Sublinear-Time Algorithm for MST-Weight Revisited

Gryphon Patlin* Jan van den Brand†

Abstract

For graphs of average degree d, positive integer weights bounded by W, and accuracy parameter $\epsilon > 0$, [Chazelle, Rubinfeld, Trevisan; SICOMP'05] have shown that the weight of the minimum spanning tree can be $(1+\epsilon)$-approximated in $\tilde{O}(Wd/\epsilon^2)$ expected time. This algorithm is frequently taught in courses on sublinear time algorithms. However, the $\tilde{O}(Wd/\epsilon^2)$-time variant requires an involved analysis, leading to simpler but much slower variations being taught instead.

Here we present an alternative that is not only simpler to analyze, but also improves the number of queries, getting closer to the nearly-matching information theoretic lower bound. In addition to estimating the weight of the MST, our algorithm is also a perfect sampler for sampling uniformly at random an edge of the MST. At the core of our result is the insight that halting Prim's algorithm after an expected $\tilde{O}(d)$ number of steps, then returning the highest weighted edge of the tree, results in sampling an edge of the MST uniformly at random. Via repeated trials and averaging the results, this immediately implies an algorithm for estimating the weight of the MST. Since our algorithm is based on Prim's, it naturally works for non-integer weighted graphs as well.

1 Introduction

As data sets grow at an unprecedented rate, traditional algorithms that operate in linear or polynomial time can become infeasible. Sublinear time algorithms, which construct approximate solutions based on only a small portion of the input, offer a solution in this setting. By reducing the dependency on input size, these algorithms address practical limitations in both computational resources and time constraints. Techniques developed for sublinear time algorithms also find application in other modern computational problems such as distributed [16], dynamic [4, 5], streaming algorithms [11] and learning [13].

In this work, we focus on the minimum spanning tree (MST) problem. Computing the MST was shown to be impossible in sublinear time (i.e., $o(n^2)$ on dense graphs) even when allowing the MST to be a constant factor approximation [15]. However, [6] showed that one can estimate the weight of the MST up to $(1+\epsilon)$ in $\tilde{O}(Wd/\epsilon^2)$ expected time[1] for graphs with average degree d and positive integer weights bounded by W. Note that this time complexity has no dependence on the number of vertices n and is just constant time for sparse graphs with small edge weights. [6] also proved a lower bound of $\Omega(Wd/\epsilon^2)$ time, matching their upper bound up to polylog factors. Later work on estimating the weight of the MST studied other settings, such as complete graphs where the weights form a metric [10] or euclidean distances [8]. Applications include, for example, $(2+\epsilon)$-approximating the length of a (metric) TSP tour in sublinear time.

Since computing the MST is a well established and common problem, algorithms for estimating the MST are a natural choice to teach in courses on sublinear time algorithms. However, since the analysis of [6] is somewhat involved, for educational purposes simplified but slower versions of the algorithm are commonly taught instead [9, 19, 20, 17, 18, 7, 22, 12, 1, 2, 14, 3]. These simplified algorithms capture the main ideas of [6] but their time complexities do not match the lower bound of $\Omega(Wd/\epsilon^2)$. Depending on the accuracy of the error analysis, the simplified versions only achieve an expected time ranging between $\tilde{O}(W^3\epsilon^{-2}d_{\max})$ and $O(\text{poly}(W/\epsilon)d_{\max})$, where $d_{\max}$ is the maximum degree in the graph. In addition to the worse dependence on W and ϵ, note that the maximum degree $d_{\max}$ can generally be substantially larger than the average degree d, even in sparse graphs.

In this work, we show that a simple modification of Prim's algorithm achieves the same $\tilde{O}(Wd/\epsilon^2)$ time complexity as [6] for estimating the weight of the MST. Our algorithm also improves the log factors of the sampling complexity, bringing the complexity closer to the lower bound of $\Omega(Wd/\epsilon^2)$. The techniques further imply a perfect sampler for sampling an edge of the MST uniformly at random in just $\tilde{O}(d)$ expected time.

*Georgia Institute of Technology.

†Georgia Institute of Technology. Jan van den Brand was supported by NSF Award CCF-2338816.

[1] We use $\tilde{O}(\cdot)$ to hide polylog factors.

