Minimum Spanning Tree in Deterministic Linear Time For Graphs of High Girth

Sarah Eisenstat[1], Dana Moshkovitz[1]*, Robert E. Tarjan[2], and Siyao Xu[1]

[1] Department of Electrical Engineering and Computer Science, MIT.
Emails: seisenst@mit.edu, dmoshkov@csail.mit.edu, siyao_xu@mit.edu.
[2] Department of Computer Science, Princeton University.
Email: ret@cs.princeton.edu.

Abstract. We show a simple deterministic linear time algorithm for computing the minimum spanning tree of graphs on n vertices with girth at least $b(t, \lg n)$, where $t \geq 1$ is a constant and $b(x, y)$ is a variant of the inverse of the fast-growing Ackermann function.
We also prove: (i) A deterministic linear time algorithm for general graphs follows from an algorithm for graphs with girth at least g for any constant $g \geq 1$; (ii) To find the minimum spanning tree of a graph G, it suffices to find the minimum spanning of any graph $\hat{G}$ obtained from G by removal of $O(n/b(t, \lg n))$ edges for any constant $t \geq 1$; (iii) Our high girth algorithm yields an alternative deterministic linear time algorithm for random graphs.

1 Introduction

Given a connected undirected graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, where the edges have real non-negative weights $w : E \to \mathbb{R}^+$, a *minimum spanning tree* (MST) of G is a connected sub-graph T on the n vertices that minimizes $\sum_{e \in T} w(e)$. For simplicity, we assume that the edge weights are distinct, in which case the MST is unique. Finding the minimum spanning tree of a given graph is a fundamental algorithmic problem with numerous applications, e.g., to design of networks, clustering, approximation algorithms for NP-hard problems, and more. The interested reader is referred to surveys such as [1, 7].

The most efficient deterministic algorithm currently known for the minimum spanning tree problem was found by Chazelle [4] and runs in $O(m\alpha(m, n))$ time, where α is the inverse Ackerman function. There is a randomized MST algorithm by Karger, Tarjan and Klein [8] that runs in expected time $O(m + n)$. Finding a deterministic linear time algorithm is an open problem. In order to solve it, one can concentrate only on sparse graphs:

Corollary 1 (Sparse graphs suffice). *If there exists a deterministic linear-time algorithm for computing the minimum spanning tree of a graph G with $m \leq \frac{3}{2}n$, then there exists a deterministic linear-time algorithm for computing the minimum spanning tree of any graph.*

* This material is based upon work supported by the National Science Foundation under Grant Number 1218547.

Fredman and Tarjan [5] have shown that there exists a deterministic linear time algorithm for graphs with $m \geq n \log^{(t)} n$ edges for any constant $t \geq 1$. Chazelle's algorithm [4] takes linear time for graphs with m and n satisfying the equation $A(t, \lceil m/n \rceil) \geq \lg n$, where $A(i, j)$ is the Ackermann function and t is any constant. But for graphs where m is $O(n)$, the problem remains open.

The *girth* of a graph is the length of its shortest cycle. Graphs of high girth are graphs that look like trees locally, even though they may have cycles. They arise in many real-life applications, and are the subject of much theoretical interest. In fact, almost all sparse graphs are of high girth after the removal of a small number of edges.

We observe that without loss of generality, we may assume that our graph does not have constant-sized cycles:

Lemma 1 (No constant-sized cycles). *For any constant $g \geq 1$, if there exists a deterministic linear-time algorithm for computing the minimum spanning tree of a graph G with girth $\geq g$, then there exists a deterministic linear-time algorithm for computing the MST of any graph.*

The main contribution of this work is a simple deterministic linear time algorithm for graphs whose girth is at least $b(t, \lg n)$, where $b(x, y)$ is a variant of the inverse Ackermann function, and t is any constant > 1.

Theorem 1 (High girth MST algorithm). *There is a deterministic linear time algorithm that, given a weighted graph G on n vertices whose girth is at least $b(t, \lg n)$ for some constant $t > 1$, finds a minimum spanning tree of G.*

Since there is a gap between the girth in Theorem 1 and the girth in Lemma 1, we do not get a deterministic linear time algorithm for general graphs this way.

We show that in order to obtain a linear time algorithm for the minimum spanning tree problem, it suffices to solve the problem on a graph that is obtained from the input graph by removal of a small number of edges:

Lemma 2 (Input approximation). *The minimum spanning tree of a graph G can be found in linear time given the minimum spanning tree of a graph $\hat{G}$ that is obtained from G by removal of at most $O(n/b(t, \lg n))$ edges where $t \geq 1$ is a constant.*

Using this lemma and the facts that: (i) a sparse random graph is close to having large girth, while (ii) there is a linear time algorithm for somewhat dense graphs, we get a deterministic linear time algorithm for computing the minimum spanning tree of a graph with randomly chosen edges and arbitrary edge weights:

Corollary 2 (Deterministic linear time algorithm for almost all graphs). *There is a deterministic linear time algorithm that satisfies the following: There exists $\delta = \delta(n) = 1/poly(n)$, such that for any $p = p(n) \in [0, 1]$, with probability at least $1 - \delta$ over $G \sim G(n, p)$, the algorithm computes the minimum spanning tree of G, regardless of the choice of edge weights. With the remaining probability, the algorithm declares that it is unable to compute the MST of G.*

In 1980, Karp and Tarjan [9] gave a deterministic $O(m)$ algorithm for computing the minimum spanning tree on graphs with randomly chosen edges and random edge weights. In 2002, Pettie and Ramachandran [11] gave a deteministic $O(m)$ algorithm for computing the minimum spanning tree on graphs with randomly chosen edges and arbitrary edge weights. Our algorithm provides an alternative to Pettie and Ramachandran's results.

