Sublinear Algorithms for Estimating Single-Linkage Clustering Costs

Pan Peng
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

Christian Sohler
Department of Mathematics and Computer Science, University of Cologne, Germany

Yi Xu
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

Abstract

Single-linkage clustering (SLC) is a fundamental method for hierarchical data analysis. In the distance setting, a k-clustering produced by SLC can be obtained by computing a minimum spanning tree (MST) and deleting its $k-1$ heaviest edges. This naturally induces a cost profile for the SLC hierarchy: for each $k \in [n]$, we define cost_k to be the weight of the resulting k-component spanning forest, equivalently, the minimum total weight of any spanning forest with exactly k connected components. The corresponding *SLC cost profile* is $(\mathrm{cost}_1, \ldots, \mathrm{cost}_n)$, and the scalar quantity $\mathrm{cost}(G) = \sum_{k=1}^{n} \mathrm{cost}_k$ is the area under this profile.

We study the problem of approximating these quantities in sublinear time. We assume that the input is a weighted graph G of average degree d with edge weights in $\{1, \ldots, W\}$, accessed through adjacency-list queries; missing edges are treated as having infinite distance. Our main result is a sampling-based algorithm that outputs a succinct sketch of the entire SLC cost profile in the distance setting. The algorithm runs in $\widetilde{O}(d\sqrt{W}/\varepsilon^3)$ time and returns a sketch from which one can derive estimates $(\widehat{\mathrm{cost}}_1, \ldots, \widehat{\mathrm{cost}}_n)$ satisfying

$$\sum_{k=1}^{n} \left| \widehat{\mathrm{cost}}_k - \mathrm{cost}_k \right| \leq \varepsilon \, \mathrm{cost}(G).$$

Thus, we obtain an ℓ_1 approximation to the full profile whose error is at most an ε-fraction of the area under the true profile. In particular, this yields a $(1 \pm \varepsilon)$-approximation to $\mathrm{cost}(G)$ within the same running time. We also prove a nearly matching lower bound of $\Omega(d\sqrt{W}/\varepsilon^2)$ queries for estimating $\mathrm{cost}(G)$.

We further extend our results to the similarity setting, where SLC is defined via a maximum spanning tree. In this case, we obtain algorithms with running time $\widetilde{O}(dW/\varepsilon^3)$ for both the profile and the total cost, together with a nearly matching lower bound of $\Omega(dW/\varepsilon^2)$ queries. These bounds reveal a genuine separation between the distance and similarity settings. Finally, we extend our algorithms to metric spaces, where we obtain $\widetilde{O}(n/\varepsilon^7)$-query algorithms for both distance and similarity metrics, and we complement our theory with experimental validation.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Sublinear Algorithms, Single-linkage Clustering, Approximation Algorithms, Minimum Spanning Tree

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.86

Related Version *Full Version*: https://arxiv.org/abs/2510.11547

Supplementary Material *Software (Source Code)*: https://doi.org/10.5281/zenodo.21278105

Funding *Pan Peng*: Supported in part by NSFC grant 62272431.
Christian Sohler: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 459420781.
Yi Xu: Supported in part by NSFC grant 62272431.

Acknowledgements The work was conducted in part while Pan Peng and Christian Sohler were visiting the Simons Institute for the Theory of Computing as participants in the Sublinear Algorithms program.

1 Introduction

Hierarchical clustering is a fundamental and widely used technique in data analysis and machine learning. In agglomerative (bottom-up) clustering, we start with a set of n objects and a defined distance or similarity relationship between these objects, as well as a method to evaluate the distance or similarity between clusters, i.e. groups of objects. The clustering process begins by treating each object as its own individual cluster. It then iteratively merges pairs of clusters based on a specified criterion related to their distance or similarity, continuing this process until only a single cluster remains. This merging process implicitly defines a hierarchy of clusters, and for every choice of the number of clusters k, the corresponding clustering is the one obtained after the first $n-k$ merges. Agglomerative clustering – typically implemented through methods such as single-linkage, complete-linkage, and average linkage – has been extensively studied and widely used in various fields, including data analysis and machine learning (see e.g. [17, 14]). These methods are well-established in standard data analysis toolkits.

In this work, we focus on one of the most widely used agglomerative clustering methods: single-linkage clustering (SLC). When the underlying relationship between objects is defined in terms of *distance*, SLC operates by iteratively identifying the *closest* pair of objects that belong to different clusters and connecting them with an edge. This approach emphasizes the merging of clusters based on proximity. To mathematically represent this process, we utilize a graph G where each vertex corresponds to an object, and the weight of an edge indicates the distance between these objects. In this framework (see e.g. [16]), SLC can be formalized by first constructing a minimum spanning tree (MST) T of G. The edges of T are then considered in *non-decreasing* order of their weights to facilitate cluster merging. This iterative process continues until the desired number of clusters, k, is reached, resulting in a k-clustering. In particular, when $k = 1$, it reveals the complete hierarchical structure of the graph. In cases where the relationship between objects is based on *similarity* rather than distance, SLC operates by iteratively identifying the *most similar* pair of objects that belong to different clusters and can be defined using a *maximum* spanning tree. Specifically, we first construct a maximum spanning tree (MaxST) T of graph G and then order the edges in T in *non-increasing* order of their weights, subsequently merging clusters iteratively.

SLC typically requires at least linear time for implementation. In many applications, the underlying graph can be massive, making computational analysis challenging – simply reading the input may become impractical or even impossible. This motivates to design *sublinear-time algorithms* that only read a small portion of the input to extract the SLC information. Since producing the full clustering hierarchy necessarily takes linear time, we instead ask:

Can one define ***cost measures*** *that summarize the structure of the single-linkage hierarchy, and can these quantities – as well as a* ***succinct representation*** *of the hierarchy itself – be approximated in sublinear time?*

1.1 Our Contributions

We answer the above question in the affirmative. Our main contribution is to formalize the *single-linkage cost profile* and to give nearly optimal sublinear-time algorithms for approximating this profile, the scalar summaries derived from it, and their analogues in the similarity and metric settings.

1.1.1 Formalizing the SLC Cost Profile in Distance Setting

Distance graphs and the single-linkage cost profile. We are given a connected weighted graph $G = (V, E, w)$ with edge weights $w(e) \in [1, W]$, for some parameter $W \geq 1$. The edge weights are the distances between the objects; for any pair i, j that does not form an edge, we define $w((i, j)) = \infty$.

In single-linkage clustering, the partition with k clusters is obtained by deleting the $k-1$ heaviest edges of an MST. This suggests a natural notion of cost at level k: the total weight of the resulting spanning forest. Let $w_1 \leq \cdots \leq w_{n-1}$ be the edge weights of an MST of G, listed in nondecreasing order. We define the *k-clustering cost* by

$$\mathrm{cost}_k := \sum_{i=1}^{n-k} w_i.$$

Equivalently, cost_k is the weight of the forest obtained by deleting the $k-1$ largest-weight edges of an MST.

The central object of this paper is the resulting *single-linkage cost profile*

$$\mathbf{p}(G) := (\mathrm{cost}_1, \mathrm{cost}_2, \ldots, \mathrm{cost}_n).$$

The profile vector records, for every level k, the cost of the k-clustering produced by SLC. Thus, rather than summarizing the hierarchy by a single number, $\mathbf{p}(G)$ captures its behavior across all resolutions.

The next proposition shows that cost_k is not an external objective imposed on single-linkage clustering. Rather, it is exactly the flat spanning-forest objective optimized by SLC at level k.

▶ **Proposition 1** (Flat spanning-forest characterization of SLC). *Let $G = (V, E, w)$ be a connected weighted graph, and let $\mathcal{F}_k$ denote the set of spanning forests of G with exactly k connected components. Then*

$$\mathrm{cost}_k = \min_{F \in \mathcal{F}_k} w(F), \qquad \text{where } w(F) := \sum_{e \in F} w(e).$$

Moreover, if T is an MST of G, then deleting the $k-1$ largest-weight edges of T yields an optimal forest attaining cost_k.

This is a standard consequence of Kruskal's algorithm and the cut and cycle properties of minimum spanning trees (see the full version for the proof).

Interpreting the SLC profile. The profile has a simple and useful derivative interpretation. If the MST edge weights are $w_1 \leq \cdots \leq w_{n-1}$, then

$$\Delta_k \;:=\; \mathrm{cost}_k - \mathrm{cost}_{k+1} \;=\; w_{n-k}.$$

Hence Δ_k is exactly the cost of the merge that reduces the hierarchy from $k+1$ clusters to k clusters. Large gaps in Δ_k therefore indicate natural scales at which further merging becomes expensive.

▶ **Example 2.** As a simple illustration, consider two graphs on five vertices whose MST edge weights are

$$G_A:\ 1,1,1,9 \qquad \text{and} \qquad G_B:\ 1,1,5,5.$$

Both have MST weight 12, but their SLC profiles are different:

$$\mathbf{p}(G_A) = (12,3,2,1,0), \qquad \mathbf{p}(G_B) = (12,7,2,1,0).$$

The first profile has a sharp final merge, indicating two well-separated high-level clusters, whereas the second has a much less pronounced elbow. Thus, the profile captures multiscale merge structure that is not visible from the MST weight alone.

The full version provides empirical evidence that the profile and its discrete derivative recover natural cluster scales and reflect hierarchical structure in both synthetic and real data.

Total cost as area under the profile. Alongside the full profile, we study the scalar quantity, which we call the *total cost* of the SLC hierarchy:

$$\mathrm{cost}(G) \ := \ \sum_{k=1}^{n} \mathrm{cost}_k = \sum_{k=1}^{n-1} \mathrm{cost}_k = \sum_{i=1}^{n-1} w_i + \sum_{i=1}^{n-2} w_i + \cdots + \sum_{i=1}^{2} w_i + w_1 = \sum_{i=1}^{n-1} (n-i) w_i \tag{1}$$

We interpret $\mathrm{cost}(G)$ as the *area under the SLC cost profile*. It is not intended to replace the level-wise quantities cost_k; rather, it serves as a compact summary of the entire hierarchy.

1.1.2 Sublinear Algorithms for Approximating the Profile and the Cost

Our first main result is a sublinear-time procedure that constructs a succinct sketch (also referred to as a succinct representation) of the entire SLC cost profile in the distance setting.