1.1 Our Results Throughout, d is the average vertex degree, where the degree $\deg(v)$ of a vertex is the number of edges adjacent to it, n the number of vertices, $W = W_{\max}/W_{\min}$ the ratio of largest to smallest edge weight. Given a set of vertices $U \subseteq V$, $\mathrm{vol}(U)$ represents the sum of the degrees of the vertices in U. Like [6], we assume that the algorithm receives n and W as input, but it does not need to know d. We consider the standard model for sublinear time graph algorithms, where query access to the adjacency list is given. Querying the next entry in the adjacency list of any vertex is assumed to take $O(1)$ time, i.e., reading the entire adjacency list of a vertex v takes $O(\deg(v))$ time.

THEOREM 1.1. *There is a randomized algorithm that computes a $(1+\epsilon)$ approximation of the MST weight using $O(Wd\epsilon^{-2}\log(W/\epsilon))$ queries in expectation and runs in $O(Wd\epsilon^{-2}\log(W/\epsilon)\log\log\frac{\log W}{\epsilon})$ expected time. The algorithm's output is correct with probability $3/4$.*

The previous bound by [6] is $O(Wd\epsilon^{-2}\log(Wd/\epsilon))$ expected time and number of queries, when the graph has integer weights. Their result was extended to real weights by discretizing the weights into multiples of $\Theta(W/\epsilon)$, resulting in a $O(Wd\epsilon^{-3}\log(Wd/\epsilon))$ bound for non-integer weights, i.e., a super-cubic dependence on ϵ. Closing this gap for integral and real weights was explicitly asked as open question in [6]. Our algorithm answers this question for the sampling complexity, and for time complexity it is answered up to a $\log\log\epsilon^{-1}$ factor.

The lower bound is $\Omega(Wd/\epsilon^2)$ and it is open if the logarithmic gap between upper and lower bound can be closed [21]. Our result makes progress towards answering this question by showing that the logarithmic dependence on d is not required.

The same techniques used to obtain Theorem 1.1 also imply the following perfect sampler for sampling edges of the MST uniformly at random in $\tilde{O}(d)$ time.

THEOREM 1.2. *In $O(d\log^2 n)$ expected time we can sample edge weights of the MST. Here any weight w is returned with probability $\frac{\#\text{MST edges of weight } w}{n-1}$. If the MST is unique then we can sample an edge of the MST uniformly at random.*

Note that the number of edges of weight w in the MST is unique, even if the MST itself is not unique. Further, we can always assume that the MST is unique by tie-breaking the edges (e.g., edges of the same weight could be ordered alphabetically by their end points).

Both Theorems 1.1 and 1.2 are obtained from a simple modification of Prim's algorithm. We show that running Prim's algorithm from a random start vertex, then halting the execution after a random number of steps, results in a uniform sample of an MST edge. The pseudo-code is given in Algorithm 1, and formally we prove the following guarantee:

LEMMA 1.1. *For $T > 2|E|$, Algorithm 1 (*$\mathrm{SampleMSTedge}_T(G)$*) returns weight w with probability*

$$\mathbb{P}[\mathrm{SampleMSTedge}_T(G) = w] = \frac{\#\textit{MST edges of weight } w}{n}$$

and returns 0 with probability $1/n$. In particular, $\mathbb{E}[\mathrm{SampleMSTedge}_T(G)] = \textit{MST-weight}/n$.

Observe that $\mathbb{E}[n \cdot \mathrm{SampleMSTedge}_T(G)]$ for $T = n^2 > 2|E|$ is the weight of an MST. Thus by repeating trials and averaging the results, we can estimate the weight of the MST via simple application of Chebyshev's inequality. (We show that a smaller $T = O(W/\epsilon)$ suffices resulting in Theorem 1.1.) Further, Theorem 1.2 is implied by repeatedly calling $\mathrm{SampleMSTedge}_T(G)$ (for $T = n^2$) until it returns a non-zero value.

Lemma 1.1 also works for disconnected graphs, in which case it returns weight w with probability (#MSF (minimum spanning *forest*) edges of weight w)$/n$. Then the probability of returning 0 is k/n, where k is the number of connected components of G. (See Section 2.1 for details.)

