithm, which was finally generated. A collection of these intermediate papers are available at the following URL: https://github.com/Louiealbp/DLMST/tree/main/IntermediateMSTs.[4]

Throughout the entire process leading up to the discovery of our main result, human intellectual intervention was limited. The largest such intervention was the advice that the authors did not believe it to be a fruitful direction to pursue "structural graph theory" approaches to the problem (whereby the model would study efficient algorithms on graphs which have special structure like low-density, no short cycles, etc.), and repeatedly redirected attempts that got stuck in such pursuits. This guidance was largely due to the fact that the authors themselves had extensively studied structural approaches (in addition to other ideas) for linear time MST algorithms in the spring and summer of this year.

2 Preliminaries

Throughout the paper, log denotes the natural logarithm. We will use the following standard bound.

Fact 2.1. For every pair of positive integers n, m,

$$\log \binom{m+n}{n} \leq 2\sqrt{mn}.$$

Graphs. Given a graph $G = (V, E)$ and an edge subset $S \subseteq E$, we denote by $G[S] = (V, S)$ the edge-induced subgraph of G with edge set S. For each vertex $v \in V$, let $\deg_S(v)$ denote its degree in $G[S]$, and write $\deg_G(v)$ for its degree in G.

Minimum spanning forest (MSF). For a weighted graph G, let $\mathsf{MSF}(G)$ denote its minimum spanning forest. For simplicity, we assume that the graph contains no isolated vertices and that edge weights are distinct with ties broken by edge indices. Under this convention, the MSF is unique.

We recall three standard properties of minimum spanning forests.

Fact 2.2 (Cut Property)**.** Let $G = (V, E)$ be a weighted graph, and let $S \subseteq V$. If an edge e is the minimum weight edge crossing the cut $(S, V \setminus S)$, then $e \in \mathsf{MSF}(G)$.

Fact 2.3 (Cycle Property)**.** Let G be a weighted graph, and let C be a cycle in G. Then the maximum weight edge of C does not belong to $\mathsf{MSF}(G)$.

Fact 2.4 (Contraction property)**.** Let G be a weighted graph, and let $S \subseteq \mathsf{MSF}(G)$. Denote by G/S the graph obtained by contracting all edges in S. Then,

$$\mathsf{MSF}(G) = S \cup \mathsf{MSF}(G/S).$$

[4] There is also a collection of the various linear time algorithms that were ultimately developed at https://github.com/Louiealbp/DLMST/tree/main/OptimalMSTs.

For a forest $F = \mathsf{MSF}(G)$ and a weight threshold k, let $F_{\leq k} = \{e \in F : w(e) \leq k\}$. We will also use the following property of MSFs: they preserve connectivity at every weight threshold.

Fact 2.5. Let G be a graph, $S \subseteq E(G)$ and $F = \mathsf{MSF}(G[S])$. For any vertices u, v and threshold k, if $G[S]$ contains a path from u to v using only edges of weight at most k, then $F_{\leq k}$ connects u and v.

Comparison complexity. Our analysis focuses on the number of edge weight comparisons rather than the running time. By the work of Pettie and Ramachandran [PR02], a deterministic comparison bound for MSF can be translated into a corresponding deterministic running time bound.

Theorem 2.6 (Theorem 5.8 of [PR02])**.** *Let $\mathcal{T}^*(m, n)$ be the deterministic decision-tree complexity of the* MSF *problem on graphs with m edges and n vertices. There is a deterministic algorithm that computes the MSF of any graph with m edges and n vertices in $O(\mathcal{T}^*(m, n))$ time.*

Remark 2.7. As noted in [PR02], their algorithm does not require the full power of the RAM model and can be implemented in the pointer machine model of [Tar79b].

Borůvka steps. We recall the classical Borůvka's algorithm [Bor26a].

Definition 2.8 (Borůvka steps)**.** Given a weighted graph $G = (V, E)$ with m edges, a *Borůvka phase* selects, for each vertex $v \in V$, its minimum weight incident edge e_v. Let $S = \{e_v : v \in V\}$ be the set of selected edges. The algorithm adds S to the minimum spanning forest and replaces G by the contracted graph G/S.

By Fact 2.2 and Fact 2.4, we know that $\mathsf{MSF}(G) = S \cup \mathsf{MSF}(G/S)$. Since each minimum weight incident edge e_v can be found using $O(\deg_G(v))$ comparisons, a Borůvka step requires $O(m)$ comparisons. Moreover, as every non-isolated vertex is incident to a selected edge, a Borůvka step reduces the number of non-isolated vertices by at least a factor of two.

Path maximum queries. Finally, we will use the following result of Komlós [Kom85] (see also [DRT92, Kin97]) for answering path maximum queries in a forest.

Lemma 2.9 ([Kom85])**.** *Given a forest F on n vertices and q query paths specified by their endpoints, the maximum edge weight on each query path can be determined deterministically using $O(n + q)$ comparisons.*

3 The Certificate Procedure

3.1 Overview and Formal Algorithm

The entire Certificate procedure is predicated around finding cycles of edges and deleting their heaviest edges. As mentioned in the technical overview, the algorithm seeks to iteratively *project* vertices to smaller and smaller vertex sets to aid in cycle discovery. To this end, the algorithm maintains a subset P of the vertices, called *portals*. For each vertex v, it may maintain an incident edge $\sigma(v)$ connecting v to a portal, called the *spoke* of v. Let $\pi(v)$ denote the portal at the other endpoint of the spoke; if v has no spoke, we write $\sigma(v) = \pi(v) = \perp$. For an edge $e = (u, v)$ whose endpoints both have spokes, the algorithm may *project* e to an edge of weight $w(e)$ with endpoints $\pi(u)$ and $\pi(v)$ in a child instance. Let $p(e) = \max\{w(\sigma(u)), w(\sigma(v))\}$ be the larger of the two spoke weights; we say the projection is *valid* if $p(e) \leq w(e)$.

The input to the certificate procedure is an undirected multigraph G. Each edge $e \in E(G)$ has a weight $w(e)$ and a *projection record*, which stores, for each preceding recursion level, the pair of spokes used to project e at that level. We say that e is *admissible* if every spoke in its projection record has weight at most $w(e)$, or equivalently, if all projections of e at preceding recursion levels are valid. The admissibility test scans the projection record from the root toward the current call and reports the first recursion level at which it encounters an invalid projection, if any.

The certificate procedure maintains a forest $F \subseteq E(G)$ consisting of admissible edges and a set R of rejected edges throughout its entire execution. The forest and rejected edges (as we will see later) get updated after each round is completed. An edge is *rejected* if it fails the admissibility test. Such edges are propagated upward until they reach the recursion level at which their first invalid projection was performed.