Computational model. Except in the metric-space setting discussed later, our algorithms operate in the *adjacency-list model*. In this model, given a vertex v, the algorithm may query its i-th neighbor (together with the corresponding edge weight) in $\Theta(i)$ time. The query complexity of an algorithm is the maximum number of such queries made on any input. Our lower bounds continue to hold even in the stronger model in which the i-th neighbor can be accessed in $O(1)$ time.

With this access model in place, we obtain the following profile guarantee.

▶ **Theorem 3.** *Assume that $\sqrt{W} \le n$ and let $0 < \varepsilon < 1$ be a parameter. Algorithm 3 generates a succinct representation of an approximation $(\widehat{\mathrm{cost}}_1, \ldots, \widehat{\mathrm{cost}}_n)$ of the SLC profile vector $(\mathrm{cost}_1, \ldots, \mathrm{cost}_n)$ in the distance graph such that with probability at least $3/4$, it holds that*

$$\sum_{k=1}^{n} \left| \widehat{\mathrm{cost}}_k - \mathrm{cost}_k \right| \le \varepsilon \cdot \mathrm{cost}(G).$$

The query complexity and running time of the algorithm are $O(\frac{\sqrt{W}d}{\varepsilon^3} \log^4(\frac{Wd}{\varepsilon}))$ in expectation.

The guarantee in Theorem 3 is an ℓ_1 approximation to the entire SLC profile. It should *not* be interpreted as a uniform multiplicative approximation that holds simultaneously for every level k; such a guarantee is impossible in sublinear time (see Proposition 6).

Nevertheless, the sketch supports efficient point queries. Given the succinct representation and any specified $k \in \{1, \dots, n\}$, Algorithm 4 returns an estimate $\widehat{\text{cost}}_k$ of cost_k in time $O(\log(\log W/\varepsilon))$. Moreover, Lemma 22 shows that, for every k,

$$\left|\widehat{\text{cost}}_k - \text{cost}_k\right| = O\Big(\varepsilon\big(\text{cost}_k + \max\{k, n/\sqrt{W}\}\, W\big)\Big).$$

We also obtain a sublinear-time estimator for the scalar quantity $\text{cost}(G)$.

▶ **Theorem 4.** *Let G be a weighted graph with edge weights in $\{1, \dots, W\}$ with average (unweighted) degree d. Assume that $\sqrt{W} \le n$ and let $0 < \varepsilon < 1$ be a parameter. Algorithm 2 outputs an estimate $\widehat{\text{cost}}(G)$ of the single-linkage clustering cost $\text{cost}(G)$ in the distance graph such that with probability at least $3/4$,*

$$(1-\varepsilon)\text{cost}(G) \le \widehat{\text{cost}}(G) \le (1+\varepsilon)\text{cost}(G).$$

The query complexity and running time of the algorithm are $O(\frac{\sqrt{W}d}{\varepsilon^3}\log^4(\frac{Wd}{\varepsilon}))$ in expectation.

To complement the upper bound, we prove a nearly matching lower bound for estimating $\text{cost}(G)$ in the distance setting.

▶ **Theorem 5.** *Let $\frac{W^{1/4}}{\sqrt{40n}} < \varepsilon < \frac{1}{2}$ and $W > 1$. Any algorithm that $(1+\varepsilon)$-approximates the SLC cost $\text{cost}(G)$ in the distance graph with success probability at least $3/4$ needs to make $\Omega(d\sqrt{W}/\varepsilon^2)$ queries.*

More discussions on the profile guarantee. Note that Theorem 3 gives only an average guarantee across the levels of the hierarchy: informally, it approximates the cost of the k-clustering within a $(1+\varepsilon)$ factor on average over k. We further show that this is essentially the strongest per-level guarantee attainable in sublinear time, as no sublinear-time algorithm can achieve a $(1+\varepsilon)$-approximation to cost_k uniformly over all levels k.

▶ **Proposition 6** (Uniform per-level relative approximation is impossible)**.** *No sublinear-time algorithm can give a uniform multiplicative approximation to cost_k for all k. In particular, even approximating cost_{n-1} within a factor $1+\varepsilon$, for any $\varepsilon < \sqrt{2}-1$, requires $\Omega(m)$ edge inspections in the worst case, even when all edge weights lie in $\{1, 2\}$.*

The obstruction already appears at the finest nontrivial level: cost_{n-1} is exactly the minimum edge weight in the graph. Distinguishing an all-2-weight graph from one containing a single hidden edge of weight 1 already requires linear inspection in the worst case.

1.1.3 Similarity Setting and Metric Spaces

We next extend the framework to the similarity setting and generalize the algorithms to metric spaces. The detailed pseudocode and formal analysis of these extensions are deferred to the full version.

1.1.3.1 SLC in the similarity graph

We consider weighted graphs $G = (V, E, w)$ in which each edge weight $w(e) \in [1, W]$ represents similarity, and non-edges are assigned similarity 0. Let $w_1^{(s)} \geq \cdots \geq w_{n-1}^{(s)}$ be the edge weights of a maximum spanning tree (MaxST) of G, listed in non-increasing order. For each $k \in \{1, \ldots, n\}$, define

$$\mathrm{cost}_k^{(s)} := \sum_{i=1}^{n-k} w_i^{(s)}.$$

Equivalently, $\mathrm{cost}_k^{(s)}$ is the weight of the forest obtained by deleting the $k-1$ smallest-weight edges of a MaxST. The corresponding *similarity cost profile* is

$$\mathbf{p}^{(s)}(G) = (\mathrm{cost}_1^{(s)}, \mathrm{cost}_2^{(s)}, \ldots, \mathrm{cost}_n^{(s)}).$$

As in the distance setting, this quantity admits a flat optimization interpretation.

▶ **Proposition 7** (Flat spanning-forest characterization in the similarity setting)**.** *Let $\mathcal{F}_k$ denote the set of spanning forests of G with exactly k connected components. Then*

$$\mathrm{cost}_k^{(s)} = \max_{F \in \mathcal{F}_k} w(F).$$

Moreover, if T is a maximum spanning tree of G, then deleting the $k-1$ smallest-weight edges of T yields an optimal forest attaining $\mathrm{cost}_k^{(s)}$.

We also define the total similarity hierarchy cost

$$\mathrm{cost}^{(s)}(G) = \sum_{k=1}^{n} \mathrm{cost}_k^{(s)} = \sum_{k=1}^{n-1} \mathrm{cost}_k^{(s)} = \sum_{i=1}^{n-1} (n-i) \cdot w_i^{(s)}, \tag{2}$$

which is the area under the similarity profile. If the MaxST edge weights are sorted as above, then the discrete derivative satisfies

$$\Delta_k^{(s)} := \mathrm{cost}_k^{(s)} - \mathrm{cost}_{k+1}^{(s)} = w_{n-k}^{(s)}.$$

Thus, the derivative records the exact similarity threshold of the merge that reduces the hierarchy from $k+1$ clusters to k clusters.

Sublinear Algorithms for Similarity Graphs. We obtain sublinear-time algorithms for approximating both the similarity profile and the total hierarchy cost.

▶ **Theorem 8.** *Assume that $W \leq n$ and let $\varepsilon < 1$. There exists an algorithm that generates a succinct representation of an approximation $(\widehat{\mathrm{cost}^{(s)}}_1, \ldots, \widehat{\mathrm{cost}^{(s)}}_n)$ of the SLC profile vector $(\mathrm{cost}_1^{(s)}, \ldots, \mathrm{cost}_n^{(s)})$ in the similarity graph such that with probability at least $3/4$, it holds that*

$$\sum_{k=1}^{n} |\widehat{\mathrm{cost}^{(s)}}_k - \mathrm{cost}_k^{(s)}| \leq \varepsilon \cdot \mathrm{cost}^{(s)}(G).$$

The query complexity and running time of the algorithm are $O(\frac{Wd}{\varepsilon^3} \log^4(\frac{Wd}{\varepsilon}))$ in expectation.

▶ **Theorem 9.** *Let G be a weighted graph with edge weights in $\{1,\dots,W\}$ with average (unweighted) degree d. Assume that $W \le n$ and let $0 < \varepsilon < 1$ be a parameter. Algorithm 6 outputs an estimate $\widehat{\mathrm{cost}^{(s)}}(G)$ of the single-linkage clustering cost $\mathrm{cost}^{(s)}(G)$ in the similarity graph such that with probability at least $3/4$,*

$$(1-\varepsilon)\mathrm{cost}^{(s)}(G) \le \widehat{\mathrm{cost}^{(s)}}(G) \le (1+\varepsilon)\mathrm{cost}^{(s)}(G).$$

The query complexity and running time of the algorithm are $O(\frac{Wd}{\varepsilon^3}\log^4(\frac{Wd}{\varepsilon}))$ in expectation.

To complement our algorithmic result, we establish a nearly matching information-theoretic lower bound for estimating $\mathrm{cost}^{(s)}(G)$.

▶ **Theorem 10.** *Let $\sqrt{\frac{W}{40n}} < \varepsilon < \frac{1}{2}$ and $W > 10$. Any algorithm that $(1+\varepsilon)$-approximates the SLC cost $\mathrm{cost}^{(s)}(G)$ in the similarity graph with success probability at least $3/4$ needs to make $\Omega(dW/\varepsilon^2)$ queries.*

Together, these bounds reveal a fundamental separation between the two settings: for any small constant $\varepsilon > 0$, distance-based estimation admits $\widetilde{\Theta}(d\sqrt{W})$ complexity, whereas similarity-based estimation inherently requires $\widetilde{\Theta}(dW)$ complexity.

1.1.3.2 Metric Space

Finally, we extend our results to metric spaces, where the relationship between vertices must satisfy the triangle inequality. In this model, the algorithm does not receive an explicit graph; instead, it may query the distance or similarity of any specified pair of points in $O(1)$ time.

When the metric represents distances, we achieve the following guarantee:

▶ **Theorem 11.** *Let G be an n-point graph in metric space, where each edge weight represents* distance *between two vertices, and let $0 < \varepsilon < 1$. There exists an algorithm that outputs an estimate $\widehat{\mathrm{cost}}(G)$ of single-linkage clustering cost $\mathrm{cost}(G)$ in metric space, such that with probability at least $3/4$,*

$$(1-\varepsilon)\mathrm{cost}(G) \le \widehat{\mathrm{cost}}(G) \le (1+\varepsilon)\mathrm{cost}(G).$$

The query complexity and running time of the algorithm are $\tilde{O}(n/\varepsilon^7)$ in expectation.