Lastly, we would like to mention that the condition of Line 4 in Algorithm 1 is just a single line modification of Prim's algorithm (see Algorithm 3 in the appendix). Prim's algorithm uses a priority queue to queue edges ordered by edge weight. Line 4 just means to halt Prim's algorithm once a total of $X \cdot \deg(v_{\text{start}})$ edges have been added to the queue. Prim's algorithm is one of the standard algorithm for computing the MST and it's an interesting insight how this small modification to Prim's algorithm makes it suitable for the sublinear-time regime.

Algorithm 1 SampleMSTedge$_T(G)$ (Full pseudo code, including Prim's Algorithm, see Algorithm 3)

Choose $X \geq 1$ according to $\mathbf{Pr}[X \geq k] = 1/k$ (i.e. $X \leftarrow \frac{1}{Y}$ for Y uniform from $(0, 1]$)
if $X \geq T$, halt and return 0.
Run Prim's algorithm from a uniformly at random picked $v_{\text{start}} \in V$.
Halt Prim early when $\text{vol}(U) > X \cdot \deg(v_{\text{start}})$ where $U \subseteq V$ are the vertices explored so far.
(Thus Prim visits a total of $X \cdot \deg(v_{start})$ edges, double counting edges when visiting them again from their other endpoint.)
return highest edge weight added to the MST so far, or 0 if Prim was not halted early.

Algorithm 2 Estimate MST weight(G, W, ϵ)

total $\leftarrow 0$
for i in $0...32W/\epsilon^2$ **do**
    total $\leftarrow$ total + SampleMSTedge$_T(G)$
end for
return $n \cdot \text{total}/(32W/\epsilon^2)$

1.2 Comparison to Previous Work Since our claim is to be simpler than previous work, we here highlight the differences between our work, the MST algorithm by [6], and previous simplifications as taught in graduate courses [19, 20, 17, 18, 7, 22, 12, 1, 2, 14, 3]. The time complexity of [6] is $\tilde{O}(W/\epsilon^2 d)$, but for educational purposes simpler variants are taught with a worse complexity (e.g., $\tilde{O}(\text{poly}(W/\epsilon)d_{\max})$). We here outline why these simplifications are slower, and explain why our algorithm does not share this issue.

Readers only interested in our algorithm can skip ahead to Section 2, as none of the following information is needed to understand our algorithm. The following information is only for comparison of simplicity and complexity.

Dependency on W and ϵ: Assume G is an integer weighted graph with maximum edge weight W. The idea of [6] is to reduce estimation of the MST-weight to estimating the number of connected components. Let G_i be the graph G containing only edges of weight $\leq i$, and let c_i be the number of connected components in G_i, then [6] showed

$$\text{MST-weight} = n - W + \sum_{i=1}^{W-1} c_i$$

By constructing a sublinear-time algorithm for estimating the number of connected components c_i, [6] could also estimate the MST-weight. Since W many estimators $\tilde{c}_i \approx c_i$ are computed and their approximation errors will add up, the accuracy of $\tilde{c}_i$ must depend on W. Depending on how precise/involved the error analysis is, a different accuracy requirement on $\tilde{c}_i$ is proven. This is why previous simplifications resulted in a worse complexity dependence on W.

Our algorithm does not encounter this issue, because we just run Prim's algorithm (Algorithm 1) on G instead of summing estimates computed from each G_i. In particular, Algorithm 1 returns MST-weight/n in expectation by Lemma 1.1, and the variance is

$$\text{Var}[\text{SampleMSTedge}_{n^2}(G)] \leq \sum_{w=1}^{W} w^2 \cdot \frac{\#\text{MST edges of weight } w}{n} \leq W \cdot \frac{\text{MST-weight}}{n}.$$

Direct application of Chebyshev-bound then shows we only need to average $O(W\epsilon^{-2})$ calls to Algorithm 1 to $(1+\epsilon)$-approximate the MST-weight.

Dependency d vs $d_{\max}$: The complexity of [6] depends on the average degree d, but the algorithm is commonly taught with dependence on the maximum degree $d_{\max}$ to simplify its analysis. Bounding the complexity wrt. d is tricky because the degrees differ between G and G_i.