Multi-edge blue and red rule. The *red rule* is that any edge that is heaviest on a cycle is not in the MST. The *blue rule* is that any edge that is lightest in a cut is an MST edge. We suggest and use throughout this work the following rules that determine the fate of many edges at once:

- The *multi-edge red rule* is that if G is a graph and H is a sub-graph of G, then edges not in the MST of H are not in the MST of G, i.e., $E(H) - \text{MST}(H) \subseteq E(G) - \text{MST}(G)$. The reason is that any edge not in the MST of H is heaviest in some cycle in H, and hence in G.
- The *multi-edge blue rule* is that if G is a graph and H is obtained from merging vertices of G, then edges in the MST of H are in the MST of G, i.e., $\text{MST}(H) \subseteq \text{MST}(G)$. The reason is that any edge in the MST of H is lightest in some cut of H, which corresponds to a cut in G.

Organization. We start with definitions. Then, we present a procedure required for our algorithm in Section 3. The procedure finds a linear-sized matching of MST edges in an appropriate modification of the input graph. We then describe the high girth algorithm in Section 4. The rest of the claims from the introduction are proven in Section 5.

2 Definitions

In this paper, every graph $G = (V(G), E(G))$ is assumed to be undirected. When not otherwise defined, $m = |E(G)|$ and $n = |V(G)|$. A graph G is *weighted* if it is accompanied by a weight function $w : E(G) \to \mathbb{R}^+$. For simplicity, we assume that all the edge weights are distinct. We overload set notation by defining $G - S = (V(G), E(G) - S)$ and $G \cup S = (V(G), E(G) \cup S)$.

Let G be a connected graph. A *spanning tree* of G is a connected subgraph that contains all vertices of G and has no cycles. Let $\text{ST}(G)$ be the set of all spanning trees of G. Then the minimum spanning tree $\text{MST}(G)$ is:

$$\text{MST}(G) = \operatorname*{arg\,min}_{T \in \text{ST}(G)} \left(\sum_{e \in T} w(e) \right).$$

The *Ackermann function* $A(i, j)$ is defined as follows:

$$A(i,j) = \begin{cases} j+1 & \text{if } i = 0 \\ A(i-1, 1) & \text{if } j = 0 \text{ and } i \neq 0 \\ A(i-1, A(i, j-1)) & \text{otherwise} \end{cases}$$

Using the Ackermann function, we can define two different types of inverse, similar to those defined in Goel et al. [6]:

$$\alpha(m,n) = \min\{i \geq 1 \mid A(i, \lceil m/n \rceil) > \lg n\}$$
$$b(x,y) = \min\{z \geq 0 \mid A(x,z) > y\}$$

Note the relationship between these two functions: if $m = n \cdot b(t, \lg n)$, then $\alpha(m,n) = t$. The *girth* of a graph G is the length of the shortest cycle in G. In this paper, we consider a graph to have *high girth* if it has girth $\geq b(t, \lg n)$ for any constant t.

When we *contract* an edge $(x,y) \in E(G)$, the result is a graph H with vertices x and y fused into a new vertex z. The notation $H = G \backslash S$ means that H is the result of contracting each edge $e \in S$. Contraction is very useful in computing minimum spanning trees — in particular[3], if $S \subseteq \mathrm{MST}(G)$, then $\mathrm{MST}(G) = S \cup \mathrm{MST}(G \backslash S)$.

3 Finding a Matching of MST Edges

In general, the minimum spanning tree of a graph might not contain a large matching. For instance, if the minimum spanning tree is a star, then it does not contain a matching of size larger than 1. However, as we show in this section, any graph G can be transformed into an equivalent graph $\hat{G}$ whose minimum spanning tree contains a matching M of size at least $n/3$

Our deterministic algorithm for computing $\hat{G}$ and M was adapted from a randomized parallel algorithm by Pettie and Ramachandran [12]. It relies on the following lemma:

Lemma 3 (from [12]). *Suppose that* $w(x,y) > w(y,z)$*, where* $(x,y), (y,z)$ *are edges in a weighted graph* G*. Let* $\hat{G}$ *be derived from* G *by replacing* (x,y) *with an edge* (x,z) *of the same weight. Then* $(x,y) \in \mathrm{MST}(G)$ *if and only if* $(x,z) \in \mathrm{MST}(\hat{G})$*.*

This lemma, which Pettie and Ramachandran call the "Endpoint relocation property," lets us take any pair of edges (x,y) and (y,z) and perform local modifications to construct the edge (x,z), without affecting the rest of the minimum spanning tree. Furthermore, if (y,z) is in the minimum spanning tree, then the newly created edge must also be in the minimum spanning tree.