An analogous result, with the same approximation guarantee and query complexity, holds when the metric encodes similarities. Moreover, these bounds are nearly optimal in their dependence on n: even for constant ε, any $(1+\varepsilon)$-approximation requires $\Omega(n)$ oracle queries.

1.2 Technical Overview

Although the SLC profile is the primary object of the paper, the algorithmic development begins with the scalar quantity $\mathrm{cost}(G)$, the area under the profile. This is where the central technical difficulty already appears, while also being conceptually simpler. Once we show how to estimate $\mathrm{cost}(G)$ in sublinear time, the same machinery can be lifted to a succinct oracle for the entire profile.

Our starting point is the component-counting paradigm of Chazelle, Rubinfeld, and Trevisan [8] (CRT) for estimating the MST weight, but the single-linkage setting introduces two difficulties that are absent from MST estimation. First, the MST weight is a *linear* functional of the connected-component counts of threshold graphs, whereas the SLC hierarchy cost is a *quadratic* functional. Second, our goal is not merely to estimate one scalar, but to

construct a succinct approximation to an entire monotone hierarchy. Moreover, the algorithm only has access to *noisy* estimates of the component counts, so the monotonicity needed for binary search is not directly available. The technical overview below explains how we overcome these obstacles.

Starting Point: the CRT Paradigm. Given a weighted graph G with edge weights in $\{1, \dots, W\}$, let G_j be the subgraph containing all edges of weight at most j, and let c_j be its number of connected components. CRT showed that the MST weight of G can be written as

$$\text{cost}(\text{MST}) = n - W + \sum_{j=1}^{W-1} c_j,$$

and gave a sublinear algorithm for estimating each c_j by sampling vertices and performing truncated BFS. Their estimator achieves additive error εn in expected time $\widetilde{O}(d/\varepsilon^2)$, and combining the estimates over all thresholds yields a $\widetilde{O}(dW/\varepsilon^2)$-time approximation to the MST weight.

We use this framework as a starting point, but the profile problem cannot be reduced to a direct application of CRT. The reason is that neither the target quantity nor the required accuracy regime matches MST estimation.

Distance Setting: Estimating cost(G). While Equation (1) defines the total cost $\text{cost}(G)$ based on the sorted MST edge weights, finding the exact MST requires at least linear time. We therefore reformulate $\text{cost}(G)$ in terms of the threshold graphs G_j. We show in Theorem 13 that

$$\text{cost}(G) = \frac{n(n-1)}{2} + \frac{1}{2} \sum_{j=1}^{W-1} (c_j^2 - c_j).$$

This identity is the first major departure from CRT: the hierarchy cost is a quadratic, not linear, functional of the connected-component counts.

A naive strategy would therefore estimate every c_j separately and plug the results into the formula above. However, to obtain a $(1+\varepsilon)$ approximation to $\text{cost}(G)$, one would need much finer additive control on the c_j's than in CRT, and doing this for all W thresholds would be too expensive.

To address this, we first refine the component estimator itself. We show in Corollary 12 that each c_j can be estimated within additive error

$$O\left(\varepsilon \max\left\{\frac{n}{\sqrt{W}}, c_j\right\}\right)$$

in expected time $\widetilde{O}(\sqrt{W}d/\varepsilon^2)$. This already improves substantially over the naive reduction, but estimating all W values individually would still be too slow.

The second step is to exploit monotonicity. Since G_j gains edges as j increases, the sequence $(c_1, \dots, c_W)$ is nonincreasing, and hence so is $(c_j^2 - c_j)$. This suggests partitioning the range of component counts into intervals, representing all values in one interval by a single breakpoint, and then using binary search to determine how many values fall into each interval. If one could access the exact sequence (c_j), this would reduce the number of required probes from W to only $O(\log^2 W/\varepsilon)$.

The main obstacle is that we do *not* observe (c_j) exactly. We only have noisy estimates $\widehat{c}_j$, and the sequence $(\widehat{c}_1, \dots, \widehat{c}_W)$ need not be monotone. This leads to our main algorithmic abstraction.

Noisy Monotone-Sequence Sketching. In Section 2.3, we isolate the following abstract problem: given a nonincreasing sequence

$$X = (x_1, \dots, x_W)$$

and only noisy estimates $\widehat{x}_j$ satisfying

$$|\widehat{x}_j - x_j| \le T_j,$$

how can one build a succinct approximation of the entire sequence without querying every entry?

Our solution is a robust binary-search framework for noisy monotone sequences. We define a *valid discretization* of the value range into intervals whose widths are chosen to dominate the local error bounds T_j. This guarantees that if x_j lies in a given interval, then $\widehat{x}_j$ lies in the same interval or a neighboring one. Even though the noisy sequence may not be sorted, binary search on the $\widehat{x}_j$'s still preserves a weak monotonicity: searching for smaller keys never returns smaller indices. This is enough to recover consistent interval counts and hence a succinct approximation to the sequence.

Applying this abstraction to (c_j) yields a sublinear-time estimator for $\mathrm{cost}(G)$ using only $O(\log^2 W/\varepsilon)$ threshold queries, for a total running time of $\widetilde{O}\Big(\frac{\sqrt{W}d}{\varepsilon^3}\Big)$.

From the Scalar Estimator to a Profile Oracle. We then lift the same sketching framework from the scalar quantity $\mathrm{cost}(G)$ to the entire profile $\mathbf{p}(G) = (\mathrm{cost}_1, \dots, \mathrm{cost}_n)$.

The key additional observation (see Equation (3)) is that for each k,

$$\mathrm{cost}_k = n + \sum_{j=1}^{w_{n-k}-1} c_j - k\, w_{n-k},$$

where w_{n-k} is the $(n-k)$-th MST edge weight. In particular, when $k = c_j$, this gives an explicit formula for cost_k in terms of the threshold counts up to level j.

Using the same interval endpoints obtained from the noisy binary-search procedure, we define representative values $\overline{\mathrm{cost}}_{B_i}$ at the breakpoints and use them as a succinct profile sketch. Since the profile is monotone in k, every cost_k can be approximated by the representative associated with the interval containing k. This yields a succinct profile oracle with two properties: it answers any specified level k in $O(\log(\log W/\varepsilon))$ time, and the total ℓ_1 error over all levels is at most $\varepsilon\mathrm{cost}(G)$.

Thus, although the scalar quantity $\mathrm{cost}(G)$ is the technical entry point, the same machinery ultimately provides a succinct approximation to the full hierarchy.

Similarity Setting. The similarity setting is *not* a formal mirror image of the distance case. Here the natural threshold graph $G_j^{(s)}$ contains all edges of similarity at least j, and if $c_j^{(s)}$ denotes its number of connected components, then we derive the identity

$$\mathrm{cost}^{(s)}(G) = \frac{1}{2}\sum_{j=1}^{W}\big(c_j^{(s)} + n - 1\big)\big(n - c_j^{(s)}\big).$$

The delicate term is now not just $c_j^{(s)}$, but also

$$D_j := n - c_j^{(s)},$$

which measures how far the threshold graph is from being completely disconnected.

A direct adaptation of the distance-case component estimator is insufficient, because when $c_j^{(s)}$ is large, the quantity D_j may be small, and an additive error proportional to $c_j^{(s)}$ is too crude. To resolve this, we separately estimate the number of non-isolated vertices and the number of connected components among the non-isolated part of the threshold graph. This yields a refined estimator for D_j with additive error

$$O\Big(\varepsilon \max\Big\{\frac{n}{W},\, \min\{D_j, n - D_j\}\Big\}\Big).$$

We then apply the same noisy binary-search framework as in the distance case, but the interval design is more intricate because the error bounds now behave differently in different regimes of D_j. This leads to a partition of the range $[0, n-1]$ into five subranges, each handled by either an arithmetic or a geometric progression of breakpoints. The resulting algorithm runs in $\widetilde{O}\big(\frac{Wd}{\varepsilon^3}\big)$ time, which is asymptotically larger than in the distance case.

Lower Bounds and Metric Spaces. Our lower bounds build on the same distributional framework as CRT, namely the problem of distinguishing between two biased coin distributions. We encode these instances into weighted graph distributions whose single-linkage hierarchy costs differ by a factor of $1+\varepsilon$, yet cannot be distinguished with too few queries. This yields nearly matching lower bounds in both the distance and similarity settings and, in particular, explains the separation between the $\widetilde{O}(d\sqrt{W})$ complexity of the distance case and the $\widetilde{O}(dW)$ complexity of the similarity case.

Finally, we extend the framework to metric spaces. Here our starting point is the representative-point method of Czumaj and Sohler [11] for MST estimation in metrics. We express the hierarchy cost as a quadratic functional of component counts across $\widetilde{O}(\log n/\varepsilon)$ threshold graphs and combine this with a refined analysis of the representative-based estimator. This yields sublinear-time approximation algorithms for both distance and similarity metrics.

1.3 Related Work

Hierarchical clustering has been extensively studied in the context of approximation algorithms with polynomial running times. A significant body of work has explored hierarchical clustering in metric and Euclidean spaces, including studies on hierarchical k-median [21] and the use of the largest cluster radius as a cost measure [13]. Recently, [12] introduced a cost function for similarity-based hierarchical clustering and proposed an approximation algorithm for this cost. Several improvements and generalizations have since emerged [6, 22, 9, 7]. Other recent work on hierarchical correlation clustering and fitting distances in ultrametrics, such as [2], has further improved approximation algorithms in the hierarchical setting.

In terms of sublinear algorithms for hierarchical clustering, [1] studied such algorithms concerning Dasgupta cost [12] within dynamic streaming, query, and massively parallel computation models. [5] also considered affinity and single-linkage clustering in the massively parallel computation setting. [4] focused on streaming algorithms for identifying a hierarchical clustering with low Dasgupta cost and estimating the value of the optimal hierarchical tree. Additionally, [19] provided a sublinear-time hierarchical clustering oracle for graphs exhibiting significant flat clustering, and [20] estimated Dasgupta cost for well-clusterable graphs in sublinear-time. Other sublinear algorithms addressing graph clustering with conductance-based measures include local graph clustering [26, 3] and spectral clustering oracles for well-clusterable graphs [23, 15, 25].