[6] shows that one can estimate the number of connected components of graph G in $\tilde{O}(d\epsilon^{-2})$ time. However, this algorithm assumes reading the adjacency list of any vertex v takes $O(\deg(v))$ time. This is true for the input graph G but not for the graphs G_i because the algorithm must scan the adjacency list of G for the edges in G_i

(i.e., the time is proportional to $\deg(v)$ in G which could be much larger than the degree in G_i). Thus showing that estimating the number of connected components of G_i still takes $\tilde{O}(d)$ time, requires a more careful analysis. Further, the algorithm requires a preprocessing step to compute a threshold d^* (roughly equal to the average degree), and then all vertices of degree larger than d^* are skipped to prevent spending too much time on high degree vertices. To simplify the analysis, it is easier to just assume the adjacency list of any vertex is accessed in $O(d_{\max})$ time.

We do not encounter this issue of differing degrees between G and G_i since Algorithm 1 runs directly on G and not any G_i. To sketch our complexity analysis, remember that the complexity of Prim's algorithm is $O(k \log n)$ where k is the number of edges encountered by Prim's, and $O(\log n)$ is the complexity of the priority queue used by Prim's. Algorithm 1 visits $\min(X, T) \cdot \deg(v_{\text{start}})$ edges, which is $O(d \log(T))$ in expectation, as $\mathbb{E}[\deg(v_{\text{start}})] = d$ and $\mathbb{E}[\min(X, T)] = \int_1^T 1/x \; dx = O(\log T)$. Together with the previously stated bound of running Algorithm 1 for $O(W\epsilon^{-2})$ times, this then implies estimating the MST-weight in $\tilde{O}(Wd\epsilon^{-2})$ expected time.

2 Estimating MST-weight in Sublinear Time

Here we prove our main results Theorems 1.1 and 1.2. We first prove in Section 2.1 that Algorithm 1 samples an MST edge as described in Lemma 1.1 which is then used to prove Theorem 1.2. Then we use Lemma 1.1 in Section 2.2 to prove Theorem 1.1.

2.1 Sampling MST-edges in Sublinear Time

Throughout this subsection we assume that parameter T of SampleMSTedge_T (Algorithm 1) satisfies $T > 2|E|$, e.g., by letting $T = n^2$. This is equivalent to halting Prim's algorithm when $\text{vol}(U) > X \cdot \deg(v_{\text{start}})$ in Line 4 as there cannot be more than n^2 edges in the graph.

LEMMA 1.1. *For $T > 2|E|$, Algorithm 1 ($\text{SampleMSTedge}_T(G)$) returns weight w with probability*

$$\mathbb{P}[\text{SampleMSTedge}_T(G) = w] = \frac{\#\textit{MST edges of weight } w}{n}$$

and returns 0 with probability $1/n$. In particular, $\mathbb{E}[\text{SampleMSTedge}_T(G)] = \textit{MST-weight}/n$.

We will here prove a slightly stronger variant of this lemma, which states that it samples edges of the minimum spanning *forest* with

$$\mathbb{P}[\text{SampleMSTedge}_T(G) = w] = \frac{\#\text{MSF edges of weight } w}{n}$$

and returning 0 with probability k/n where k is the number of connected components in G. Here we assume that Prim's algorithm terminates naturally on its own (i.e., SampleMSTedge return 0) when it explored the entire connected component that contains the start vertex, and it only produces an MST of that connected component.

Proof. For any $w > 0$, let G_w contain all edges of G with weight $\leq w$, and let G_w^- contain all edges of weight $< w$. Let $\mathcal{U}_w$ and $\mathcal{U}_w^-$ be the set of all connected components in G_w and G_w^- respectively. Here we let each component $C \in \mathcal{U}_w$ be represented by a set of vertices, so $C \subseteq V$.

For $u \in V$ let $C_{w,u}^- \in \mathcal{U}_w^-$ and $C_{w,u} \in \mathcal{U}_w$ be the connected components containing u in each of their respective domains. When Algorithm 1 starts from u, then it returns weight w if and only if all of the following 3 conditions hold: (i) Prim was halted early (the algorithm did not return 0), (ii) Prim managed to explore outside of $C_{w,u}^-$ (i.e., the constructed MST contains an edge of weight $\geq w$) and (iii) Prim did not yet explore outside of $C_{w,u}$ (the constructed MST does not contain an edge of weight $> w$).