To compute $\hat{G}$ and M, we begin by performing one round of Borůvka's algorithm, in which we find a smallest edge incident to each vertex. Because each such edge is a minimum-weight edge crossing the cut between the vertex and the rest of the graph, all of those edges are in a minimum spanning tree of the graph. Let F be a rooted forest of MST edges. We take advantage of the endpoint relocation property to connect pairs of siblings directly, without having to

[3] Although contracting an edge (x,y) changes the endpoints of the edges incident to x and y, we still consider them to be the same edge for the purposes of calculating the set of edges contained in the minimum spanning tree.

```
MST-Match($F$)
1   use depth-first search to root each tree in $F$ and calculate depths
2   set $M = \{\}$
3   set $D = \{\}$
4   set $U = \{v \in V \mid \text{depth}(v) > 0\}$
5   while there exists $x \in U$
6       pick a node $x \in U$ with maximum depth
7       set $y$ to the parent of $x$
8       if $y$ has another child $z \in U$
9           if $w(x, y) < w(y, z)$
10              set $D = D \cup \{(y, z)\}$
11          else
12              set $D = D \cup \{(x, y)\}$
13          set $M = M \cup \{(x, z)\}$
14          set $U = U - \{x, z\}$
15      else
16          set $D = D \cup \{(x, y)\}$
17          set $M = M \cup \{(x, y)\}$
18          set $U = U - \{x, y\}$
19  return $(M, D)$
```

Fig. 1: An algorithm for converting a forest $F \subseteq E(G)$ of edges that are part of the minimum spanning tree of some graph G into a matching of minimum spanning tree edges for a modified version of G. The set D contains edges to be removed from the original graph G; the set M contains edges to be added to G. The number of edges in the matching M is at least $n/3$, where n is the number of vertices in the forest. We use the set U to keep track of which vertices are still unused.

match them to their parent, and then remove them from the forest. If a node has no siblings, then it can be matched with its parent and removed from the forest. To maximize the number of pairs generated in this way, we begin with the deepest nodes and work our way up the tree. Once there are no more nodes to pair up, the algorithm MST-Match, formally defined in Figure 1, returns the set of matched pairs M and the set of deleted edges D.

Theorem 2. *Let G be a graph of n vertices, let F be a subset of the edges in $\text{MST}(G)$, and let $(M, D) = \text{MST-Match}(F)$. Then the pair (M, D) satisfies the following properties: (i) $|M| = |D| \geq n/3$; (ii) M is a matching, i.e., for any two pairs $(x_1, y_1), (x_2, y_2) \in M$, $\{x_1, y_1\} \cap \{x_2, y_2\} = \phi$; (iii) $\text{MST}(G) = D \cup \text{MST}(((G - D) \cup M) \setminus M)$.*

Proof. First, we wish to show that $|M| \geq n/3$. Let $T_1, \ldots, T_\ell$ be the set of trees in F and for each T_i, let k_i be the number of vertices in T_i. Then there are initially $k_i - 1$ vertices of T_i in the set U. Every iteration, if $x \in V(T_i)$ then $|V(T_i) \cap U|$ decreases by at most 2, and otherwise $|V(T_i) \cap U|$ does not change. Because $k_i \geq 2$, the number of iterations for tree T_i is at least $k_i/3$, and therefore the total number of iterations is at least $\sum k_i/3 = n/3$. Each iteration increases the size of M by 1, so $|M| \geq n/3$.

Next, we wish to show that all of the pairs in M are disjoint. To do so, we must characterize the set of nodes removed from U. When a pair of nodes (x, z) is removed from U in line 14, x has maximum depth, and z is its sibling, so neither node can be the parent of any node still in U. When a pair of nodes (x, y) is removed from U in line 18, x has maximum depth, y is the parent of x, and x has no other siblings in U. So neither x nor y is the parent of any node still in U after the pair has been removed.

When some pair (x, y) is added to M in line 17, x is guaranteed to belong to U, but y is not. However, because y is the parent of x, y cannot have been removed from U on any previous iteration. So if y is not in U, then y was never in U. Hence y must be a root, and at the time (x, y) is added to M, x must be the only child of y left in U. Once (x, y) is added to M, x will be removed from U, so any root y will occur at most once in M.

Each non-root vertex can only be added to a pair in M if, at the time of the addition on line 13 or line 17, it belongs to the set U. However, immediately after being added to M, each such vertex will be removed from U. So any non-root vertex will also occur at most once in M, ensuring that M is a matching.