Our work is closely related to a series of studies on estimating the number of connected components and the weight of minimum spanning trees. [8] pioneered a sublinear-time algorithm for these problems using sampling and truncated BFS. Subsequent investigations have explored these problems in various contexts, including Euclidean space [10], metric space [11], and graph streaming [18, 24].

Paper Organization. Due to space constraints, we present only proof sketches for estimating $\mathrm{cost}(G)$ and the profile vector in the distance setting, and briefly introduce the similarity setting in the Appendix. Complete proofs and the remaining settings and extensions are deferred to the full version.

2 Proof Sketches for Estimating $\mathrm{cost}(G)$ in Distance Setting

We start by estimating the number of connected components, and then show that computing the cost $\mathrm{cost}(G)$ can be reduced to this problem. We next abstract the task and introduce a binary search method to speed up the estimation, and then show how to apply this method to our specific setting.

2.1 Estimating the Number of Connected Components

We first give an algorithm that efficiently approximates the number of connected components in a subgraph H of a graph G. This approximation serves as a starting point in our overall methodology and builds upon an algorithm to approximate the number of connected components and the cost of a minimum spanning tree from [8].

By applying median trick to the algorithm in [8], we have the following corollary.

▶ **Corollary 12.** *Let $1 > \varepsilon > 0$, $1 > \delta > 0$ and $k \geq 1$. Suppose (an upper bound on) the average degree d of graph G is known. Given access to the graph G and an implicit subgraph $H \subseteq G$ (where H shares the same vertex set as G, and the edges of H are determined by evaluating conditions on the edges of G), there exists an algorithm that outputs $\hat{c}$ such that*

$$|\hat{c} - c| \leq \varepsilon \cdot \max\left\{\frac{n}{k}, c\right\}$$

with probability at least $1 - \delta$, where c is the number of connected components in the subgraph H. The query complexity and running time of the algorithm are $O(\frac{k \log(1/\delta)}{\varepsilon^2} d \log(\frac{k}{\varepsilon} d))$ in expectation.

2.2 Reduction to Estimating Connected Components

Recall that the cost of a hierarchical SLC is $\mathrm{cost}(G) = \sum_{k=1}^{n} \mathrm{cost}_k = \sum_{i=1}^{n-1} (n-i) \cdot w_i$, where w_i is the i-th smallest edge weight in the MST. Similarly to the work on approximating the cost of an MST [8], we reduce the problem by expressing $\mathrm{cost}(G)$ as a function of the number of connected components in certain subgraphs of G. For this purpose, let G_j denote the subgraph induced by edges of weight at most j and let c_j denote the number of connected components in G_j. In particular, we let G_0 denote the graph consisting of n singleton vertices, so $c_0 = n$. We assume the graph G is connected, so that the top cluster (corresponding to 1-SLC) contains all vertices.

Assume that the edge weights of G are integers. If this is not the case, one may rescale the weights by a factor of $\approx 1/\varepsilon$ and round them to their closest integer.

▶ **Theorem 13.** *Let G be a connected graph on n vertices, with edge weights from $\{1, \ldots, W\}$. Then*

$$\text{cost}(G) = \frac{n(n-1)}{2} + \frac{1}{2} \cdot \sum_{j=1}^{W-1} (c_j^2 - c_j).$$

Proof. Let n_j be the number of edges in the MST with weight j. Observe that the number of edges in the MST with weight 1 is $n_1 = n - c_1 = c_0 - c_1$ and the number of edges with weight 2 is $n_2 = c_1 - c_2$, ..., the number of edges with weight j is $n_j = c_{j-1} - c_j$ for any integer $j \in \{1, \ldots, W\}$. By our assumption that the graph G is connected, $c_W = 1$. Then we have,

$$\begin{aligned}
\text{cost}(G) &= \sum_{i=1}^{n-1} (n-i) \cdot w_i && \text{(by Equation (1))} \\
&= \sum_{i=1}^{n_1} (n-i) \cdot 1 + \sum_{i=n_1+1}^{n_1+n_2} (n-i) \cdot 2 + \ldots + \sum_{i=n_1+\ldots+n_{W-1}+1}^{n_1+\ldots+n_W} (n-i) \cdot W \\
&&& \text{(reorganize the sum by grouping the contributions by edge weights)} \\
&= \sum_{i=1}^{n-c_1} (n-i) \cdot 1 + \sum_{i=n-c_1+1}^{n-c_2} (n-i) \cdot 2 + \cdots + \sum_{i=n-c_{W-1}+1}^{n-c_W} (n-i) \cdot W \\
&&& \text{(since } n_j = c_{j-1} - c_j\text{)} \\
&= \sum_{i=1}^{n-c_W} (n-i) + \sum_{i=n-c_1+1}^{n-c_2} (n-i) \cdot 1 + \cdots + \sum_{i=n-c_{W-1}+1}^{n-c_W} (n-i) \cdot (W-1) \\
&&& \text{(factoring out } (n-i) \text{ from each summation and combining the terms)} \\
&= \sum_{i=1}^{n-c_W} (n-i) + \sum_{i=n-c_1+1}^{n-c_W} (n-i) + \cdots + \sum_{i=n-c_{W-1}+1}^{n-c_W} (n-i) \\
&&& \text{(repeating this decomposition until all summations end with } i = n - c_W\text{)} \\
&= \sum_{i=1}^{n-1} i + \sum_{i=1}^{c_1-1} i + \cdots + \sum_{i=1}^{c_{W-1}-1} i \\
&&& \text{(substituting } (n-i) \text{ with } i\text{, and noting that } c_W = 1\text{)} \\
&= \sum_{j=0}^{W-1} \frac{(1 + c_j - 1) \cdot (c_j - 1)}{2} && \text{(noting that } c_0 = n\text{)} \\
&= \frac{n(n-1)}{2} + \frac{1}{2} \cdot \sum_{j=1}^{W-1} (c_j^2 - c_j) && \blacktriangleleft
\end{aligned}$$

2.3 Noisy Monotone Sequence Sketching

From the above cost formula, estimating $\text{cost}(G)$ reduces to estimating the sum $\sum_{j=1}^{W-1}(c_j^2 - c_j)$. A key observation is that the sequence $(c_1, \ldots, c_W)$ is non-increasing: as j increases, the subgraph G_j includes more edges, leading to fewer connected components. This monotonicity suggests the possibility of summarizing the sequence succinctly using only a few representative values. However, we only have access to estimated values $\hat{c}_j$, and estimation errors may disrupt the monotonicity. This motivates us to study an abstract version of the problem: how to efficiently approximate the sum over a non-increasing sequence when only noisy estimates

are available. This abstraction also arises in our algorithm for similarity-based clustering, and we believe it could be of independent interest in other sublinear or local computation settings.

Now, let us describe this abstract problem in a bit more formal manner: Given a noisy version of a *non-increasing* sequence X with elements $(x_1, x_2, \ldots, x_W)$, and each element x_j from $[L, R]$ (where L and R are lower and upper bounds for the smallest and largest element of X), we want to find a succinct approximation of every x_j without approximating each x_j separately.

Initially, assume the exact values of X are known. We divide the range $[L, R]$ into $t-1$ intervals, for some integer t to be chosen later. The intervals are specified by some numbers $L = B_t < B_{t-1} < \cdots < B_1 = R$. Specifically, we divide $[L, R]$ into:

$$[B_t, B_{t-1}], (B_{t-1}, B_{t-2}], \ldots, (B_2, B_1]$$

Then to get our estimates we do the following. For each $i \in \{1, \ldots, t\}$ we perform binary search on the sequence $X' = (x_1, \ldots, x_W, L)$ to find the smallest index $j_i \in \{1, \ldots, W+1\}$, $1 \leq i \leq t-1$ such that $x_{j_i} \leq B_i$ and where we define $j_t = W + 1$. This results in a non-decreasing sequence $(j_1, j_2, \ldots, j_t)$ that partitions $\{1, \ldots, W\}$ sets into $J_i = \{j_i, \ldots, j_{i+1} - 1\}, 1 \leq i \leq t-1$ (where the set is empty when $j_i = j_{i+1}$). Then every x_j with $j \in J_i$ is estimated by B_i.

Now consider that we only have access to estimates $\hat{x}_j \in [L, R]$ of the x_j in X. In particular, we assume that there is a sequence $T = (T_1, \ldots, T_W)$ of error bounds T_j, such that $|\hat{x}_j - x_j| \leq T_j$. To find succinct approximation of all entries in X, we can still apply the approach sketched above and perform a variant of the standard binary search on the sequence $\widehat{X}' = (\hat{x}_1, \ldots, \hat{x}_W, L)$ corresponding to the estimates, searching for each interval endpoint B_i. For completeness, we give the pseudocode in Algorithm 1. The algorithm is initialized with $\ell = 1$, $r = W + 1$, $B = B_i$, and sequence $\widehat{X}'$.

Algorithm 1 BinarySearch$(\widehat{X}', \ell, r, B)$.

Input: estimated array $\widehat{X}'$, leftmost and rightmost indices ℓ and r, search key B
Output: index i

```
if ℓ = r then
    return ℓ
else
    m ← ⌊(ℓ+r)/2⌋
    if x̂_m ≤ B then
        return BinarySearch(X̂', ℓ, m, B)
    else
        return BinarySearch(X̂', m+1, r, B)
    end if
end if
```

We note that, since we only have access to the estimates $\hat{x}_j$, the sequence $\widehat{X}' = (\hat{x}_1, \ldots, \hat{x}_W, L)$ may not be non-increasing. Furthermore, we note that Algorithm 1 always returns the index that is reached at the end of the recursion when ℓ and r become equal. We show that our binary search approach satisfies a certain monotonicity property, namely, that the indices $\hat{j}_i$ found by the searches for the interval endpoints B_i are in non-decreasing order. This is stated formally in the following lemma.

▶ **Lemma 14.** *Let $a, b \in [L, R]$ with $a < b$. Let $\hat{j}_a$ and $\hat{j}_b$ be the two indices returned by Algorithm 1 on a (possibly unsorted) input sequence $\widehat{X}'$ with $B = a$ and $B = b$, respectively. Then we have $\hat{j}_a \geq \hat{j}_b$.*

We can apply the above lemma directly on the B_i to obtain the following corollary.