An edge $e = (u, v)$ is *supported* by F if $F_{\leq w(e)}$ connects u and v. As we will show, F together with the unsupported edges contains an MSF of G. We may therefore filter out (discard) every supported edge outside F. Thus, the goal of the certificate procedure is to construct a forest F using only $O(m)$ comparisons that supports most of the edges. The following lemma bounds the number of retained edges, the total size of the edge sets passed to exact (recursive) MSF calls, and the number of weight comparisons performed outside those calls.

Lemma 3.1. *Let G be an undirected multigraph with n vertices, m edges, and no isolated vertices. For every fixed constant $\varepsilon > 0$, there exists a deterministic certificate procedure that outputs a forest $F \subseteq E(G)$. The procedure may invoke exact MSF calls on edge sets $E_1, \ldots, E_\ell$, each defined on a subset of $V(G)$. Apart from these calls, the procedure performs $O_\varepsilon(m)$ weight comparisons. Let $U(F) \subseteq E(G)$ denote the set of edges unsupported by F. The procedure guarantees that*

$$|F| + |U(F)| + \sum_{i=1}^{\ell} |E_i| \leq \varepsilon m + C_\varepsilon \sqrt{nm},$$

where C_ε is a constant depending only on ε.

The procedure is outlined in Algorithm 1. For a call at recursion level d, we set $t = \gamma \cdot d^3, r = 8t^3$, where γ is a fixed constant. The algorithm consists of t^2 rounds, partitioned into t blocks of t rounds each. After initialization, we schedule each edge of G to take action in *exactly one round*.

Let A be the set of edges scheduled for the current round. Each edge $e = (u, v) \in A$ takes one of two actions. The find action searches for a valid projection, while the project action projects e to $(\pi(u), \pi(v))$ using the current spokes, regardless of whether the projection is valid. Only edges taking the project action enter a child instance (i.e., a recursive Certificate call). Ideally, we would find a valid projection for each edge before projecting it. However, even checking validity of the current projections for k edges can take $\Theta(k)$ weight comparisons. At a level handling $\Theta(m)$ edges, these checks alone can therefore cost $\Theta(m)$ comparisons per level.

At the start of each round, the *preferred action* of an edge $e = (u, v)$ is find if either endpoint has no spoke or if both spokes lead to the same portal. When the spokes lead to distinct portals, the preferred action is find if $F_{\leq p(e)}$ connects u and v, and project otherwise. This preceding preference rule will be incredibly important in the bounding the number of unsupported edges. Unfortunately, exactly computing the preferred action of each edge will be prohibitively expensive, and thus the final algorithm *estimates* the preferred action of each edge. Its estimate is always find when a spoke is missing or both spokes lead to the same portal. We will later show that the estimate is still correct for *most edges* whose spokes lead to distinct portals.

Within each block (i.e., a sequence of t rounds) we also maintain a *batch* B of edges, initialized to be the empty set. Edges whose estimated preferred action is find are collected in this batch B

and processed at the end of the block. Each edge $e = (u, v)$ whose estimated preferred action is project is instead projected to $(\pi(u), \pi(v))$, forming a graph G' on the portal set P. We recursively invoke the certificate procedure on G', obtaining a forest F' and a set of rejected edges R'. Because F' and R' are projections of edges in G, we first lift them back to their corresponding edges in G. We further let $R'' \subseteq R'$ denote the edges whose first invalid projection occurs at the *current recursion level*. We then incorporate F' and R'' into F via an exact MSF call. The remaining rejected edges $R' \setminus R''$ must have had their first invalid projections at earlier recursive levels, and thus are added to R for propagation to those levels.

At the end of each block, we process the batch B. This procedure identifies a set D of *declared edges* to be incorporated into F via an exact MSF call, introduces a set of new portals, and updates the spokes so that every nonrejected edge in B is either supported by F or has a valid projection. Any edges found to be inadmissible during this process are added to R.

Algorithm 1: Certificate(G)

```
if E(G) = ∅ then
    return (∅, ∅)
F ← ∅, R ← ∅, P ← ∅.
Initialize σ(v) ← ⊥, π(v) ← ⊥ for every v ∈ V(G).
Schedule each edge of G to take action in one of the t² rounds.
for t blocks do
    B ← ∅.
    for t rounds do
        Let A be the set of edges scheduled for this round.
        Determine (estimate) the preferred action for each edge in A.
        Add all edges whose preferred action is find to the batch B.
        Initialize an empty graph G' on vertex set P.
        foreach e = (u, v) whose preferred action is project do
            Add the projected edge (π(u), π(v)) to G' with weight w(e).
            Append the spokes of u and v to the projection record.
        F', R' ← Certificate(G').
        Lift the edges of F', R' back to their corresponding edges in G.
        Let R'' ⊆ R' be the set of edges whose first invalid projection occurs at this level.
        F ← MSF(G[F ∪ F' ∪ R'']), R ← R ∪ (R' \ R'').
    Process the batch B, obtaining a set D of declared edges, a set P' of new portals, and a
     set R' of new rejected edges. Spokes are also updated.
    F ← MSF(G[F ∪ D]), P ← P ∪ P', R ← R ∪ R'.
return (F, R).
```

3.2 The Batch Processing

We first describe how to process a batch B of edges, then give an efficient implementation that bounds the number of weight comparisons and admissibility tests.

For each vertex v, group the incident edges of B by their other endpoint (this is only necessary because there may be parallel edges from v to the same vertex). We define the representative of a group to be its *minimum weight* admissible edge, if one exists. At v, select the $k = r^2$ lightest

representatives, or all representatives if there are fewer than k. Let S_v be the set of neighbors of v reached by the selected edges. The implementation tests admissibility of edges as needed. Every selected edge passes the test, and every edge that fails the test is rejected. When an edge fails an admissibility test, we store its first invalid projection level and *reuse* the recorded failure on every later test request for that edge in the batch. We *declare* the selected edges, adding these to a set D. In this way, D becomes the union of declared edges over all vertices. As explained in Algorithm 1, we then replace F by an MSF of $F \cup D$ via an exact MSF call.

We next select a set P' of portals that intersects every S_v of size k. Indeed, to do this, we call each such set S_v *covered* once it contains a selected portal. We find a small portal set P' as follows: while some vertex belongs to at least r uncovered sets, select it as a portal. Each selection covers at least r additional sets, so this step selects at most n/r portals. At termination, each vertex belongs to fewer than r uncovered sets. Thus the remaining sets have fewer than nr incidences in total. Each has size $k = r^2$, so fewer than n/r sets remain. Select one vertex from each set that is still uncovered. Altogether, this means that $|P'| \leq 2n/r$. We then update the existing portal set P by adding P' to P.