Let $H = (G - D) \cup M$. For $i \in \{1, \ldots, |M|\}$, let m_i be the edge added to M in iteration i, and let d_i be the edge added to D in iteration i. By construction, either $m_i = d_i$, or we can relocate d_i to m_i using Lemma 3. Because each $d_i \in F \subseteq \mathrm{MST}(G)$, $m_i \in \mathrm{MST}(H)$. Hence, we have $\mathrm{MST}(G) = (\mathrm{MST}(H) - M) \cup D$. Because $M \subseteq \mathrm{MST}(H)$, this is equivalent to saying $\mathrm{MST}(G) = \mathrm{MST}(H \backslash M) \cup D$, which is precisely what we wanted. $\square$

Lemma 4. *The running time of* MST-MATCH(F) *is linear in the number of vertices in* F.

Proof. Depth-first search requires linear time, and can be used to compute node depths. The set U can be stored as a boolean associated with each vertex. Because no vertex is added back to U after it has been removed, we can implement the main loop by examining each vertex once, in decreasing order of depth.

To find a sibling $z \in U$ efficiently, each parent y should store an array of its children and a pointer to the first child contained in U, updated whenever the algorithm looks for a sibling. Although it may take linear time to find the next child in the worst case, the amortized time for each computation is $O(1)$. All other operations in the loop cost $O(1)$, so the total running time is $O(n)$. $\square$

Next we show that even if we contract the matching M in $\hat{G}$, the girth of the remaining graph is not much smaller than the girth of G:

Lemma 5. *Let G be a graph of girth $g \geq 8$, and let $F \subseteq G$ be an arbitrary forest. If $(M, D) =$* MST-MATCH(F)*, then the girth of $H = ((G - D) \cup M) \backslash M$ is $\geq g/4$.*

Proof. Suppose, to the contrary, that H contains a cycle C_A of length $< g/4$. We construct the corresponding cycle $C_B \in ((G - D) \cup M)$ by "un-contracting" the edges of M — splitting each vertex $z \in C_A$ that was the result of contracting $(x, y) \in M$. Because all of the edges in M are disjoint, $|C_B| \le 2|C_A| < g/2$.

Next, we define a multiset of edges S as follows. For each edge $(x, z) \in C_B$, if $(x, z) \in G$, add (x, z) to S. Otherwise, $(x, z) \in M$ and must have been added on line 13 of the algorithm in Figure 1. By examining this algorithm, we can see that there must exist some vertex y (the parent of x and z in the rooted forest) such that either $(x, y) \in D$ or $(y, z) \in D$. Furthermore, both (x, y) and (y, z) must be edges in the original graph G. In this case, we add both (x, y) and (z, y) to S. As a result, we have $|S| \le 2|C_B| < g$

We wish to show that there is at least one edge in S that is not duplicated. No two edges C_B are the same, so any duplicate edge in S must have been generated by some $(x_1, z_1) \in C_B - E(G)$. Let y_1 be the parent of x_1 and z_1 in the rooted forest, so that (x_1, y_1) and (y_1, z_1) are the edges added to S. For contradiction, suppose that both (x_1, y_1) and (y_1, z_1) occur twice. By construction, either $(x_1, y_1) \in D$ or $(y_1, z_1) \in D$. Without loss of generality, suppose that $(x_1, y_1) \in D$. Then $(x_1, y_1) \notin ((G - D) \cup M)$, and so $(x_1, y_1) \notin C_B$. So for (x_1, y_1) to be duplicated, the other copy must have been generated by some other pairing $(x_2, z_2) \in C_B - E(G)$. Let y_2 be the parent of x_2 and z_2 in the rooted forest. Because M is a matching, $\{x_1, z_1\} \cap \{x_2, z_2\} = \phi$. Hence, to ensure that (x_1, y_1) is duplicated, we must have $x_1 = y_2$ and either $x_2 = y_1$ or $z_2 = y_1$. Therefore, x_1 is the parent of both x_2 and z_2, and either x_2 or z_2 is the parent of x_1. This is clearly a contradiction, so it must be at most one of (x_1, y_1) and (y_1, z_1) occurs twice.

We have seen that if S contains any edges, then S must contain at least one non-duplicated edge. By construction, S is an Eulerian tour. So if we remove all duplicate edges of S, there must be at least one cycle $C_C \subset S$. Hence we have $|C_C| \le |S| \le 2|C_B| \le 4|C_A| < g$, which is a contradiction. □

4 Minimum Spanning Tree for High-Girth Graphs

In this section, we present an algorithm for computing the minimum spanning tree of a high-girth graph. Intuitively, this algorithm is a modification of Borůvka's algorithm [2, 3, 10], with a few key differences.