▶ **Corollary 15.** *For $i \in \{1, \ldots, t-1\}$ let $\hat{j}_i$ be the index returned when Algorithm 1 is run on a (possibly unsorted) input sequence $\widehat{X}'$ with $B = B_i$ and let $\hat{j}_t = W + 1$. Then $\hat{j}_1 \leq \hat{j}_2 \leq \cdots \leq \hat{j}_t$.*

Given the above corollary, our plan is to simply search for the interval endpoints B_i and denote the corresponding resulting indices $\hat{j}_i$, and we always set $\hat{j}_t = W + 1$. We define $\hat{J}_i = \{\hat{j}_i, \ldots, \hat{j}_{i+1} - 1\}, 1 \leq i \leq t-1$, as in the case of binary search with error. This ensures that the $\hat{J}_i$ form a partition of $\{1, \ldots, W\}$. We then define the notion of a valid discretization with respect to (w.r.t.) X and T that is a partition of $[L, R]$ into intervals, such that for every x_i, the error of the interval containing x_i is at most the minimum length of the neighboring intervals.

▶ **Definition 16.** *Let $X = (x_1, \ldots, x_W)$ be a non-increasing sequence with $x_j \in [L, R]$ for all $j \in \{1, \ldots, W\}$ and let $T = (T_1, \ldots, T_W)$ be a sequence of error bounds. Let $\widehat{X} = (\hat{x}_1, \ldots, \hat{x}_W)$, $\hat{x}_j \in [L, R]$ with $|\hat{x}_j - x_j| \leq T_j$. We say that a sequence of interval endpoints $L = B_t < B_{t-1} < \cdots < B_1 = R$ is a* valid discretization *of $[L, R]$ w.r.t. X and T, if for every $j \in \{1, \ldots, W\}$ and $i \in \{1, \ldots, t-1\}$ such that $B_{i+1} \leq x_j \leq B_i$ we have that*

- *$T_j < B_{i-1} - B_i$, if $i \geq 2$, and*
- *$T_j < B_{i+1} - B_{i+2}$, if $i \leq t-2$.*

Intuitively, this ensures that whenever we search for a key $\hat{x}_j$ with $B_{i+1} \leq x_j \leq B_i$ we will end up in the set of indices $\hat{J}_i$ or a neighboring one.

For every $j \in \hat{J}_i$ we now define the estimator for x_j to be $\bar{x}_j = B_i$. In the following lemma, we formalize our intuition and show that the value of x_j is between B_{i+2} and B_{i-1} and thus $\bar{x}_j$ is close to x_j if our discretization is sufficiently fine.

▶ **Lemma 17.** *Let $X = (x_1, \ldots, x_W)$ be a non-increasing sequence with $x_j \in [L, R]$ for all $j \in \{1, \ldots, W\}$ and let $T = (T_1, \ldots, T_W)$ be a sequence of error bounds. Let $\widehat{X} = (\hat{x}_1, \ldots, \hat{x}_W)$, where $\hat{x}_j \in [1, n]$ with $|\hat{x}_j - x_j| \leq T_j$ for any $j \in [W]$. Let $L = B_t < \cdots < B_1 = R$ be a sequence of interval endpoints that is a valid discretization of $[L, R]$ w.r.t. X and T. Let $\hat{j}_i$ be the index returned by Algorithm 1 on $\widehat{X}' = (\hat{x}_1, \ldots, \hat{x}_W, L)$ for search key $B = B_i$ and let $\hat{J}_i = \{\hat{j}_i, \ldots, \hat{j}_{i+1} - 1\}$ and define $\hat{j}_t = W + 1$. For any $1 \leq i \leq t-1$ and every $j \in \hat{J}_i$ we have*

$$B_{i+2} \leq x_j \leq B_{i-1},$$

where we define $B_0 = \infty$ and $B_{t+1} = -\infty$.

2.4 Estimating $\mathrm{cost}(G)$ in Sublinear Time

To estimate $\mathrm{cost}(G)$, we leverage the non-increasing nature of c_j. Since computing the exact c_j values requires linear time, we apply the binary search technique from Section 2.3 over a sequence of noisy estimates $\hat{c}_j$, and its error bound is defined in the following lemma.

▶ **Lemma 18.** *For any $1 \leq j \leq W$, let $\hat{c}_j = \min\{\max\{\hat{c}'_j, 1\}, n\}$, where $\hat{c}'_j$ is obtained from Corollary 12 with input G, $H = G_j$, $\varepsilon/8$, $k = \sqrt{W}$, d, $\delta = 1/(4W)$. Then with probability at least $3/4$, it holds that for* all *$1 \leq j \leq W$, $\hat{c}_j \in [1, n]$, and*

$$|\hat{c}_j - c_j| \leq T_j := \frac{\varepsilon}{8} \cdot \max\left\{\frac{n}{\sqrt{W}}, c_j\right\}.$$

Throughout the following, we will assume that for all j*, the inequality* $|\hat{c}_j - c_j| \le T_j$ *holds, and* $\hat{c}_j \in [1, n]$. According to Lemma 18, this occurs with probability at least 3/4.

Next, we define the endpoints of intervals that partition the range $[1, n]$.

▶ **Definition 19.** *Let* $0 < \varepsilon < 1$, t_1 *be the largest integer such that* $\frac{n}{(1+\varepsilon)^{t_1-1}} \ge \frac{n}{\sqrt{W}}$, *and* t_2 *be the largest integer such that* $\frac{n}{\sqrt{W}}(1 - \varepsilon \cdot t_2) \ge \frac{\varepsilon n}{\sqrt{W}}$. *Note that* $t_1 = \lfloor \log_{1+\varepsilon} \sqrt{W} + 1 \rfloor$ *and* $t_2 = \lfloor \frac{1}{\varepsilon} - 1 \rfloor$. *Define* B_i *such that*

$$B_i = \begin{cases} \frac{n}{(1+\varepsilon)^{i-1}} & \text{if } 1 \le i \le t_1 \\ \frac{n}{\sqrt{W}}(1 - \varepsilon \cdot (i - t_1)) & \text{if } t_1 < i \le t_1 + t_2 \\ 1 & \text{if } i = t := t_1 + t_2 + 1 \end{cases}$$

We then prove that the endpoints B_i's defined in this way form a valid discretization of $[1, n]$ w.r.t. C and $T = (T_1, \ldots, T_W)$.

▶ **Lemma 20.** *Assume that the event stated in Lemma 18 holds. The sequence of endpoints* $(B_1, \ldots, B_t)$ *defined in Definition 19 is a valid discretization of* $[1, n]$ *w.r.t.* C *and error bound* $T_j = \frac{\varepsilon}{8} \cdot \max\left\{\frac{n}{\sqrt{W}}, c_j\right\}$ *on each* $c_j, j \in [W]$.

Now we are ready to exploit the binary search-based algorithm for succinctly approximating the sequence $C = (c_1, \ldots, c_W)$ to estimate the single-linkage clustering cost $\mathrm{cost}(G)$. The algorithm simply performs binary search Algorithm 1 on the estimated array $\widehat{C} = \{\hat{c}_1, \ldots, \hat{c}_W, 0\}$ with search keys $B_1, \ldots, B_t$ as defined in Definition 19, where $\hat{c}_j$'s are estimators of c_j's as defined in Lemma 18. Then we obtain the corresponding indices $\hat{j}_1, \ldots, \hat{j}_t$. For $i \in \{1, \ldots, t-1\}$, let $\hat{J}_i = \{\hat{j}_i, \ldots, \hat{j}_{i+1} - 1\}$. Note that $\hat{J}_1, \ldots, \hat{J}_{t-1}$ form a partition of the index set $[W] = \{1, \ldots, W\}$. For any $j \in \hat{J}_i$, we define $\bar{c}_j = B_i$ as the estimate for c_j and then use $\bar{c}_j$'s to estimate the cost $\mathrm{cost}(G)$.

Algorithm 2 AppCost(G, ε, W, d).

Input: graph G, approximation parameter ε, maximum weight W, average degree d
Output: $\widehat{\mathrm{cost}}(G)$, which estimates $\mathrm{cost}(G)$

1: Set t according to Definition 19 and set parameters $k \leftarrow \sqrt{W}$, $\quad d^{(G)} \leftarrow d \cdot \lceil \frac{4k}{\varepsilon} \rceil$.
2: **for** $i \leftarrow 1$ **to** t **do**
3: Set B_i according to Definition 19.
4: Invoke BinarySearch$(\widehat{C}, 1, W, B_i)$ and get output index $\hat{j}_i$, where $\widehat{C} = (\hat{c}_1, \ldots, \hat{c}_W, 1)$
and each $\hat{c}_j = \min\{\max\{\hat{c}'_j, 1\}, n\}$, and $\hat{c}'_j$ is obtained from Corollary 12 with parameters
G, $H = G_j$, $\varepsilon/8$, $k = \sqrt{W}$, d, $\delta = 1/(4W)$. ▷ Reuse $\hat{c}_j$ if already estimated.
5: **end for**
6: For each $i \in \{1, \ldots, t-1\}$ and each $j \in \hat{J}_i := \{\hat{j}_i, \ldots, \hat{j}_{i+1} - 1\}$, define $\bar{c}_j \leftarrow B_i$.
▷ $\bar{c}_j$'s are defined for the analysis only.
7: Let $\widehat{\mathrm{cost}}(G) \leftarrow \frac{1}{2}\sum_{i=1}^{t-1}(\hat{j}_{i+1} - \hat{j}_i) \cdot (B_i^2 - B_i)$.
8: **return** $\widehat{\mathrm{cost}}(G)$ and the sequence $\{\hat{j}_1, \ldots, \hat{j}_t\}$.

The pseudocode of the algorithm is described in Algorithm 2. It is important to note that we do not need to access all the values in the estimated array $\widehat{C}$. Instead, we only need to access each $\hat{c}_m$ in the search path corresponding to the binary search process. Besides, to ensure consistency in $\widehat{C}$, we reuse $\hat{c}_m$'s stored value, if it was accessed previously. Intuitively, this means that we only need to approximate the number of connected components for

$O(\text{poly}\log W)$ subgraphs G_j. Furthermore, we note that given the values $\hat{j}_i$'s and B_i's , Line 6 and 7 can be implemented in $O(t)$ time, as we only need to use the right-hand side (RHS) of the estimator $\widehat{\text{cost}}(G)$ to estimate $\text{cost}(G)$, and the definitions for $\bar{c}_j$'s are primarily used for the analysis.