For each v with $|S_v| = k$, let x be the first portal selected that belongs to S_v. The selected edge from v to x is the new *candidate spoke* of v. If v has no spoke, we assign it this candidate spoke. Otherwise, v chooses the lighter of the candidate spoke and its current spoke $\sigma(v)$.

We obtain the following guarantee:

Lemma 3.2. *Let B be a batch of edges of a graph G on n vertices. The batch procedure declares a set $D \subseteq B$ of at most $kn = r^2 n$ admissible edges and selects a set P' of at most $2n/r$ portals. Next, let $R' \subseteq B$ denote the edges that fail an admissibility test during the batch. After replacing F by $\mathsf{MSF}(G[F \cup D])$, every edge $e = (u, v) \in B \setminus R'$ satisfies at least one of the following conditions.*

- *The edge e is supported by F, that is, $F_{\leq w(e)}$ connects u and v.*
- *Both endpoints have spokes and $p(e) = \max\{w(\sigma(u)), w(\sigma(v))\} \leq w(e)$, so e admits a valid projection.*

Proof. Each vertex selects at most k edges, so $|D| \leq kn$. The portal bound $|P'| \leq 2n/r$ follows from the covering argument just discussed above.

We define R' to be the set of edges that failed admissibility test during the selection procedure. Now consider any edge $e = (u, v) \in B \setminus R'$. Note that e may or may not pass the admissibility test, but any untested edge can be made admissible without changing the execution or output. If $u \in S_v$ or $v \in S_u$, then one endpoint selected the representative of the group containing e. That representative joins u and v and has weight at most $w(e)$. Since it belongs to D, Fact 2.5 implies that the updated forest connects u and v in $F_{\leq w(e)}$.

Otherwise, neither endpoint selected the group containing e. It follows that each endpoint selected k representatives, all of weight at most $w(e)$. Thus $|S_u| = |S_v| = k$, and each endpoint receives a candidate spoke of weight at most $w(e)$. The spoke update applied at the end can only improve on this bound, so $p(e) \leq w(e)$. □

Efficient implementation. For each vertex v, consider its incident edges in B, grouped by their other endpoint. Our goal is to select the $k = r^2$ lightest edges among the minimum weigh admissible edges of these groups. We now describe how to implement this efficiently without performing an admissibility test on every edge. We bound the number of successful admissibility tests and failed tests will be charged separately.

Lemma 3.3. *Suppose m edges of unknown admissibility are partitioned into groups. For each group containing an admissible edge, let its representative be its minimum weight admissible edge. Given an integer $k \geq 0$, we can select the k lightest representatives, or all representatives if there are fewer than k, using $O(k \log(m+1))$ successful admissibility tests and $O(m)$ additional weight comparisons.*

Proof. We use a recursive procedure. If $k = 0$, return the empty set. If $m \leq k$, test every edge and return the minimum weight admissible edge from each group that has one.

Otherwise, use deterministic median selection [BFP+72] to partition the edges into a lower half L of size $\lfloor m/2 \rfloor$ and an upper half H, using the fixed edge order to break weight ties. Scan L, testing edges until either k distinct groups have passed a test or the scan is exhausted. Once an edge passes, skip the remaining edges of its group for the rest of this scan.

If k groups pass, at least k groups have representatives in L, so all k desired representatives lie in L. We then recurse on L with quota k.

Otherwise, let $k' < k$ be the number of groups that passed. These are exactly the groups with an admissible edge in L, and each contributes a representative to the answer. For each such group, recurse with quota one on its edges in L to find that representative. Remove all edges of these k' groups from H, and recurse on the remaining edges of H with quota $k - k'$. Every remaining representative must come from this last call.

The correctness follows immediately. We now analyze the number of comparisons. Deterministic median selection requires $O(m)$ comparisons (see [BFP+72]). First consider a call with quota one. Such a call never starts another quota-1 group search. At each level, it performs $O(m)$ comparisons and at most one successful admissibility test, and recurses on half of the edges. Therefore, the call uses $O(m)$ comparisons and $O(\log(m))$ successful tests in total.

Now consider an arbitrary call with quota k. At each level, the procedure performs $O(m)$ comparisons and at most k successful admissibility tests. Moreover, there is at most one recursive call with quota more than one, and the call contains at most half of the edges. All other recursive calls are quota-1 group searches which operate on disjoint subsets of the discarded half. Therefore, the main recursion uses $O(m)$ comparisons and $O(k \log(m))$ successful tests. The quota-1 group searches contribute another $O(m)$ comparisons since their inputs are disjoint, and $O(k \log(m))$ successful tests since there are at most k group searches in total. This completes the proof. □

We apply Lemma 3.3 to the edges incident to each vertex v with quota k. The total number of comparisons is therefore bounded by $\sum_v O(\deg_B(v)) = O(|B|)$. In addition, the total number of successful admissibility tests is at most

$$\sum_v O(k \log(\deg_B(v) + 1)) \leq \sum_v O\left(k\sqrt{\deg_B(v)}\right) = O\left(r^2\sqrt{|B| \cdot n}\right),$$

where we used $\log(x+1) \leq \sqrt{x}$, Cauchy–Schwarz, and the fact that $k = r^2$.

Throughout batch processing, comparisons other than admissibility tests arise from two sources: $O(|B|)$ from the bound above and $O(n)$ comparisons for updating the spokes, yielding a total of $O(n + |B|)$ comparisons.

3.3 Bounding Unsupported Edges

Recall that an edge $e = (u, v)$ is supported by F if $F_{\leq w(e)}$ connects u and v. Our goal is to ensure that most edges are supported by the final forest F. In this section, we characterize the unsupported edges and bound their number.

Fix an edge $e = (u, v) \in E(G)$. For the analysis, we consider its preferred action at the start of each round, even though each edge is assigned to take action in only one round. Recall that its preferred action is find if (1) either endpoint has no spoke, (2) $\pi(u) = \pi(v)$, or (3) $F_{\leq p(e)}$ connects u, v; otherwise, it is project.

Recall that the t^2 rounds are partitioned into t blocks of t rounds each. Within each block, the spokes remain unchanged; they are updated only at the end of the block when a batch is processed. In addition, the connectivity in F only improves as the rounds progress – that is to say, connectivity in $F_{\leq a}$ is preserved at every possible choice of the threshold a. This means that whenever a spoke s is declared, going forward, it will always be the case that $F_{\leq w(s)}$ connects its endpoints. Thus, within each block, the preferred action of e is project for an initial sequence of rounds, followed by find for the remaining rounds, where either sequence may be empty.

We now define the notion of a *bad* round for e; all other rounds are called *good*.

Definition 3.4. A round is *bad* for e if either of the following holds:

- It is the last round in its block with a project preference.
- The preferred action is find, and the block is the last block containing any find preference.