We begin by reducing the degree of the input graph to a constant d in a girth-preserving way. The specifics of this transformation are detailed in Lemma 7. Next, we associate with each vertex a Fibonacci heap containing all of the edges incident to that vertex, thus allowing us to efficiently calculate the minimum-weight incident edge. When a pair of vertices is contracted, the edge between them is removed from their heaps, and their heaps are merged. We use the MST-Match algorithm of Section 3 to ensure that all of the heaps are of roughly equal size. Finally, we only perform a limited number of iterations — determined by the girth of the graph — before switching to Chazelle's minimum spanning tree algorithm. This way we never have to remove more than one edge from heaps when we contract an edge. A more formal description of this algorithm may be found in Figure 2.

```
MST-HIGH-GIRTH($G, g$)
 1  create a table $H[\cdot]$
 2  for each vertex $v \in V(G)$
 3          set $H[v] =$ MAKE-HEAP($\{\}$)
 4  for each edge $e = (u, v) \in E(G)$
 5          HEAP-INSERT($H[u], e$)
 6          HEAP-INSERT($H[v], e$)
 7  set $G_0 = G$
 8  set $T = \{\}$
 9  set $\ell = \lfloor \log_4 g \rfloor - 2$
10  for $i$ from 1 to $\ell$
11          set $F = \{\}$
12          for each vertex $v \in V(G_{i-1})$
13                  set $e =$ HEAP-MIN($H[v]$)
14                  set $F = F \cup \{e\}$
15          set $(M, D) =$ MST-MATCH($F$)
16          set $T = T \cup D$
17          set $G_i = G_{i-1} - D$
18          for each edge $e = (x, y) \in D$
19                  HEAP-REMOVE($H[x], e$)
20                  HEAP-REMOVE($H[y], e$)
21          for each pair $(x, y) \in M$
22                  merge $x$ and $y$ in $G_i$ to create a vertex $z$
23                  update the endpoints for all edges originally incident to $x$ or $y$
24                  set $H[z] =$ HEAP-MERGE($H[x], H[y]$)
25  return $T \cup$ MST-CHAZELLE($G_\ell$)
```

Fig. 2: Pseudocode for a deterministic linear-time algorithm for computing the minimum spanning tree for input graphs of girth $g \geq b(t, \lg n)$.

Lemma 6. *Given a graph G with maximum degree d and girth $g \geq b(t, \lg n)$ the algorithm* MST-HIGH-GIRTH *computes the minimum spanning tree in time $O((t+1)(m+n) + n \lg d)$.*

Proof. First, we wish to bound the runtime. By Theorem 3, we need only bound the number of comparisons performed by the algorithm. By using techniques similar to the UNION-FIND data structure to implement the endpoint reassignment in line 23, it is possible bound the running time without using the results of Theorem 3, but we omit this explanation for the sake of brevity.

Creating an empty heap takes time $O(1)$; inserting into a Fibonacci heap takes time $O(1)$. Hence, the overall running time of lines 1 through 6 is $O(m + n)$. Let n_i be the number of vertices in G_i. During iteration i, we contract $|M| \geq n_{i-1}/3$ edges. Thus $n_i \leq (2/3) \cdot n_{i-1}$, and therefore $n_i \leq n \cdot (2/3)^i$. Let t_i be the maximum size of any heap at the beginning of iteration i. By definition, $t_1 = d$. Because M is a matching, each heap is merged with at most one other heap during iteration i. Hence, $t_{i+1} \leq 2t_i$, and therefore $t_i \leq d \cdot 2^{i-1}$. Thus each Heap-Remove operation takes time $O(\lg(d \cdot 2^{i-1})) = O(i + \lg d)$. We perform $2|M|$ such operations per iteration of the loop, for a total cost of $O(n_{i-1} \cdot (i + \lg d)) = O(n \cdot (i + \lg d) \cdot (2/3)^i)$. All other heap operations take time $O(1)$, and none are performed more than $O(n_{i-1})$ times per iteration, so the runtime of a single iteration is dominated by the Heap-Remove operation. The MST-Match algorithm also requires time $O(n_{i-1})$.

By summing these costs, we may compute the runtime of the main loop:

$$c \cdot \sum_{i=1}^{\ell} n \cdot (i + \lg d) \cdot \left(\tfrac{2}{3}\right)^i = cn \cdot \sum_{i=1}^{\ell} i \left(\tfrac{2}{3}\right)^i + cn \lg d \cdot \sum_{i=1}^{\ell} \left(\tfrac{2}{3}\right)^i = O(n \lg d)$$

Finally, we must compute the running time for MST-Chazelle(G_ℓ). The graph G_ℓ has n_ℓ vertices and at most m edges. Because $m \geq n$ we know that

$$\frac{m}{n_\ell} \geq \frac{n}{n\left(\frac{2}{3}\right)^\ell} = \left(\tfrac{3}{2}\right)^{\lfloor \log_4 g \rfloor - 2} > \lg g \geq \lg b(t, \lg n) \geq b(t+1, \lg n)$$

Hence the cost of running MST-Chazelle(G_ℓ) is $O((t+1)(m+n))$, and the overall runtime of MST-High-Girth is $O((t+1)(m+n) + n \lg d)$.

Next, we must examine correctness. To do so, we wish to show that each edge e computed on line 13 is in fact the lowest-weight edge leaving v. Every edge $e \in D$ that is removed from the graph on line 17 is also removed from both containing heaps on lines 19 and 20. No other edges are removed from the heaps. Furthermore, for every pair of vertices in M that are fused, the corresponding heaps are merged. Hence, as long as there are no self-loops in G_i, the minimum-weight edge in $H[v]$ is the same as the minimum-weight edge leaving v. Self-loops are generated by contracting parallel edges. A pair of parallel edges forms a cycle of length 2. By combining Lemma 5 with an inductive argument, we can show that the girth of G_i is at least $g/4^i \geq g/4^\ell \geq 4$, so there are no self-loops.