We can then prove Theorem 4 by analyzing Algorithm 2 (see full version for details).

3 Proof Sketches for Estimating the Profile Vector

Recall that $\text{cost}(G) = \sum_{k=1}^{n} \text{cost}_k = \sum_{k=1}^{n} \sum_{i=1}^{n-k} w_i$, where cost_k is the cost of the k-SLC and $w_1, \ldots, w_{n-1}$ are the weights of the minimum spanning tree in non-decreasing order. Now we give an algorithm for approximating the SLC profile vector $(\text{cost}_1, \ldots, \text{cost}_n)$ and prove Theorem 3.

In the following, we first derive a formula for the quantity cost_k for each $k = c_j$, where $1 \le j \le W$. Using this formula, we define a corresponding quantity $\overline{\text{cost}}_{B_i}$ for each B_i, where $1 \le i \le t$, as introduced in Definition 19. These quantities constitute our succinct representation. Finally, we define a profile oracle that, given any specified k, outputs the estimator $\widehat{\text{cost}}_k$ of cost_k.

3.1 Formulas of Profile

By definition, we can equivalently express cost_k as follows:

$$\text{cost}_k = \sum_{i=1}^{n-k} w_i = n + \sum_{j=1}^{w_{n-k}-1} c_j - k \cdot w_{n-k} \tag{3}$$

Now we observe that for any given integer $j \in \{1, \ldots, W\}$, we can determine the number of edges in the MST that have weights at most j. This allows us to compute cost_k, where k corresponds to the rank of the first edge in the MST with weight j. We have the following lemma.

▶ **Lemma 21.** *Given any integer $j \in \{1, ..., W\}$ and define $k = c_j$, where c_j is the number of connected components in G_j. Here, G_j is a subgraph of G, and contains all edges with weights at most j. Then we have,*

$$\text{cost}_k = n + \sum_{i=1}^{j-1} c_i - c_j \cdot j.$$

Proof. Let n_j denote the number of edges in the MST with weight j. From our earlier analysis in Theorem 13, we have that $n_j = c_{j-1} - c_j$ for any $1 \le j \le W$, where $c_0 = n$. Then the number of edges in the MST with weights at most j is

$$n_1 + n_2 + \cdots + n_j = (n - c_1) + (c_1 - c_2) + \cdots + (c_{j-1} - c_j) = n - c_j$$

As $k = c_j$, the $(n-k)$-th smallest edge in the MST has weight $w_{n-k} = w_{n-c_j} = j$. Then the statement of lemma follows from Equation (3). ◀

3.2 Algorithm to Estimate Profile

Our main idea of approximating the profile of clustering is to first define $\overline{\text{cost}}_{B_i}$ for non-integer B_i, $1 \le i \le t$, where B_i's are interval endpoints defined in Definition 19. We will make use of Algorithm 2, which performs a binary search using the search key B_i and parameters as described in Algorithm 2, ultimately returning the index $\hat{j}_i$. By substituting j with $\hat{j}_i$ and c_j with $\bar{c}_j = B_i$ in the formula given in Lemma 21, we obtain:

$$\overline{\text{cost}}_{B_i} = n + \sum_{j=1}^{\hat{j}_i - 1} \bar{c}_j - B_i \cdot \hat{j}_i = n + \sum_{k=1}^{i-1} (\hat{j}_{k+1} - \hat{j}_k) \cdot B_k - B_i \cdot \hat{j}_i. \tag{4}$$

This approach gives a t-dimensional vector $(\overline{\text{cost}}_{B_1}, \ldots, \overline{\text{cost}}_{B_t})$ and effectively summarizes the histogram of profile vector. The detailed algorithm is described in Algorithm 3.

Algorithm 3 AppProfile(G, ε, W, d).

1: Get sequence $\{\hat{j}_1, \ldots, \hat{j}_t\}$ from Algorithm 2 AppCost(G, ε, W, d).
2: Set t according to Definition 19.
3: Set $\overline{\text{cost}}_{B_1} \leftarrow 0$.
4: **for** $i \leftarrow 2$ **to** t **do**
5: Set B_i according to Definition 19.
6: Set

$$\overline{\text{cost}}_{B_i} \leftarrow n - B_i \cdot \hat{j}_i + \sum_{k=1}^{i-1} (\hat{j}_{k+1} - \hat{j}_k) \cdot B_k.$$

7: **end for**
8: **return** $(\overline{\text{cost}}_{B_1}, \ldots, \overline{\text{cost}}_{B_t})$.

Given $\{B_1, \ldots, B_t\}$, the vector $(\overline{\text{cost}}_{B_1}, \ldots, \overline{\text{cost}}_{B_t})$, and any specified integer $k \in [1, n]$ such that $B_{i+1} \le k < B_i$, we define $\widehat{\text{cost}}_k = \overline{\text{cost}}_{B_{i+1}}$ to be the estimate for cost_k, as described in Algorithm 4.

Algorithm 4 ProfileOracle$\big(k, \{B_1, \ldots, B_t\}, (\overline{\text{cost}}_{B_1}, \ldots, \overline{\text{cost}}_{B_t})\big)$.

1: Define $B_0 \leftarrow \infty$.
2: Use binary search over $(B_0, B_1, \ldots, B_t)$ to find the index i such that $B_{i+1} \le k < B_i$.
3: **return** $\widehat{\text{cost}}_k \leftarrow \overline{\text{cost}}_{B_{i+1}}$.

We call the vector $(\overline{\text{cost}}_{B_1}, \ldots, \overline{\text{cost}}_{B_t})$ a *succinct representation* of the vector $(\widehat{\text{cost}}_1, \ldots, \widehat{\text{cost}}_n)$. We call Algorithm 4 a *profile oracle*, as it can take any index k as input and answer $\widehat{\text{cost}}_k$.

3.3 Analysis of Algorithms 3 and 4

Now we analyze Algorithms 3 and 4. The following lemma provides an error bound for individual $\widehat{\text{cost}}_k$, demonstrating that the error grows as k increases. Observe that $|\widehat{\text{cost}}_k - \text{cost}_k| = |\overline{\text{cost}}_{B_{i+1}} - \text{cost}_k| \le |\overline{\text{cost}}_{B_{i+1}} - \text{cost}_{c_{j_{i+1}}}| + |\text{cost}_{c_{j_{i+1}}} - \text{cost}_k|$, where j_{i+1} is the smallest index such that $\hat{c}_{j_{i+1}} \le B_{i+1}$. We bound these two terms separately. First, for each interval, B_{i+1} is close to $c_{j_{i+1}}$, so the gap $|\overline{\text{cost}}_{B_{i+1}} - \text{cost}_{c_{j_{i+1}}}|$ is small. Second, by definition, $\text{cost}_k = \sum_{j=1}^{n-k} w_j$, where w_j is the j-th smallest edge weight in the MST. Since both k and $c_{j_{i+1}}$ are constrained by nearby interval endpoints, the gap $|\text{cost}_{c_{j_{i+1}}} - \text{cost}_k|$ is also small. Consequently, the error $|\widehat{\text{cost}}_k - \text{cost}_k|$ is bounded.

▶ **Lemma 22.** *Assume that $\sqrt{W} \le n$ and let $0 < \varepsilon < 1$ be a parameter. With probability at least 3/4, for any integer $k \in \{1, \ldots, n\}$, Algorithm 4 returns an estimate $\widehat{\text{cost}}_k$ for the k-clustering cost, i.e., cost_k, such that*

$$|\widehat{\text{cost}}_k - \text{cost}_k| \le 4\varepsilon \cdot \text{cost}_k + 48\varepsilon \cdot \max\{k, \frac{n}{\sqrt{W}}\}W.$$

Given the succinct representation $(\overline{\text{cost}}_{B_1}, \ldots, \overline{\text{cost}}_{B_t})$*, the running time of the algorithm is* $O(\log(\frac{\log W}{\varepsilon}))$.

Proof sketch. Observe that $|\widehat{\text{cost}}_k - \text{cost}_k| \leq |\overline{\text{cost}}_{B_{i+1}} - \text{cost}_{c_{j_{i+1}}}| + |\text{cost}_{c_{j_{i+1}}} - \text{cost}_k|$, the first term is controlled by applying a variation of Lemma 17 and Lemma 21, and the second by the fact that k and $c_{j_{i+1}}$ lie in neighboring intervals. Together they give $|\widehat{\text{cost}}_k - \text{cost}_k| \leq 4\varepsilon \text{cost}_k + 48\varepsilon \max\{k, n/\sqrt{W}\}W$; the oracle answers in $O(\log(\log W/\varepsilon))$ time. See details in the full version. ◀

Note that when k is very close to n, it becomes impossible to estimate cost_k within a constant factor in sublinear time. For instance, when $k = n - 1$, estimating cost_k amounts to finding the minimum edge weight in the graph – an inherently hard task to perform in sublinear time. Consequently, no algorithm can provide an estimator $\widehat{\text{cost}}_k$ with a constant-factor approximation guarantee for arbitrary values of k under sublinear time constraints. However, in Theorem 3 we prove that the *average* error bound for $\widehat{\text{cost}}_k$ is $\varepsilon \cdot \text{cost}_k$, that is, $\sum_{k=1}^{n} |\widehat{\text{cost}}_k - \text{cost}_k| \leq \varepsilon \sum_{k=1}^{n} \text{cost}_k$.