By definition, there are at most $2t$ bad rounds. Indeed, there are at most t blocks, each containing at most one bad project round. In addition, bad find rounds can occur in at most one block, which contains at most t rounds.

Recall that the algorithm does not compute the true preferred action for every edge, but instead uses estimates. We say that an *error* occurs for an edge e if either the schedule assigns e to take action in a bad round, or the estimated preferred action in its assigned round differs from the true preferred action. We establish the following key property: if no error occurs for e throughout the recursion of Algorithm 1, then e is supported by the final forest F.

To prove this, we first establish the following useful properties.

Claim 3.5. *For a call* Certificate(G)*, every edge in the returned forest F is admissible.*

Proof. We consider all edges that may be inserted into F. Every declared edge in D passes the admissibility test during batch processing, as described in Section 3.2. All the returned child forest edges are admissible by induction. A rejected edge is inserted at the first recursion level where it performs an invalid projection, and hence it is admissible before that projection. Therefore, every edge inserted into F is admissible. □

Claim 3.6. *For an edge (u, v) with projection $(\pi(u), \pi(v))$, if the child recursion supports the projected edge, then on return $F_{\leq \max\{w(e), p(e)\}}$ connects u and v.*

Proof. Since the projected edge is supported, there exists a path in the child forest connecting $\pi(u)$ and $\pi(v)$ using only edges of weight at most $w(e)$. Consider every edge $f' = (\pi(x), \pi(y))$ on this path, note that f' is first lifted back to the corresponding edge $f = (x, y)$ in G before being incorporated into F. By Claim 3.5, f' is admissible, so both spokes $\sigma(x)$ and $\sigma(y)$ used in this lifting have weight at most $w(f) \leq w(e)$. Thus, replacing each edge f' on the child forest path with the walk $(\pi(x), x, y, \pi(y))$ using $\sigma(x)$ the lifted edge of f' and $\sigma(y)$. Concatenating these walks and adding $\sigma(u)$ and $\sigma(v)$ gives a walk from u to v with all weights at most $W = \max\{w(e), p(e)\}$. By the preceding arguments, every edge on this path has weight at most $w(e)$, except for $\sigma(u), \sigma(v)$. Moreover, these edges are either spokes or child forest edges, which are all incorporated into F via exact MSF calls (note that spokes can only arise from declared edges D during batch processing). By Fact 2.5, u and v are connected in $F_{\leq \max\{w(e), p(e)\}}$. □

Now we are ready to prove the key lemma, which shows that every unsupported or rejected edge can be charged to an error.

Lemma 3.7. *Consider a root instance* Certificate(G) *and an edge* $e \in E(G)$*. Suppose no error occurs for* e *or any of its projections throughout the recursion, then* e *is supported by the final forest* F *returned by the algorithm. If a rejected edge has its first invalid projection at some recursion level, then an error occurs for that edge at that level.*

Proof. Fix an edge $e \in E(G)$, and suppose that throughout the recursion, no error occurs for e or any of its projections, none of them is rejected, and all their estimated preferred actions are correct. Note that an edge is projected and passed to exactly one child recursion instance if its preferred action at its assigned round is project; if its preferred action is find, it does not appear in any subsequent recursion instance. We prove by backward induction on the recursion level that the projection $e' = (u', v')$ of e at every level is supported.

Consider the last recursion level in which the projection of e exists. Its preferred action must be find. Thus, e' is included in the batch and processed at the end of the block. By Lemma 3.2, either e' is supported, or its spokes satisfy $p(e') \le w(e')$. In the former case, we are done. In the latter case, since the assigned round is good, there must be a later block in which e' prefers find. In that round, either the spokes coincide ($\pi(u') = \pi(v')$), or $F_{\le p(e')}$ connects u', v'. Since $p(e')$ can only decrease, we still have $p(e') \le w(e')$. In the former case, the two spokes meet at a common portal, so $F_{\le p(e')}$ connects u' and v'. Therefore, in either case, $F_{\le p(e')}$ and hence $F_{\le w(e')}$ connects u', v', and e' is supported.

Now consider a recursion level where the preferred action of e' is project. In its assigned round, e' is projected to $(\pi(u'), \pi(v'))$ and passed to a child recursion instance. By Claim 3.6, on return, $F_{\le \max\{w(e'), p(e')\}}$ connects u' and v'. If $p(e') \le w(e')$, e' is supported as desired. Otherwise, $F_{\le p(e')}$ connects u' and v'. However, since the assigned round is good, the preferred action in the next round must also be project, contradicting the fact $F_{\le p(e')}$ connects u', v'. Therefore, e' must be supported. The first claim follows by backward induction up to the root recursion instance.

For a rejected edge e, suppose an admissibility test fails and identifies the first level where an invalid projection occurs. When e is projected in some round at this level, we have $p(e) > w(e)$. At the end of that round, the edge returns from the recursive call and is incorporated into the forest F via an exact MSF call. By Fact 2.5, u and v are connected in $F_{\le p(e)}$ at that point. Since the spokes remain unchanged in the block containing this round, $p(e)$ remains fixed, and u and v remain connected in $F_{\le p(e)}$. Thus, all subsequent rounds in the block prefer find. If the estimated preferred action is correct, the assigned round must be bad. Otherwise, the estimate is incorrect. In either case, an error occurs. This also proves the first claim for a rejected edge. □

3.4 Building the Schedule

The preceding sections have mostly dealt with the complexity of ensuring the algorithm behaves well, provided the edges are scheduled to their rounds appropriately. In this section, we now establish a deterministic schedule that assigns most edges to good rounds. There is a subtle circularity: the preferred actions of edges depend on the schedule, which in turn determines which rounds are good or bad. To reason about this dependence, we introduce an object called the *history*.

Definition 3.8 (History)**.** Consider a call Certificate(G) on a graph G with n vertices and m edges. We index the t^2 rounds by $i \in [t^2]$. We define the history to be

$$H = (\sigma_i, F_i, a_i)_{i \in [t^2]},$$

where:

- σ_i is a vector of length n, where $\sigma_i(v) \in E(G) \cup \perp$ specifies the spoke of vertex v at the start of round i.
- $F_i \subseteq E(G)$ is the forest F maintained by the algorithm at the start of round i.
- a_i is a vector of length m, where $a_i(e) \in \{\mathsf{find}, \mathsf{project}\}$ specifies the *true preferred action* of edge e at the start of round i.

Note that the history completely determines which rounds are good or bad for each edge. Ultimately, we will show that the number of possible histories is not too large. This will allow us to show that there exists a deterministic schedule that satisfies the desired property for every possible history of the algorithm.

Bounding the number of histories. We use the following bound on the number of possible vectors of maxima, appearing in [GYY80] attributed to Fredman. We include a proof for completeness.