By construction, $G_i = ((G_{i-1} - D) \cup M)\backslash M$. Because F is equal to the set of Borůvka merges, we may use Theorem 2 to show that $\mathrm{MST}(G_{i-1}) = D \cup \mathrm{MST}(G_i)$. Hence, the result returned by MST-High-Girth is correct. □

To use the algorithm MST-High-Girth, we want to ensure that the maximum degree d is constant, without adversely affecting the girth of the graph. The following lemma shows how to achieve this:

Lemma 7 (Degree reduction). *Given a graph G we can find in linear time a graph H where: (i) All the vertices in H have degree ≤ 3; (ii) The size of H is linear in the size of G; (iii) The girth of H is as high as the girth of G; (iv) Given $\mathrm{MST}(H)$ one can compute $\mathrm{MST}(G)$ in linear time.*

Proof. We define H as follows:

1. For each vertex $u \in V(G)$, let $v_1, \ldots, v_{k_u}$ be the neighbors of u. For each neighbor v_i, construct a vertex x_{u,v_i}. For each $1 \leq i \leq k_u - 1$, construct an edge of weight 0 between the new vertices x_{u,v_i} and $x_{u,v_{i+1}}$.

2. For each edge $(u, v) \in E(G)$, construct an edge of weight $w(u, v)$ between the new vertices $x_{u,v}$ and $x_{v,u}$.

If we contract all edges added to H in step 1, the resulting graph is equal to G. All of the contracted edges have weight 0, so all of them will be included in MST(H). Hence, we can use MST(H) to compute MST(G) in linear time.

By construction, the maximum degree of H is 3. Because $x_{u,v_1}, \ldots, x_{u,v_{k_u}}$ are connected in a path, any cycle in H must involve at least one edge $(x_{u,v}, x_{v,u})$. So if there is a cycle of length $< g$ in H, we can contract all of the edges added in step 1 to get an even shorter cycle in G. □

Hence, if we run the algorithm MST-High-Girth on H, it will correctly compute MST(H) in time $O((t + 1)(m + n) + n \lg 3) = O(m + n)$. From there, it is trivial to compute MST(G). This completes the proof of Theorem 1.

5 Proofs of Claims From the Introduction

In this section we prove the lemmas and corollaries presented in the introduction. We use the following theorem derived by Pettie and Ramachandran [11]:

Theorem 3. *If there exists some algorithm of any runtime that deterministically computes the minimum spanning tree of a graph using $\mathcal{T}^*(m, n)$ edge-weight comparisons, then it is possible to construct a deterministic algorithm for computing the minimum spanning tree of a graph with overall runtime $O(\mathcal{T}^*(m, n))$.*

Hence, when constructing an algorithm, it is sufficient to examine the number of comparisons performed; all other computations do not affect the overall runtime.

Our first claim follows directly from Lemma 7, as a result of the relationship between m and n in graphs of constant degree:

Corollary 3 (Sparse graphs suffice, Corollary 1 in the introduction). *If there exists a deterministic linear-time algorithm for computing the minimum spanning tree of a graph G with $m \leq \frac{3}{2}n$, then there exists a deterministic linear-time algorithm for computing the minimum spanning tree of any graph.*

Next we show how to apply Theorem 3 to remove all cycles of constant size:

Lemma 8 (No constant-sized cycles, Lemma 1 in the introduction). *For any constant $g \geq 1$, if there exists a deterministic linear-time algorithm for computing the minimum spanning tree of a graph G with girth $\geq g$, then there exists a deterministic linear-time algorithm for computing the MST of any graph.*

Proof. For each cycle of length at most g, find the heaviest edge in the cycle and remove it. Then apply the algorithm for graphs of girth at least g. Since for each cycle of length at most g we perform g comparisons, but remove one non-MST edge from the graph, the overall number of comparisons is at most $g \cdot m = O(m)$. By the premise, the number of comparisons required for finding the MST after the removal of cycles of length g is at most $O(m + n)$. Hence, the total number of comparisons is linear, and the lemma follows from Theorem 3. □

Lemma 9 (Input approximation; Lemma 2 in the introduction). *The minimum spanning tree of a graph G can be found in linear time given the minimum spanning tree of a graph $\hat{G}$ that is obtained from G by removal of at most $O(n/b(t, \lg n))$ edges where $t \geq 1$ is a constant.*