Since Prim halts early when $\text{vol}(U) > X \cdot d_u$ where U are the vertices it explored, we have by (ii) $\text{vol}(C_{w,u}^-) \leq X \cdot d_u$, and by (iii) $\text{vol}(C_{w,u}) > X \cdot d_u$. Thus we get the following probability

$$\mathbb{P}[\text{SampleMSTedge} = w | v_{\text{start}} = u] = \mathbb{P}[\text{vol}(C_{w,u}^-) \leq X \cdot d_u < \text{vol}(C_{w,u})]$$

Given the distribution of X, this translates to:

$$\begin{aligned}\mathbb{P}[\text{SampleMSTedge} = w | v_{\text{start}} = u] &= \mathbb{P}\left[\frac{\text{vol}(C^-_{w,u})}{d_u} \le X < \frac{\text{vol}(C_{w,u})}{d_u}\right] \\ &= \mathbb{P}\left[X \ge \frac{\text{vol}(C^-_{w,u})}{d_u}\right] - \mathbb{P}\left[X \ge \frac{\text{vol}(C_{w,u})}{d_u}\right] \\ &= \frac{d_u}{\text{vol}(C^-_{w,u})} - \frac{d_u}{\text{vol}(C_{w,u})} \end{aligned} \tag{2.1}$$

Since the start vertex is picked uniformly at random, we can write

$$\begin{aligned}\mathbb{P}[\text{SampleMSTedge} = w] &= \frac{1}{n}\sum_{u \in V} \mathbb{P}[\text{SampleMSTedge} = w | v_{\text{start}} = u] \\ \text{(by (2.1))} \quad &= \frac{1}{n}\sum_{u \in V}\left(\frac{d_u}{\text{vol}(C^-_{w,u})} - \frac{d_u}{\text{vol}(C_{w,u})}\right) \\ \left(\text{by } \dot{\bigcup_{C^- \in \mathcal{U}^-_w}} C^- = V \text{ and } \dot{\bigcup_{C \in \mathcal{U}_w}} C = V\right) \quad &= \frac{1}{n}\left(\left(\sum_{C^- \in \mathcal{U}^-_w}\sum_{u \in C^-}\frac{d_u}{\text{vol}(C^-)}\right) - \left(\sum_{C \in \mathcal{U}_w}\sum_{u \in C}\frac{d_u}{\text{vol}(C)}\right)\right) \\ \text{(by vol}(\cdot)\text{ being the sum of degrees)} \quad &= \frac{1}{n}\left(\left(\sum_{C^- \in \mathcal{U}^-_w}\frac{\text{vol}(C^-)}{\text{vol}(C^-)}\right) - \left(\sum_{C \in \mathcal{U}_w}\frac{\text{vol}(C)}{\text{vol}(C)}\right)\right) \\ &= \frac{1}{n}(|\mathcal{U}^-_w| - |\mathcal{U}_w|) \\ &= \frac{\#\text{MSF edges of weight } w}{n}\end{aligned}$$

The last equality holds, because $|\mathcal{U}^-_w|$ is the number of components using edges of weight $< w$, and $|\mathcal{U}_w|$ is the number of components using edges of weight $\le w$. So the difference $|\mathcal{U}^-_w| - |\mathcal{U}_w|$ is the number of components in $\mathcal{U}^-_w$ that the MSF connects via edges of weight w.

At last, we observe that the probabilities of all edge weights add up $(n-k)/n < 1$ where k is the number of connected components of G. If the algorithm does not return any edge weight, then it returns 0. This happens with probability k/n. ∎

LEMMA 2.1. *For $T \ge 1$, $\text{SampleMSTedge}_T(G)$ visits $O(d \log T)$ edges in expectation and runs in $O(dQ \log T)$ expected time where Q is the insert/pop time complexity of the priority queue used by Prim's algorithm.*

Proof. The number of edges visited by $\text{SampleMSTedge}_T(G)$ (Algorithm 1) is $\min(X, T) \cdot \deg(v_{\text{start}})$ (where we count edges double if they are visited again from the other end point). Here $\mathbb{E}[\deg(v_{\text{start}})] = d$, as the start vertex is sampled uniformly, and $\mathbb{E}[\min(X,T)] = \int_1^T \mathbb{P}[X \ge x]\, dx = \int_1^T 1/x\, dx = O(\log T)$. Since X and v_{start} are sampled independently, we have $\mathbb{E}[\min(X,T) \cdot \deg(v_{\text{start}})] = O(d \log T)$. Hence the time complexity is $O(dQ \log T)$ as Prim's algorithm inserts/pops each visited edge to/from the priority queue. ∎

A typical comparison-based min-heap takes at most $Q = O(\log n)$ time per insert/pop, since we add at most $O(n^2)$ edges to the queue. This then leads to the following Theorem 1.2. For our MST-weight algorithm (Theorem 1.1) we will use a different kind of queue with $Q = O(\log\log \frac{\log W}{\epsilon})$, see next subsection.