Lemma 3.9. *Let $S_1, \ldots, S_m$ be arbitrary nonempty subsets of a ground set of size n. For each total order π over the ground set, let x_π be the vector that records the names of the maxima for each set S_i. Then, as π ranges over all possible total orders, the number of distinct vectors x_π is at most $\binom{n+m-1}{m}$.*

Proof. For each total order π, let y_π record, for each element, the number of sets in which it is the maximum. Since $\sum_j y_\pi(j) = m$, there are at most $\binom{n+m-1}{m}$ possible vectors y_π. We claim that y_π uniquely determines x_π.

Suppose otherwise that there exist two total orders π_1, π_2 such that $y_{\pi_1} = y_{\pi_2}$ but $x_{\pi_1} \neq x_{\pi_2}$. Construct a directed multigraph as follows: for each set S_i whose maxima under π_1 and π_2 differ, add a directed edge from its maximum under π_2 to its maximum under π_1. Thus, every edge points to a strictly larger element in π_1. However, $y_{\pi_1} = y_{\pi_2}$ implies that every element has equal in-degree and out-degree. Since the graph is nonempty, it must contain a directed cycle, a contradiction $\square$

With this in hand, we now bound the number of possible histories.

Lemma 3.10. *Consider a call $\mathsf{Certificate}(G)$ on a graph G with n vertices and m edges. Let $\mathcal{H}$ denote the set of all possible histories of this call, obtained by enumerating every sequence of possible spokes and forests that appear at each round, and then enumerating every possible list of preferences induced by these spokes and forests. Then*

$$\log |\mathcal{H}| = O(t^2 \sqrt{mn}).$$

Proof. First we bound the number of possible sequences $(\sigma_i)_{i \in [t^2]}$. In each round, the spoke $\sigma_i(v)$ has at most $\deg_G(v) + 1$ possible values. Thus, the total number of such sequences is at most

$$\left(\prod_{v \in V} (\deg(v) + 1) \right)^{t^2}.$$

Taking logarithms, we obtain

$$\begin{aligned}\log\left(\prod_{v\in V(G)}(\deg(v)+1)\right)^{t^2} &= t^2\cdot\sum_{v\in V(G)}\log(\deg_G(v)+1)\\ &\le t^2\cdot\sum_{v\in V(G)}\sqrt{\deg_G(v)} && (\log(x+1)\le\sqrt{x})\\ &\le t^2\sqrt{2mn}. && (\text{Cauchy–Schwarz})\end{aligned}$$

Next, we bound the number of possible sequences $(F_i)_{i\in[t^2]}$. Since each forest uses at most $n-1$ edges out of the m edges in the graph, the total number of possible sequences is at most

$$\binom{m+n-1}{n-1}^{t^2}.$$

Taking logarithms, we obtain

$$\log\binom{m+n-1}{n-1}^{t^2}\le t^2\log\binom{m+n}{m}\le 2t^2\sqrt{mn},$$

where we used Fact 2.1.

Finally, we bound the number of possible true preference vectors $(a_i)_{i\in[t^2]}$. Fix the spokes σ_i and the forest F_i at the start of round i. For each edge $e=(u,v)\in E(G)$, recall that its preferred action is find if (1) either endpoint has no spoke, (2) $\pi(u)=\pi(v)$, or (3) $F_{\le p(e)}$ connects u,v; otherwise, it is project. The first two cases are completely determined by the spokes. For any remaining edges that are not connected in F_i, their action is forced to be project. So it remains only to analyze how edges in the same component that are also in case (3) can be assigned to project and find.

Consider a ground set T of size at most $2n$ consisting of all the spokes and edges of F_i. The total order is determined by edge weights, with spokes ranked higher than forest edges in case of ties. For each edge $e=(u,v)\in E(G)$, let $S_e\subseteq T$ consist of the spokes $\sigma_i(u),\sigma_i(v)$ and all edges on the u to v path in F_i, if such a path exists. The preferred action of e is find precisely when the maximum element of the set S_e is a spoke (either $\sigma_i(u)$ or $\sigma_i(v)$). Indeed, this occurs precisely when the heaviest spoke is at least as heavy as every edge on the u to v path in F_i, which is equivalent to u,v being connected in $(F_i)_{\le p(e)}$. Conversely, if the maximum weight edge in S_e is a path edge (weight strictly larger than both spokes), then u and v are not connected in $(F_i)_{\le p(e)}$. Note that here we use the fact that there is a *unique* path between u and v in F_i (and that u,v are connected - as disconnected u,v are already determined by the previous discussion).

This now exactly mirrors the setting of Lemma 3.9 on a ground set of size at most $2n$ with m sets. Consequently, for any fixed choices of the spokes and forests, the number of possible true preference vectors is at most

$$\binom{m+2n-1}{m}^{t^2}.$$

Taking logarithms, we obtain

$$\log\binom{m+2n-1}{m}^{t^2}\le t^2\log\binom{m+2n}{m}\le 2t^2\sqrt{2mn},$$

where we used Fact 2.1.

Combining the three bounds, we obtain

$$\ln|\mathcal{H}|\le t^2\sqrt{2mn}+2t^2\sqrt{mn}+2t^2\sqrt{2mn}=O(t^2\sqrt{mn})$$

as desired. □

Finding a good schedule. We next show that, at the start of the algorithm, we can fix a deterministic schedule under which most edges perform their actions in good rounds for every possible history.

Lemma 3.11. *Consider a call* Certificate(G) *on a graph G with n vertices and m edges. For each history H and schedule $\rho : E(G) \to [t^2]$, we let $X_H(\rho)$ denote the number of edges whose action is scheduled in one of their bad rounds. Then there exists a schedule ρ^* such that for every $H \in \mathcal{H}$,*

$$X_H(\rho^*) \leq \frac{\log |\mathcal{H}| + 2m/t}{\log 2}.$$

Proof. Fix an arbitrary history $H \in \mathcal{H}$. Recall that this determines the good and bad rounds for each edge. Now consider a random schedule ρ that independently assigns each edge to a uniformly random round. For each edge $e \in E(G)$, let Y_e be a random variable with $Y_e = 1$ if e is assigned to a bad round, and $Y_e = 0$ otherwise. Since each edge has at most $2t$ bad rounds out of t^2 rounds, we have

$$\mathbb{E}_\rho \left[2^{Y_e}\right] = 1 + \Pr[Y_e = 1] \leq 1 + \frac{2t}{t^2} = 1 + \frac{2}{t}.$$

Since $X_H(\rho) = \sum_{e \in E(G)} Y_e$, and the assignments are independent, it follows that

$$\mathbb{E}_\rho \left[2^{X_H(\rho)}\right] = \prod_{e \in E(G)} \mathbb{E}_\rho \left[2^{Y_e}\right] \leq \left(1 + \frac{2}{t}\right)^m \leq e^{2m/t}.$$