Proof. Suppose that r edges were removed from G to obtain $\hat{G}$. Let us denote them by E_r. Let T be a minimum spanning tree of $\hat{G}$. We find a minimum spanning tree in $T \cup E_r$ as follows: We perform $O(b(t, \lg n))$ Borůvka merges. At the beginning of merge i, let n_i be the number of nodes left, and let m_i be the number of edges left. Let ℓ_i be the number of edges contracted during merge i. Each contraction decreases the number of nodes by at most 1, and decreases the number of edges by at least 1. So we have $n_{i+1} \geq n_i - \ell_i$ and $m_{i+1} \leq m_i - \ell_i$. We can combine these equations to see that $m_{i+1} \leq m_i - (n_i - n_{i+1})$. So the total time required for performing $q = \Theta(b(t, \lg n))$ Borůvka merges is:

$$\sum_{i=1}^{q} c \cdot m_i \leq c \cdot \sum_{i=1}^{q} \left(m_1 - \sum_{j=1}^{i-1} (n_j - n_{j+1}) \right) = c \cdot \sum_{i=1}^{q} (m_1 - (n_1 - n_i))$$
$$\leq c \cdot (m_1 - n_1) q + c \cdot \sum_{i=1}^{q} n_i$$

By construction, $n_1 = n$ and $m_1 = (n-1) + r$, so $m_1 - n_1 = r + 1$. The number of nodes n_i shrinks geometrically, so the running time for our Borůvka merges is $O(n + rq) = O(n + r \cdot b(t, \lg n))$. After $q = \Theta(b(t, \lg n))$ merges, the number of vertices in the graph is $O(n/b(t, \lg n))$, which allows us to apply the algorithm of Chazelle [4] for an additional runtime cost of $O(n)$. When the costs of all the steps are combined, we find that computing the minimum spanning tree of the original graph G takes time $O(m + n + r \cdot b(t, \lg n))$. $\square$

The next lemma and corollary show that almost all sparse graphs are close to having high girth:

Lemma 10. *Given a graph $G \sim G(n, p)$, let Z_k be the number of cycles of length k. If $pn \leq 100 \lg n$ and $k \leq \lg \lg n$, then $\Pr[Z_k > n/\lg^2 n] < O\big(n^{-(2-\epsilon)}\big)$.*

The proof of Lemma 10 involves computing the expectation and variance of Z_k, and then applying Chebyshev's inequality. For those interested, the details can be found in the appendix.

Using a union bound on the probability that some Z_k might exceed $n/\lg^2 n$, it is straightforward to show the following:

Corollary 4. *Given a graph $G \sim G(n, p)$ with arbitrary edge weights, where $pn \leq 100 \lg n$, the probability that G has more than $O(n/b(t, \lg n))$ cycles of length $\leq g = b(t, \lg n)$ for some integer $t \geq 2$ is at most $O\big(n^{-(2-\delta)}\big)$ for any constant $\delta > 0$.*

By combining Theorem 1 with Corollary 4, we can derive an alterate algorithm for random graphs:

Corollary 5 (Deterministic linear time algorithm for almost all graphs; Corollary 2 in the introduction). *There is a deterministic linear time algorithm that satisfies the following: There exists $\delta = \delta(n) = 1/poly(n)$, such that for any $p = p(n) \in [0,1]$, with probability at least $1 - \delta$ over $G \sim G(n,p)$, the algorithm computes the minimum spanning tree of G, no matter what are the weights on its edges. With the remaining probability, the algorithm declares that it is unable to compute the minimum spanning tree of G.*

Acknowledgments

We are grateful to David Karger for discussions and to Ronitt Rubinfeld for encouraging us to write up this paper.

References

1. C. F. Bazlamaçci and K. S. Hindi. Minimum-weight spanning tree algorithms a survey and empirical study. *Comput. Oper. Res.*, 28(8):767–785, July 2001.
2. O. Borůvka. O jistém problému minimálním (about a certain minimal problem). *Praáce mor. přírodověd. spol. v Brně III*, pages 37–58, 1926.
3. O. Borůvka. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí (contribution to the solution of a problem of economical construction of electrical networks). *Elektronický Obzor*, 15:153–154, 1926.
4. B. Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.
5. M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.
6. A. Goel, S. Khanna, D. Larkin, and R. Tarjan. Disjoint set union with randomized linking. *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1005–1017.
7. R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *IEEE Ann. Hist. Comput.*, 7(1):43–57, January 1985.
8. D. Karger, P. Klein, and R. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42:321–328, 1995.
9. Richard M Karp and Robert Endre Tarjan. Linear expected-time algorithms for connectivity problems. *Journal of Algorithms*, 1(4):374 – 393, 1980.
10. J. Nešetřil, E. Milková, and H. Nešetřilová. Otakar borůvka on minimum spanning tree problem: Translation of both the 1926 papers, comments, history. *Discrete Math.*, 233(1-3):3–36, April 2001.
11. S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49(1):16–34, 2002.
12. S. Pettie and V. Ramachandran. Randomized minimum spanning tree algorithms using exponentially fewer random bits. *ACM Transactions on Algorithms*, 4(1), 2008.