THEOREM 1.2. *In $O(d \log^2 n)$ expected time we can sample edge weights of the MST. Here any weight w is returned with probability $\frac{\#MST\ edges\ of\ weight\ w}{n-1}$. If the MST is unique then we can sample an edge of the MST uniformly at random.*

Proof. By Lemma 1.1, we can assume Algorithm 1 (for $T = n^2$) samples w with probability $\frac{\#\text{MST edges of weight } w}{n-1}$ by restarting whenever it returns 0. Returning 0 happens with probability $1/n$ and happens if and only if

Prim visits the entire graph. Thus the time complexity of Algorithm 1, conditioned on it returning 0, is $O(|E| \log n) = O(dn \log n)$. With Lemma 2.1, this implies the expected time (until an edge weight is returned) is

$$O(d \log^2 n) + \sum_{k \geq 1} \mathbb{P}[\text{restart } k \text{ times}] \cdot \mathbb{E}[\text{time of Algorithm 1}|\text{return 0}] = O(d \log^2 n) + \sum_{k \geq 1} \frac{1}{n^{k-1}} O(d \log(n))$$

which is $O(d \log^2 n)$. By also returning the edge that used the highest edge weight, the algorithm samples edges of the MST uniformly at random. ∎

2.2 Estimating the MST Weight By repeatedly calling Algorithm 1, taking the average, and multiplying by n, we obtain an estimate of the MST. For $T = n^2$, we would know by Lemma 1.1 that this would be correct in expectation. However, we use $T = W/\epsilon$ to reduce the worst-case number of visited edges and make our time complexity independent of n. The following Lemma 2.2 shows that the expectation is not too far off for this T.

LEMMA 2.2. *For $T = 4W/\epsilon$, we have $(1 - \epsilon/2)MST/n \leq \mathbb{E}[\text{SampleMSTedge}_T(G)] \leq MST/n$, and $\text{Var}(\text{SampleMSTedge}_T(G)) \leq \text{MST} \cdot W_{\max}/n$.*

Proof. Observe that both $\text{SampleMSTedge}_{(n^2)}(G)$ and $\text{SampleMSTedge}_T(G)$ (Algorithm 1) return the same result when conditioned on $X < T$, while the latter returns 0 when conditioning on $X \geq T$. Thus

$$\begin{aligned}
\mathbb{E}[\text{SampleMSTedge}_T] &= \mathbb{E}[\text{SampleMSTedge}_T \,|X < T] \cdot \mathbb{P}[X < T] + \underbrace{\mathbb{E}[\text{SampleMSTedge}_T \,|X \geq T]}_{=0} \cdot \mathbb{P}[X \geq T] \\
&= \mathbb{E}[\text{SampleMSTedge}_{(n^2)} \,|X < T] \cdot \mathbb{P}[X < T] \\
&= \mathbb{E}[\text{SampleMSTedge}_{(n^2)}] - \mathbb{E}[\text{SampleMSTedge}_{(n^2)} \,|X \geq T] \cdot \mathbb{P}[X \geq T] \\
\text{(by Lemma 1.1)} \quad &= \text{MST}/n - \mathbb{E}[\text{SampleMSTedge}_{(n^2)} \,|X \geq T]/T
\end{aligned}$$

As Algorithm 1 will always return either an edge weight or 0, we have $0 \leq \mathbb{E}[\text{SampleMSTedge}_{(n^2)} \,|X \geq T] \leq W_{\max}$. Together with $\text{MST}/(n-1) \geq W_{\min}$, $2/n > 1/(n-1)$ for $n \geq 2$, and $W = W_{max}/W_{min}$, this yields an overall bound of:

$$\frac{MST}{n} = \mathbb{E}[\text{SampleMSTedge}_{(n^2)}] \geq \mathbb{E}[\text{SampleMSTedge}_T] \geq \frac{MST}{n} - W_{\max} \cdot \frac{\epsilon}{4W} \geq \frac{MST}{n}(1 - \frac{\epsilon}{2})$$