Summing over all histories gives

$$\mathbb{E}_\rho \left[\sum_{H \in \mathcal{H}} 2^{X_H(\rho)}\right] \leq |\mathcal{H}| \cdot e^{2m/t}.$$

By averaging, this implies that there exists a schedule ρ^* such that

$$\sum_{H \in \mathcal{H}} 2^{X_H(\rho^*)} \leq |\mathcal{H}| \cdot e^{2m/t}.$$

In particular, for every $H \in \mathcal{H}$,

$$2^{X_H(\rho^*)} \leq |\mathcal{H}| \cdot e^{2m/t},$$

and hence

$$X_H(\rho^*) \leq \frac{\log |\mathcal{H}| + 2m/t}{\log 2}$$

□

The key observation is that this allows us to choose a schedule ρ^* *independent* of the actual execution of the certificate call. In particular, we can enumerate all possible schedules and check all possible histories to find a good schedule. Although this may incur enormous computational overhead, it requires no comparisons, and we are only concerned with the number of comparisons made by the algorithm.

3.5 Estimating Preferred Actions

In the preceding section, we described how to choose a schedule that determines the round in which each edge takes action. In this section, we show how to efficiently estimate the preferred actions, ensuring that the estimates are correct for most edges.

Consider a call $\mathsf{Certificate}(G)$ on a graph G with n vertices and m edges. Fix a round i, and let A_i denote the set of edges assigned to this round, with $m_i = |A_i|$. Consider the vector a_i of length m_i, which specifies the preferred action for each edge in A_i. By the argument in Lemma 3.10 (via Lemma 3.9), conditioned on the current forest F_i and spokes σ_i, the number of possible preference vectors is at most $e^{O(\sqrt{m_i n})}$. Let $\mathcal{A}$ denote the family of all such vectors.

We first show that, by querying the true preferred actions of a small subset of edges, we can identify a vector in $\mathcal{A}$ that differs from a_i on at most m_i/t coordinates.

Claim 3.12. *There exists a subset $S^* \subseteq A_i$ of size $O(t\sqrt{m_i n})$ such that any two vectors $a, b \in \mathcal{A}$ that agree on all coordinates corresponding to S^* differ in at most m_i/t coordinates.*

Proof. We sample $k = 2t(\log|\mathcal{A}| + 1)$ edges from A_i independently and uniformly at random. Let S be the set of sampled edges.

Fix any two vectors $a, b \in \mathcal{A}$ that differ in more than m_i/t coordinates. The probability that they agree on all coordinates corresponding to S is at most $(1 - 1/t)^k \le e^{-k/t}$. By a union bound over all pairs $a, b \in \mathcal{A}$, the probability that there exist two vectors differing in more than m_i/t coordinates but agreeing on all coordinates in S is at most $|\mathcal{A}|^2 e^{-k/t} < 1$. Therefore, there exists a choice S^* satisfying the desired property. Since $|\mathcal{A}| = e^{O(\sqrt{m_i n})}$, we have $|S^*| = O(t\sqrt{m_i n})$. □

Again, the algorithm can explicitly find such a set S^* by enumerating and checking, without making any comparisons. It then queries the true preferred actions for edges in S^* and selects any vector in $\mathcal{A}$ consistent with the queried answers as the estimate. These queries correspond precisely to path maximum queries on the forest F, which can be answered using the algorithm from Lemma 2.9. In round i, there are $O(t\sqrt{m_i n})$ queries, and it requires $O(n + t\sqrt{m_i n})$ comparisons. Summing over all t^2 rounds, the total number of comparisons is at most

$$\sum_{i=1}^{t^2} O(n + t\sqrt{m_i n}) = O(t^2 n + t^2\sqrt{mn})$$

by Cauchy–Schwarz. In addition, the total number of edges with incorrect estimates is at most

$$\sum_{i=1}^{t^2} \frac{m_i}{t} = \frac{m}{t}.$$

3.6 Putting the Pieces Together

We now prove Lemma 3.1. Recall that the parameters are set to $t = \gamma \cdot d^3, r = 8t^3$, where γ is a fixed constant. We also observe that $n \le 2m$ since the initial graph has no isolated vertices.

For a call on n vertices, the batch processing procedure is executed t times, each introducing at most $2n/r$ new portals. Thus, the total number of portals is bounded by $2nt/r$. Since the algorithm makes t^2 recursive calls, the total number of vertices across all child instances is bounded by

$$t^2 \cdot \frac{2nt}{r} = \frac{2nt^3}{r} = \frac{n}{4}$$

by the parameter choices. Therefore, the total number of vertices decreases by a factor of 4 in each recursion level.

For each recursion level d, let n_d be the total number of vertices and m_d the total number of edges. By the above argument, we have $n_d \le 4^{-(d-1)} \cdot n$ and $m_d \le m$ since each edge is passed to at most one child instance. For each call c, let d_c denote its recursion level, n_c and m_c its numbers of vertices and edges, and $t_c = \gamma \cdot d_c^3, r_c = 8t_c^3$ its parameter values. We obtain the following useful property.

Claim 3.13.

$$\sum_{calls\ c} poly(d_c, t_c, r_c)(n_c + \sqrt{m_c n_c}) = O_\gamma(\sqrt{mn}).$$

Proof. Fix a recursion level d, and consider all calls c at level d, we have

$$\sum_c \text{poly}(d_c, t_c, r_c)(n_c + \sqrt{m_c n_c}) \le O_\gamma\left(\text{poly}(d)(n_d + \sqrt{m_d n_d})\right).$$

by Cauchy–Schwarz and the fact that $t_c, r_c = \text{poly}(d_c, \gamma)$. Since $n_d \le 4^{-(d-1)} \cdot n$ and $n \le 2m$, this is at most

$$O_\gamma\left(\frac{\text{poly}(d)}{2^{d-1}} \cdot \sqrt{mn}\right).$$

Summing over all recursion levels, we obtain

$$\sum_{\text{calls } c} \text{poly}(d_c)(n_c + \sqrt{m_c n_c}) \le \sum_{d \ge 1} O_\gamma\left(\frac{\text{poly}(d)}{2^{d-1}} \cdot \sqrt{mn}\right) = O_\gamma(\sqrt{mn}).$$

□

Bounding the number of errors. For a call c, an error on an edge e can arise from two sources: either the schedule assigns e to take action in a bad round, or the estimated preferred action on its assigned round is incorrect. By Lemmas 3.10 and 3.11, the number of errors of the first type is at most

$$\frac{\log|\mathcal{H}| + 2m_c/t_c}{\log 2} \le O(t_c^2\sqrt{m_c n_c}) + \frac{2m_c}{t_c \log 2}.$$

By the analysis in Section 3.5, the number of errors of the second type is at most m_c/t_c. Therefore, the total number of errors in call c is at most

$$O(t_c^2\sqrt{m_c n_c}) + \left(1 + \frac{2}{\log 2}\right)\frac{m_c}{t_c} \le O(t_c^2\sqrt{m_c n_c}) + \frac{4m_c}{t_c}.$$

Summing over all calls, we obtain

$$\sum_{\text{calls } c} O(t_c^2\sqrt{m_c n_c}) + \frac{4m_c}{t_c} \le O_\gamma(\sqrt{mn}) + \sum_{d \ge 1} \frac{4m_d}{\gamma \cdot d^3} \le O_\gamma(\sqrt{mn}) + \frac{8m}{\gamma}.$$

By Lemma 3.7, the same bound applies to both the number of unsupported edges.