A Omitted Proofs

Proof (of Lemma 10). Let C_k be the set of potential cycles of length k. For each potential cycle $c \in C_k$, let Y_c be an indicator random variable that is 1 if and only all the edges in c occur in G. Then we can write $Z_k = \sum_{c \in C_k} Y_c$. The expected value of Z_k is:

$$\mathrm{E}[Z_k] = \sum_{c \in C_k} \mathrm{E}[Y_c] = \sum_{c \in C_k} p^k = \binom{n}{k} \cdot \frac{(k-1)!}{2} \cdot p^k \leq (pn)^k$$

Because $pn \leq 100 \lg n$ and $k \leq \lg \lg n$, we may show that $\mathrm{E}[Z_k] = O(n^\delta)$ for any constant $\delta > 0$. The variance of Z_k is:

$$\mathrm{Var}[Z_k] = \mathrm{E}[(Z_k)^2] - (\mathrm{E}[Z_k])^2 = \sum_{c_1 \in C_k} \sum_{c_2 \in C_k} (\mathrm{E}[Y_{c_1} Y_{c_2}] - \mathrm{E}[Y_{c_1}] \cdot \mathrm{E}[Y_{c_2}])$$

For each pair of cycles $c_1, c_2 \in C_k$, the number of shared edges can range between 0 and k. If the number of shared edges is 0, then Y_{c_1} and Y_{c_2} are independent variables, so $\mathrm{E}[Y_{c_1} Y_{c_2}] - \mathrm{E}[Y_{c_1}]\mathrm{E}[Y_{c_2}] = 0$. If the number of shared edges is k, then $c_1 = c_2$, so $\mathrm{E}[Y_{c_1} Y_{c_2}] - \mathrm{E}[Y_{c_1}] \cdot \mathrm{E}[Y_{c_2}] = p^k - p^{2k} \leq p^k$.

Otherwise, let $0 < j < k$ be the number of shared edges. There are $\binom{k}{j} \leq (k!)$ possible ways to pick those j edges from the edges in c_1, which fixes at least $j+1$ of the vertices in c_2. So the number of cycles c_2 sharing j edges with c_1 is at most $n^{k-(j+1)} \cdot (k!)$. Hence, we may bound the variance by:

$$\begin{aligned}
\mathrm{Var}[Z_k] &\leq |C_k| \cdot \left(p^k + \sum_{j=1}^{k-1} n^{k-(j+1)} \cdot (k!) \cdot p^{2k-j} \right) \\
&\leq |C_k| \cdot p^k \cdot \left(1 + \sum_{j=1}^{k-1} \frac{1}{n} \cdot (pn)^{k-j} \cdot (k!) \right) \\
&\leq (pn)^k \cdot \left(1 + \frac{k!}{n} \sum_{j=1}^{k-1} (pn)^{k-j} \right).
\end{aligned}$$

Because $pn \leq 100 \lg n$ and $k \leq \lg \lg n$, the variance is also $O(n^\delta)$ for any $\delta > 0$. Hence we may use Chebyshev's inequality to show that

$$\begin{aligned}
\Pr[Z_k > n/\lg^2 n] &= \Pr[Z_k - \mathrm{E}[Z_k] > n/\lg^2 n - \mathrm{E}[Z_k]] \\
&\leq \frac{\mathrm{Var}[Z_k]}{\left(n/\lg^2 n - \mathrm{E}[Z_k]\right)^2} \leq O\left(\frac{1}{n^{2-\epsilon}}\right)
\end{aligned}$$

for any $\epsilon > 0$. □

Proof (of Corollary 4). For each $k \leq g < \lg \lg n$, let Z_k be the number of cycles in G of length k. The total number of cycles of length $\leq g$ is equal to $\sum Z_k$.

If the total number of short cycles is more than $n/b(t, \lg n)$, then at least one of $Z_3, \ldots, Z_g$ must be greater than $n/((g-3) \cdot b(t, \lg n)) > n/g^2 > n/\lg^2 n$. Hence, we can use a union bound on the probability of Lemma 10 to show that $\Pr[\#\text{ of short cycles} > n/b(t, \lg n)] \leq (g-3) \cdot O(n^{-(2-\epsilon)}) \leq O(n^{-(2-\delta)})$ for any constant $\delta > 0$. □

Proof (of Corollary 5). We present an algorithm that makes a linear number of comparisons. The corollary then follows from Theorem 3.

If the graph is dense in the sense that $m \geq n \log^{(t)} n$, we use the algorithm of Fredman and Tarjan [5]. In the remainder of the proof we assume that $p \leq (100 \lg n)/n$. For larger p the graph is sufficiently dense except with exponentially low probability (and in this low probability event, the algorithm can declare so). We also assume that the number of cycles in G with length less than $g = b(t, \lg n)$ for $t \geq 2$ is $r = O(n/b(t, \lg n))$. By Corollary 4, this is the case except with probability $1/poly(n)$. Moreover, in the low probability event that this is not so, the algorithm can detect that.

We begin by finding all cycles of length $\leq g$. For each of the r short cycles, pick an arbitrary edge on the cycle. Let S be the union of those edges. We construct a graph $G_{high} = (V, E - S)$ by removing all edges in S from G. Because S contains at least one edge from each short cycle in G, the girth of G_{high} must be at least $g = b(t, \lg n)$. Therefore, we may use the high-girth algorithm from Theorem 1 to compute the MST of G_{high} in $O(m + n)$ time. Next we apply Lemma 2 to compute the minimum spanning tree of G. □