Variance Since SampleMSTedge_T is more likely to return 0 immediately for smaller T, we have

$$\begin{aligned}
\text{Var}(\text{SampleMSTedge}_T) &\leq \mathbb{E}[(\text{SampleMSTedge}_T)^2] \leq \mathbb{E}[(\text{SampleMSTedge}_{(n^2)})^2] \\
\text{(by Lemma 1.1)} \quad &= \sum_w w^2 \frac{\#\text{MST edges of weight } w}{n} \leq W_{\max} \cdot \frac{\text{MST}}{n}
\end{aligned}$$

∎

As we now have the expectation and variance of SampleMSTedge_T, we can use Chebyshev inequality to prove Theorem 1.1.

THEOREM 1.1. *There is a randomized algorithm that computes a $(1+\epsilon)$ approximation of the MST weight using $O(Wd\epsilon^{-2} \log(W/\epsilon))$ queries in expectation and runs in $O(Wd\epsilon^{-2} \log(W/\epsilon) \log\log \frac{\log W}{\epsilon})$ expected time. The algorithm's output is correct with probability 3/4.*

Proof. We run SampleMSTedge_T for a total of $s = \Theta(W/\epsilon^2)$ times, average their results, and multiply by n. Let Z be this result, then by Lemma 2.2 we have

$$(1 - \epsilon/2)\,\text{MST} \leq \mathbb{E}[Z] \leq \text{MST}, \qquad \text{Var}(Z) \leq \text{MST} \cdot W_{\max} n/s.$$

Further, by Chebyshev-inequality we have

$$P[|Z - \mathbb{E}[Z]| > \frac{\epsilon}{2}\,\text{MST}] \leq 4\frac{\text{Var}(Z)}{\epsilon^2\,\text{MST}^2}$$

To bound this by failure probability by 3/4 we observe that

$$\frac{\text{Var}(Z)}{\epsilon^2\,\text{MST}^2} \leq \frac{n\,\text{MST} \cdot W_{\max}}{s\epsilon^2\,\text{MST}^2} \leq \frac{n \cdot W_{\max}}{s\epsilon^2\,\text{MST}} \leq \frac{n \cdot W_{\max}}{s\epsilon^2 n W_{\min}} = \frac{W}{s\epsilon^2}$$

Thus by $s = O(W/\epsilon^2)$ large enough, the failure probability is at most 3/4.

Time Complexity By having $s = O(W/\epsilon^2)$ calls to Algorithm 1 for $T = 4W/\epsilon$, the expected time complexity is $O(WdQ\epsilon^{-2}\log(W/\epsilon))$ by Lemma 2.1. Here Q is the time complexity of an insert/pop to the priority queue used by Prim's algorithm.

We can implement the priority queue to take $O(\log\log\frac{\log W}{\epsilon}$ time as follows. We round every encountered edge weight to the nearest $(1+\epsilon/2)^i$ for $i \in \mathbb{N}$, as that increases our MST estimate by at most a $(1 \pm \epsilon/2)$ factor. Then there are at most $O(\epsilon^{-1}\log W)$ distinct priorities in the priority queue, using the integer exponent i as the priority. Using a non-comparison based priority queue like the van-Emde-Boas-tree has $Q = O(\log\log\frac{\log W}{\epsilon})$ time complexity. Thus our MST algorithm runs in $O(Wd\epsilon^{-2}\log(W/\epsilon)\log\log\frac{\log W}{\epsilon})$ expected time. ∎

References

[1] Alexandr Andoni. COMS E6998-9: Algorithms for Massive Data, Spring 2019. https://www.cs.columbia.edu/~andoni/algoS19/scribes/scribe17.pdf.

[2] Sepehr Assadi. CS 514: Advanced Algorithms II – Sublinear Algorithms, Spring 2020. https://people.cs.rutgers.edu/~sa1497/courses/cs514-s20/problem-set1.pdf.

[3] Soheil Behnezhad. CS 7880 Algorithms for Big Data, Fall 2022. http://behnezhad.com/cs7880-fall22/scribes/lec-note-3.pdf.

[4] Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *SODA*, pages 129–162. SIAM, 2023.