Bounding the number of comparisons. We now bound the number of comparisons performed in a call c, excluding the exact MSF calls.

First, consider the comparisons used in batch processing. Let $B_1, \dots, B_{t_c}$ denote the batches. By the analysis in Section 3.2, processing a batch B_i requires $O(n_c + |B_i|)$ comparisons. The $O(n_c)$ term contributes $O(t_c n_c)$ over the entire call. For the $O(|B_i|)$ term, since each edge enters at most one batch throughout the entire recursion, these comparisons contribute $O(m)$ in total across all calls. In addition, by the analysis in Section 3.2, processing B_i requires $O(r_c^2\sqrt{|B_i| \cdot n_c})$ successful admissibility tests. Each successful test requires $O(d_c)$ comparisons. Therefore, the total number of comparisons due to successful tests is at most

$$O(d_c) \cdot \sum_{i=1}^{t_c} O\left(r_c^2\sqrt{|B_i| \cdot n_c}\right) = O\left(d_c r_c^2 \sqrt{t_c m_c n_c}\right)$$

by Cauchy–Schwarz.

Next, by the analysis in Section 3.5, estimating the preferred actions requires $O(t_c^2 n_c + t_c^2\sqrt{m_c n_c})$ comparisons. Summing these bounds over all calls, we obtain

$$\sum_{\text{calls } c} O\left(t_c n_c + d_c r_c^2\sqrt{t_c m_c n_c} + t_c^2 n_c + t_c^2\sqrt{m_c n_c}\right) = O_\gamma(\sqrt{mn}).$$

Finally, consider the comparisons incurred by failed admissibility tests. Each failed test causes an edge to be rejected. If the first invalid projection of a rejected edge occurs at recursion level d, then the test costs at most $O(d)$ comparisons. By Lemma 3.7 and the preceding bound on the number of errors, the total number of comparisons due to failed tests is at most

$$\sum_{\text{calls } c} O(d_c) \cdot \left(O(t_c^2\sqrt{m_c n_c}) + \frac{4m_c}{t_c}\right) = O_\gamma(\sqrt{mn}) + O\left(\sum_{d \ge 1} \frac{4d m_d}{\gamma \cdot d^3}\right) = O_\gamma(\sqrt{mn}) + O(m).$$

Combining the above bounds, the total number of comparisons is at most $O_\gamma(\sqrt{mn}) + O(m) = O(m)$ since $n \le 2m$.

Bounding the total size of exact MSF calls. For a call c, the algorithm makes an exact MSF call at each of the t_c^2 rounds. Each invocation takes as input the old forest and the child forest, each containing at most $n_c - 1$ edges, together with the rejected edges whose first invalid projection occurs at the current level. In addition, at the end of each block, the algorithm makes an exact MSF call on the old forest, containing at most $n_c - 1$ edges, and the set of declared edges D of size at most $r_c^2 n_c$. Thus, across all calls where $m_c > 0$, the total input size of all exact MSF calls is at most

$$[\#\text{ rejected edges}] + \sum_{\text{calls } c : m_c > 0} \left(2t_c^2(n_c - 1) + t_c(n_c - 1 + r_c^2 n_c)\right)$$

By Lemma 3.7 and the preceding bound on the number of errors, the first term is at most $O_\gamma(\sqrt{mn}) + 8m/\gamma$. The second term is at most $O_\gamma(\sqrt{mn})$. Therefore, the total input size of all exact MSF calls is bounded by $O_\gamma(\sqrt{mn}) + 8m/\gamma$.

Overall, we have

$$|F| + |U(F)| + \sum_{i=1}^{\ell} |E_i| \le O_\gamma(\sqrt{mn}) + \frac{16m}{\gamma}.$$

Taking $\gamma \ge 16/\varepsilon$ proves Lemma 3.1.

4 The Final Algorithm

With the certificate algorithm established, we are ready to present and prove our main result.

Theorem 4.1. *Let $\mathcal{T}^*(m,n)$ be the deterministic decision-tree complexity of the* MSF *problem on graphs with m edges and n vertices. Then $\mathcal{T}^*(m,n) = O(m)$.*

Combined with the result of [PR02], this immediately yields the following corollary.

Corollary 4.2. *There exists a uniform deterministic $O(m)$ time algorithm for computing the MST of any connected graph with m edges and n vertices, in both the RAM model and pointer machine model. In general, there is a deterministic $O(m+n)$ time algorithm for computing the MSF of any graph (the extra $O(n)$ time is necessary for even reading over all of the vertices).*

The corollary above proves Theorem 1.1. We now prove Theorem 4.1.

Algorithm 2: $\mathsf{MSF}(G)$

```
Perform k Borůvka steps on G for a large enough constant k.
Let S be the set of contracted edges, and let G' be the resulting contracted graph.
Remove all isolated vertices from G'.
if G' = ∅ then
    return S.
Run Certificate(G') to obtain a forest F ⊆ E(G').
Compute the set U(F) of edges unsupported by F.
return S ∪ MSF(G'[F ∪ U(F)]).
```

Proof of Theorem 4.1. Consider a graph G with n vertices and m edges. We may assume without loss of generality that G has no isolated vertices, and hence $n \le 2m$. We also assume that all edge weights are distinct with ties broken by edge indices.

The algorithm is described in Algorithm 2. It first performs a constant number of Borůvka steps to reduce the number of vertices, then invokes the certificate procedure from Lemma 3.1 to obtain a forest F, and finally recurses on $F \cup U(F)$ where $U(F)$ is the set of edges unsupported by F.

We set $\varepsilon = 1/8$, and choose a sufficiently large constant integer k such that $C_{1/8} \cdot 2^{(1-k)/2} \le 3/8$, where $C_{1/8}$ is the constant from Lemma 3.1.