References

1. Arpit Agarwal, Sanjeev Khanna, Huan Li, and Prathamesh Patil. Sublinear algorithms for hierarchical clustering. *Advances in Neural Information Processing Systems*, 35:3417–3430, 2022.
2. Hyung-Chan An, Mong-Jen Kao, Changyeol Lee, and Mu-Ting Lee. Handling lp-rounding for hierarchical clustering and fitting distances by ultrametrics, 2025. doi:10.48550/arXiv.2504.06700.
3. Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006. doi:10.1109/FOCS.2006.44.
4. Sepehr Assadi, Vaggos Chatziafratis, Vahab Mirrokni, Chen Wang, et al. Hierarchical clustering in graph streams: Single-pass algorithms and space lower bounds. In *Conference on Learning Theory*, pages 4643–4702. PMLR, 2022.
5. MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. Affinity clustering: Hierarchical clustering at scale. *Advances in Neural Information Processing Systems*, 30, 2017.
6. Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 841–854. SIAM, 2017. doi:10.1137/1.9781611974782.53.
7. Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. Hierarchical clustering better than average-linkage. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2291–2304. SIAM, 2019. doi:10.1137/1.9781611975482.139.
8. Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005. doi:10.1137/S0097539702403244.
9. Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)*, 66(4):1–42, 2019. doi:10.1145/3321386.
10. 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 Journal on Computing*, 35(1):91–109, 2005. doi:10.1137/S0097539703435297.
11. Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009. doi:10.1137/060672121.

12 Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 118–127, 2016. doi:10.1145/2897518.2897527.

13 Sanjoy Dasgupta and Philip M Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, 2005. doi:10.1016/J.JCSS.2004.10.006.

14 Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

15 Grzegorz Gluch, Michael Kapralov, Silvio Lattanzi, Aida Mousavifar, and Christian Sohler. Spectral clustering oracles in sublinear time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1598–1617. SIAM, 2021. doi:10.1137/1.9781611976465.97.

16 John C Gower and Gavin JS Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18(1):54–64, 1969.

17 Trevor Hastie, Robert Tibshirani, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

18 Zengfeng Huang and Pan Peng. Dynamic graph stream algorithms in o (n) space. *Algorithmica*, 81:1965–1987, 2019. doi:10.1007/S00453-018-0520-8.

19 Michael Kapralov, Akash Kumar, Silvio Lattanzi, and Aida Mousavifar. Learning hierarchical cluster structure of graphs in sublinear time. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 925–939. SIAM, 2023. doi:10.1137/1.9781611977554.CH36.

20 Michael Kapralov, Akash Kumar, Silvio Lattanzi, Aida Mousavifar, and Weronika Wrzos-Kaminska. Approximating dasgupta cost in sublinear time from a few random seeds, 2025. arXiv:2207.02581.

21 Guolong Lin, Chandrashekhar Nagarajan, Rajmohan Rajaraman, and David P Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal on Computing*, 39(8):3633–3669, 2010. doi:10.1137/070698257.

22 Benjamin Moseley and Joshua R Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. *Journal of Machine Learning Research*, 24(1):1–36, 2023. URL: https://jmlr.org/papers/v24/18-080.html.

23 Pan Peng. Robust clustering oracle and local reconstructor of cluster structure of graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2953–2972. SIAM, 2020. doi:10.1137/1.9781611975994.179.

24 Pan Peng and Christian Sohler. Estimating graph parameters from random order streams. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2449–2466. SIAM, 2018. doi:10.1137/1.9781611975031.157.

25 Ranran Shen and Pan Peng. A sublinear-time spectral clustering oracle with improved preprocessing time. *Advances in Neural Information Processing Systems*, 36, 2023.

26 Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013. doi:10.1137/080744888.

Appendix

Organization of Appendix. This appendix introduces the extension to the similarity setting, together with the algorithms and the main ideas of their analysis. Due to the page limit, the complete proofs, the remaining details, and the other settings and extensions are deferred to the full version.

A Sublinear Algorithms in Similarity Case

Now, we consider single-linkage clustering in similarity graphs. In this case, the agglomerative clustering algorithm merges the two clusters with the highest similarity at each step. The similarity between two clusters is defined as the maximum similarity between any pair of their members. Recall that $w_1^{(s)}, w_2^{(s)}, ..., w_{n-1}^{(s)}$ are the weights of a maximum spanning tree (MaxST) of G in non-increasing order, and the cost of the SLC in the similarity graph is defined as

$$\mathrm{cost}^{(s)}(G) = \sum_{k=1}^{n} \mathrm{cost}_k^{(s)} = \sum_{i=1}^{n-1} (n-i) \cdot w_i^{(s)},$$

where $\mathrm{cost}_k^{(s)} = \sum_{i=1}^{n-k} w_i^{(s)}$ is the cost of the corresponding k-clustering.

A.1 Cost Formula for Similarity-based Clustering

We first derive an equivalent formula for the clustering cost. We let n_j denote the number of edges of weight j in the MaxST. For each weight j, $G_j^{(s)}$ contains all edges of weight $\geq j$. We let $c_j^{(s)}$ be the number of connected components in $G_j^{(s)}$. We observe that $c_1^{(s)} = 1$ and $c_{W+1}^{(s)} = n$, if we assume the whole graph G is connected. Furthermore, it holds that $\sum_{i<j} n_i = c_j^{(s)} - 1$, which gives $n_j = c_{j+1}^{(s)} - c_j^{(s)}$.

▶ **Theorem 23.** *Let G be a connected graph on n vertices, with edge weights from $\{1, \ldots, W\}$. Then*

$$\mathrm{cost}^{(s)}(G) = \sum_{j=1}^{W} \frac{(c_j^{(s)} + n - 1)(n - c_j^{(s)})}{2}$$

We remark that the total weight of the MaxST can be derived in an analogous way: $\mathrm{cost}(\mathrm{MaxST}) = \sum_{j=1}^{W} j \cdot (c_{j+1}^{(s)} - c_j^{(s)}) = 1 \cdot (c_2^{(s)} - c_1^{(s)}) + 2 \cdot (c_3^{(s)} - c_2^{(s)}) + 3 \cdot (c_4^{(s)} - c_3^{(s)}) + \cdots + W \cdot (c_{W+1}^{(s)} - c_W^{(s)}) = nW - \sum_{j=1}^{W} c_j^{(s)} = \sum_{j=1}^{W} (n - c_j^{(s)})$.

A.2 Estimating the Clustering Cost

The formula of $\mathrm{cost}^{(s)}(G)$ in Theorem 23 includes a sum of products $(c_j^{(s)} + n - 1) \cdot (n - c_j^{(s)})$. Our approach is to estimate each product in $\tilde{O}(W d \mathrm{poly}(1/\varepsilon))$ time with small additive error. To do so, we note that by applying Corollary 12, the first term $c_j^{(s)} + n - 1$ can be estimated with an additive error of $O(\varepsilon n)$, which is sufficient for our purpose. For the second term $D_j := n - c_j^{(s)}$, it is also tempting to directly applying the algorithm stated in Corollary 12 to approximate it. However, the resulting additive error will be too large when $c_j^{(s)}$ is large, i.e., close to n.

A.2.1 Approximating D_j

A.2.1.1 High Level Idea of Approximating D_j

To resolve the above issue, we relate D_j to the number of *non-isolated* vertices and the connected components in the graph induced by such vertices in the threshold graph $G_j^{(s)}$.

Specifically, for a given graph G and its subgraph H with the same vertex set, we define H_{nis} to be the subgraph of H induced by all non-isolated vertices w.r.t. H. The number of vertices in H_{nis} is defined as n', and the number of connected components is c'. It is

important to note that the number of connected components $\mathrm{cc}(H)$ of H is exactly $n-n'+c'$. Thus, the quantity $n-\mathrm{cc}(H)=n'-c'$, and one can approximate $n-\mathrm{cc}(H)$ by approximating n' and c' respectively. We remark that the edges of subgraph H are implicitly defined, so we must inspect *all* edges incident to a vertex v in G to find its neighbors in H. Based on this, we give an algorithm for approximating the $n-\mathrm{cc}(H)=n'-c'$ with small enough additive error in Algorithm 5.

Now we give a bit more details of Algorithm 5, we sample vertices to simultaneously estimate the number $\mathrm{cc}(H)$ of connected components, the number c' of connected components in H_{nis} (each of which is a component with at least 2 vertices in H) and the number n' of non-isolated vertices. If there are only a few isolated vertices, we estimate $n-\mathrm{cc}(H)$ using $n-\hat{c}$, where $\hat{c}$ is an estimate of $\mathrm{cc}(H)$. Otherwise, we estimate $n-\mathrm{cc}(H)=n'_j-c'_j$ using $\hat{n'}-\hat{c'}$, where $\hat{n'}$ and $\hat{c'}$ are estimates of n' and c', respectively.

Then to approximate $D_j=n-c_j^{(s)}$, we let H be the threshold graph $G_j^{(s)}$, and let and n'_j, c'_j be the number of vertices, and the number of connected components of H_{nis}, respectively. Then, one can observe that $\mathrm{cc}(H)=c_j^{(s)}=n-n'_j+c'_j$, so $D_j=n'_j-c'_j$, which can be approximated by Algorithm 5 with appropriate input parameters.

A.2.1.2 Analysis of Algorithm 5

We first provide an approximation guarantee of Algorithm 5 in the following lemma.

▶ **Lemma 24.** *Let $0<\varepsilon<1$, and k is an integer greater than 10. Let $r=\lceil\frac{64k}{\varepsilon^2}\rceil$, and $\Gamma=\lceil\frac{4k}{\varepsilon}\rceil$. Suppose the average degree d of graph G is known, and set $d^{(G)}=d\cdot\Gamma=d\cdot\lceil\frac{4k}{\varepsilon}\rceil$. Given access to the graph G and an implicit subgraph $H\subseteq G$ (where H shares the same vertex set as G, and the edges of H are determined by evaluating conditions on the edges of G), Algorithm 5 runs in time $O(\frac{k}{\varepsilon^2}d\log(\frac{k}{\varepsilon}d))$ and outputs a value $\widehat{D}$ such that*

$$\left|\widehat{D}-D\right|\leq\varepsilon\cdot\max\left\{\frac{n}{k},\min\{D,n-D\}\right\}$$

with error probability at least $\frac{3}{4}$, where $D=n-c$ and c is the number of connected components in H.

Proof sketch. Since every component of H_{nis} has at least two vertices, $0\leq c'\leq D\leq n'\leq 2D$. Splitting on the size of D into the regimes $D\leq n/k$, $n/k<D\leq 0.2n$, $0.2n<D\leq 0.6n$, and $D>0.6n$, the algorithm outputs either $\hat{n'}-\hat{c'}$ (when few vertices are non-isolated) or $n-\hat{c}$ (when few are isolated); in each regime a Chebyshev bound on $\hat{n'}$ yields $|\widehat{D}-D|\leq O(\varepsilon)\max\{n/k,\min\{D,n-D\}\}$ with probability $\geq 3/4$. See details are in the full version. ◀

Similar to the distance setting, by applying median trick to Algorithm 5, we have the following corollary.