[5] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *SODA*, pages 100–128. SIAM, 2023.

[6] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

[7] Artur Czumaj. Sublinear algorithms, August 2014. http://www.dcs.warwick.ac.uk/~czumaj/SFU/Slides.ppsx.

[8] Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM J. Comput.*, 35(1):91–109, 2005.

[9] Artur Czumaj and Christian Sohler. Sublinear-time algorithms. *Bull. EATCS*, 89:23–47, 2006.

[10] Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM J. Comput.*, 39(3):904–922, 2009.

[11] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geom. Appl.*, 18(1/2):3–28, 2008.

[12] Seth Gilbert. CS 5234 - Algorithms at Scale, Fall 2019. https://www.comp.nus.edu.sg/~gilbert/CS5234/2019/lectures/SublinearGraphAlgos.pdf.

[13] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[14] Elena Grigorescu. CS 590 Sublinear Algorithms, Spring 2021. https://www.cs.purdue.edu/homes/egrigore/590SLA-ST21/scribed-lect2.pdf.

[15] Piotr Indyk. Sublinear time algorithms for metric space problems. In *STOC*, pages 428–434. ACM, 1999.

[16] Rajesh Jayaram, Vahab Mirrokni, Shyam Narayanan, and Peilin Zhong. Massively parallel algorithms for high-dimensional euclidean minimum spanning tree. In *SODA*, pages 3960–3996. SIAM, 2024.

[17] Robert Krauthgamer. Seminar on sublinear time algorithms, 2010. https://www.wisdom.weizmann.ac.il/~robi/teaching/2010b-SeminarSublinearAlgs/.

[18] Sofya Raskhodnikova. Sublinear Algorithms, 2012. CSE 598A: https://cs-people.bu.edu/sofya/slides/WIM-sublinear-algorithms-lec2.pdf and 2020 CS 591: https://cs-people.bu.edu/sofya/sublinear-course/lecture-notes/lecture3.pdf.

[19] Ronitt Rubinfeld. Sublinear Time Algorithms, 2007. https://people.csail.mit.edu/ronitt/COURSE/S07/l11.pdf and 2020 https://people.csail.mit.edu/ronitt/COURSE/F20/Handouts/scribe2.pdf and 2022 https://people.csail.mit.edu/ronitt/COURSE/F22/Handouts/scribe2.pdf.

[20] Ronitt Rubinfeld. Sublinear time algorithms I, August 2022. FODSI Summer School, http://people.csail.mit.edu/ronitt/rrsublin1.pdf.

[21] Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM J. Discret. Math.*, 25(4):1562–1588, 2011.

[22] Grigory Yaroslavtsev. Sublinear algorithms for big datasets, 2014. http://grigory.us/big-data.html and 2015 CIS 700: http://grigory.us/big-data-class.html.

A Modification to Prim's Algorithm

Algorithm 3 Full SampleMSTedge$(G = (V, E), T)$. Changes to Prim's algorithm highlighted in blue.

Choose $X \geq 1$ according to $\mathbf{Pr}[X \geq k] = 1/k$ (i.e. $X \leftarrow \frac{1}{Y}$ for Y uniform from $(0, 1]$)
if $X \geq T$, **return** 0.
Choose $v_{\text{start}} \in V$ uniformly at random.
// Start running Prim's from v_{start}
$U \leftarrow \{v_{\text{start}}\}$ // visited vertices
$Tree \leftarrow \emptyset$ // this will be the MST
$Q \leftarrow \{(v, \{v_{\text{start}}, v\}, w(v_{\text{start}}, v)) \mid v \text{ adjacent to } v_{\text{start}}\}$
while $Q \neq \emptyset$ **do**
    $u, e, p \leftarrow$ **EXTRACT-MIN**(Q) // pop smallest edge weight p from queue
    if $u \in U$ **then**
        continue // skip this vertex if it was visited already
    end if
    $U \leftarrow U \cup \{u\}$
    $Tree \leftarrow Tree \cup \{e\}$
    for v adjacent to u **do**
        ADD$(Q, v, \{u, v\}, w(u, v))$
    end for
    if $\text{vol}(U) > X\dot{\deg}(v_{\text{start}})$ **then** // Early Stopping Condition
        return $\max_{e \in Tree} w(e)$ // return highest edge weight added to MST so far
    end if
end while
return 0