The algorithm begins by performing k Borůvka steps. This requires $O(km) = O(m)$ comparisons. Let S denote the set of edges contracted during these steps. The number of remaining non-isolated vertices is at most $n/2^k$. We then remove all isolated vertices from the contracted graph and let G' denote the resulting graph, with $n' \le n/2^k$ vertices. If G' is empty, the algorithm returns S and terminates. Otherwise, we apply the certificate procedure from Lemma 3.1 to G'.

The call $\mathsf{Certificate}(G')$ returns a forest F and makes recursive exact MSF calls on edge sets $E_1, \dots, E_\ell$. Let $U(F) \subseteq E(G')$ denote the set of edges unsupported by F. By Lemma 3.1, we have

$$|F| + |U(F)| + \sum_{i=1}^{\ell} |E_i| \le \frac{1}{8}m + C_{1/8}\sqrt{mn'}.$$

Moreover, given the forest F and $E(G')$, we can efficiently identify $U(F)$ using the algorithm from Lemma 2.9. This requires only $O(m)$ additional comparisons.

By the correctness of the Borůvka steps and the cycle property (Fact 2.3), we know that

$$\mathsf{MSF}(G) = S \cup \mathsf{MSF}(G'[F \cup U(F)]).$$

Thus, the algorithm recursively computes $\mathsf{MSF}(G'[F \cup U(F)])$ and combines the result with S to obtain $\mathsf{MSF}(G)$.

The total number of edges across all recursive MSF calls is bounded by

$$\begin{aligned}
|F \cup U(F)| + \sum_{i=1}^{\ell} |E_i| &\le \frac{m}{8} + C_{1/8}\sqrt{mn'} \\
&\le \frac{m}{8} + \frac{C_{1/8}}{2^{k/2}}\sqrt{mn} && (n' \le n/2^k) \\
&\le \frac{m}{8} + \frac{C_{1/8}}{2^{k/2}}\sqrt{2m^2} && (n \le 2m) \\
&\le \frac{m}{8} + \frac{3m}{8} && (C_{1/8} \cdot 2^{(1-k)/2} \le 3/8) \\
&= \frac{m}{2}.
\end{aligned}$$

Consequently, if we let $T(m)$ denote the maximum number of comparisons required to find a MSF in a graph with m edges, we obtain the recurrence

$$T(m) \le \sum_i T(m_i) + O(m), \qquad \text{where } \sum_i m_i \le m/2.$$

This yields $T(m) = O(m)$, and completes the proof. □

References

[BFP+72] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Linear time bounds for median computations. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing, May 1-3, 1972, Denver, Colorado, USA*, pages 119–124. ACM, 1972. 3, 16

[BGK+08] Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E. Tarjan, and Jeffery R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008. 4

[BKRW98] Adam L. Buchsbaum, Haim Kaplan, Anne Rogers, and Jeffery R. Westbrook. Linear-time pointer-machine algorithms for least common ancestors, MST verification, and dominators. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 279–288, 1998. 4

[Bor26a] Otakar Borůvka. O jistém problému minimálním. 1926. 1, 2, 12

[Bor26b] Otakar Borůvka. Příspěvek k řešení otázky ekonomické stavby elektrovodných sítí. *Elektrotechnický obzor*, 15(10):153–154, 1926. 1, 2

[Cha97] Bernard Chazelle. A faster deterministic algorithm for minimum spanning trees. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 22–31, 1997. 2, 3

[Cha00a] Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM (JACM)*, 47(6):1028–1047, 2000. 1, 2, 3, 10

[Cha00b] Bernard Chazelle. The soft heap: an approximate priority queue with optimal error rate. *Journal of the ACM*, 47(6):1012–1027, 2000. 3

[CT76] David Cheriton and Robert Endre Tarjan. Finding minimum spanning trees. *SIAM Journal on Computing*, 5(4):724–742, 1976. 2, 3, 4

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 2

[DRT92] Brandon Dixon, Monika Rauch, and Robert E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992. 1, 4, 12

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987. Preliminary version in FOCS 1984. 2, 3, 4

[FW94] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533–551, 1994. Preliminary version in FOCS 1990. 1, 2, 4

[GGS84] Harold N. Gabow, Zvi Galil, and Thomas H. Spencer. Efficient implementation of graph algorithms using contraction. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 347–357, 1984. 2, 3

[GGST86] Harold N. Gabow, Zvi Galil, Thomas Spencer, and Robert Endre Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986. 2

[GH85] Ronald L. Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985. 2

[GYY80] Ronald L Graham, Andrew C Yao, and F Frances Yao. Information bounds are weak in the shortest distance problem. *Journal of the ACM (JACM)*, 27(3):428–444, 1980. 10, 19

[Jar30] Vojtěch Jarník. O jistém problému minimálním. (Z dopisu panu O. Borůvkovi). *Práce Moravské přírodovědecké společnosti*, 6(4):57–63, 1930. 2

[Joh75] Donald B. Johnson. Priority queues with update and finding minimum spanning trees. *Information Processing Letters*, 4(3):53–57, 1975. 2

[Kar93] David R. Karger. Random sampling in matroids, with applications to graph connectivity and minimum spanning trees. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 84–93, 1993. 4

[Kin97] Valerie King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997. 4, 12

[KKT95] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995. 1, 2, 4

[Kom85] János Komlós. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985. 4, 6, 12

[Kru56] Joseph B. Kruskal, Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. 2

[KT94] Philip N. Klein and Robert E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 9–15, 1994. 1, 2, 4

[Mar04] Martin Mareš. Two linear time algorithms for MST on minor closed graph classes. *Archivum Mathematicum*, 40(3):315–320, 2004. 4

[NMN01] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar Borůvka on minimum spanning tree problem: Translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1–3):3–36, 2001. 2

[Pet99] Seth Pettie. Finding minimum spanning trees in $O(m\alpha(m,n))$ time. Technical Report TR99-23, Department of Computer Sciences, The University of Texas at Austin, October 1999. 3

[Pet06] Seth Pettie. An inverse-Ackermann type lower bound for online minimum spanning tree verification. *Combinatorica*, 26(2):207–230, 2006. 4, 10

[PR02] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM (JACM)*, 49(1):16–34, 2002. 1, 2, 4, 5, 10, 12, 25

[Pri57] Robert C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. 2

[Sch16] Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Transactions on Computation Theory (TOCT)*, 8(1):1–36, 2016. 11

[Tar75] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975. 3

[Tar79a] Robert Endre Tarjan. Applications of path compression on balanced trees. *Journal of the ACM*, 26(4):690–715, 1979. 4

[Tar79b] Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of computer and system sciences*, 18(2):110–127, 1979. 10, 12

[Tar83] Robert Endre Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1983. 2

[Yao75] Andrew Chi-Chih Yao. An $O(|E|\log\log|V|)$ algorithm for finding minimum spanning trees. *Information Processing Letters*, 4(1):21–23, 1975. 2, 3

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il