▶ **Corollary 25.** *For every $1\leq i\leq W$, an appropriately amplified version of Algorithm 5 accepts the same inputs as Algorithm 5, except for an additional success parameter δ; and it outputs $\widehat{D}$ satisfying*

$$\left|\widehat{D}-D\right|\leq\varepsilon\cdot\max\left\{\frac{n}{k},\min\{D,n-D\}\right\}$$

with probability at least $1-\delta$. Here, $D=n-c$ where c is the number of connected components in the subgraph H. The algorithm runs in time $O(\frac{k\log(1/\delta)}{\varepsilon^2}d\log(\frac{k}{\varepsilon}d))$.

Algorithm 5 AppNCCSim$(G, H, \varepsilon, k, d)$.

Input: graph G, subgraph H, approximation parameter ε, threshold k, avg. degree d
Output: an estimate $\widehat{D}$ of $n - c$, where c is the number of connected components in H

Choose $r \leftarrow \lceil 64\frac{k}{\varepsilon^2} \rceil$ vertices $v_1, \ldots, v_r$ uniformly at random.
for $i \leftarrow 1$ **to** r **do**
    Identify neighbors of v_i in H by examining all incident edges in G.
    if v_i is isolated in H **then**
        $x_i \leftarrow 0$.
    else
        $x_i \leftarrow 1$.
    end if
end for
$\hat{n}' \leftarrow \frac{n}{r} \sum_{i=1}^{r} x_i$.
Choose another $r \leftarrow \lceil 64\frac{k}{\varepsilon^2} \rceil$ vertices $u_1, \ldots, u_r$ uniformly at random.
Set threshold $\Gamma \leftarrow \lceil 4\frac{k}{\varepsilon} \rceil$ and $d^{(G)} \leftarrow d \cdot \Gamma$.
for $i \leftarrow 1$ **to** r **do**
    $\alpha_i \leftarrow 0$, $\beta_i \leftarrow 0$.
    Identify neighbors of u_i in H by examining all incident edges in G.
    Let $d_{u_i}^{(G)}$ be the degree of u_i in G.
    if u_i is isolated in H **then**
        $\beta_i \leftarrow 1$.
    else
        (*) Flip a coin.
        if heads and (# vertices visited in H during BFS $< \Gamma$) and (no visited vertex in H has degree in $G > d^{(G)}$ during BFS) **then**
            Resume BFS on H, doubling the number of visited edges in G.
            if this allows BFS on H to complete **then**
                $\alpha_i \leftarrow \beta_i \leftarrow d_{u_i}^{(G)} \cdot 2^{(\#\text{ coin flips})}/(\#\text{ edges visited in } G)$.
            else
                go to (*).
            end if
        end if
    end if
end for
$\hat{c} \leftarrow \frac{n}{r} \sum_{i=1}^{r} \beta_i, \quad \hat{c}' \leftarrow \frac{n}{r} \sum_{i=1}^{r} \alpha_i$.
if $\hat{n}' < 0.5n$ **then**
    return $\widehat{D} \leftarrow \hat{n}' - \hat{c}'$.
else
    return $\widehat{D} \leftarrow n - \hat{c}$.
end if

A.2.2 Adapting Binary Search for Clustering Cost Estimation

Note that $(D_1, \ldots, D_W)$ is a non-increasing array with values in the range $[0, n-1]$. Therefore, we can utilize binary search, similar to the approach used for the distance case, to obtain effective estimators for all elements in this sequence. Specifically, we will apply binary search on the sequence $\widehat{D} = (\widehat{D}_1, \ldots, \widehat{D}_W)$ with some newly defined search keys B_i's. Now we define $\widehat{D}_j$'s in the following lemma.

▶ **Lemma 26.** *For any $1 \le j \le W$, let $\widehat{D}_j = \min\{\max\{\widehat{D}'_j, 0\}, n-1\}$, where $\widehat{D}'_j$ is the output of Corollary 25 with input G, $H = G_j^{(s)}$, $\varepsilon/8$, $k = W$, d, $\delta = 1/(8W)$. Then with probability at least $7/8$, it holds that for* all *$1 \le j \le W$, $\widehat{D}_j \in [0, n-1]$, and*

$$\left|\widehat{D}_j - D_j\right| \le T_j := \frac{\varepsilon}{8} \cdot \max\left\{\frac{n}{W}, \min\{D_j, n - D_j\}\right\}.$$

Throughout the following, we will assume that for all *j, the inequality $\left|\widehat{D}_j - D_j\right| \le T_j$ holds, and $\widehat{D}_j \in [0, n-1]$.* According to Lemma 26, this occurs with probability at least $7/8$.

Now we define the endpoints of intervals which partition $[0, n-1]$.

▶ **Definition 27.** *Let $0 < \varepsilon < 1$, t_1 be the largest integer such that $n - t_1 \frac{\varepsilon n}{W} \ge n - \frac{n}{W}$, t_2 be the largest integer such that $n - (1+\varepsilon)^{t_2} \frac{n}{W} \ge 0.5n$ (i.e. the largest integer such that $\frac{n}{2(1+\varepsilon)^{t_2}} \ge \frac{n}{W}$), and let t_3 be the largest integer such that $\frac{n}{W}(1 - t_3\varepsilon) \ge \frac{\varepsilon n}{W}$. Note that $t_1 = \lfloor \frac{1}{\varepsilon} \rfloor$, $t_2 = \lfloor \log_{1+\varepsilon} \frac{W}{2} \rfloor$, and $t_3 = \lfloor \frac{1-\varepsilon}{\varepsilon} \rfloor$. Then we define B_i as*

$$B_i = \begin{cases} n-1 & \text{if } i = 1 \\ n - i\frac{\varepsilon n}{W} & \text{if } 1 < i \le t_1 \\ n - (1+\varepsilon)^{i-t_1}\frac{n}{W} & \text{if } t_1 < i \le t_1 + t_2 \\ \frac{n}{2(1+\varepsilon)^{i-(t_1+t_2)}} & \text{if } t_1 + t_2 < i \le t_1 + 2t_2 \\ \frac{n}{W}(1 - (i - t_1 - 2t_2)\varepsilon) & \text{if } t_1 + 2t_2 < i \le t_1 + 2t_2 + t_3 \\ 0 & \text{if } i = t := t_1 + 2t_2 + t_3 + 1 \end{cases}$$

We have the following fact.

For each $j \in [W]$, we define the error bound as $T_j = \frac{\varepsilon}{8}\max\{\frac{n}{W}, \min\{D_j, n - D_j\}\}$. Then, we prove that the sequence of B_i's defined above form a valid discretization of $[0, n-1]$ w.r.t. $(D_1, \ldots, D_W)$ and error bound $T = (T_1, \ldots, T_W)$.

▶ **Lemma 28.** *The sequence of endpoints $(B_1, \ldots, B_t)$ defined in Definition 27 is a valid discretization of $[0, n-1]$ w.r.t. $(D_1, \ldots, D_W)$ and error bound $T_j = \frac{\varepsilon}{8}\max\{\frac{n}{W}, \min\{D_j, n - D_j\}\}$.*

Proof sketch. The endpoints of Definition 27 are built from two geometric and two arithmetic pieces meeting at n/W, $0.5n$, and $n - n/W$. In each piece the gap between consecutive endpoints around D_j dominates $T_j = \frac{\varepsilon}{8}\max\{n/W, \min\{D_j, n - D_j\}\}$, which is the valid-discretization condition. The case analysis across the five regimes is deferred to the full version. ◀

A.2.2.1 The Algorithm

Now we are ready to exploit the binary search based algorithm for succinctly approximating the sequence $D = (D_1, \ldots, D_W)$ to estimate the clustering cost $\text{cost}^{(s)}(G)$. The algorithm simply performs binary search Algorithm 1 on the estimated array $\widehat{D} = \{\widehat{D}_1, \ldots, \widehat{D}_W, 0\}$ with search keys $B_1, \ldots, B_t$ as defined in Definition 27, where $\widehat{D}_j$'s are estimators of D_j's as defined in Lemma 26. Then we obtain the corresponding indices $\hat{j}_1, \ldots, \hat{j}_t$. For $i \in \{1, \ldots, t-1\}$, let $\hat{J}_i = \{\hat{j}_i, \ldots, \hat{j}_{i+1} - 1\}$. Note that $\hat{J}_1, \ldots, \hat{J}_{t-1}$ form a partition of the index set $[W] = \{1, \ldots, W\}$. For any $j \in \hat{J}_i$, we define $\overline{D}_j = B_i$ as the estimate for D_j and then use $\overline{D}_j$'s to estimate the cost $\text{cost}^{(s)}(G)$. The algorithm is described in Algorithm 6. Note that we estimate $c_j^{(s)}$ and D_j separately, and then estimate $\text{cost}^{(s)}(G)$ by multiplying the

estimates of $c_j^{(s)}$ and D_j. Furthermore, for the same reason as in distance case, we do not need to access the whole array $\widehat{D}$, but only access $\mathrm{poly}(\log W)$ values $\widehat{D}_i$ in it. We also store previously estimated value of $\widehat{D}_i$ to maintain consistency.

Algorithm 6 AppCostSim(G, ε, d, W).

Input: graph G, approximation parameter ε

Output: $\widehat{\mathrm{cost}^{(s)}}(G)$, which estimates $\mathrm{cost}^{(s)}(G)$

1: Set t according to Definition 27.

2: Get estimates $\{\hat{c}_1^{(s)}, \ldots, \hat{c}_W^{(s)}\}$ from Corollary 25 with parameters

$$G,\ H = G_j,\ \varepsilon,\ k = 1,\ d^{(G)} = d \cdot \left\lceil \frac{4}{\varepsilon} \right\rceil,\ \delta = \frac{1}{8W}.$$

3: **for** $i \leftarrow 1$ **to** t **do**

4: Set B_i according to Definition 27.

5: Invoke BinarySearch$(\widehat{D}, 1, W, B_i)$ and get output index $\hat{j}_i$, where $\widehat{D} = (\widehat{D}_1, \ldots, \widehat{D}_W, 0)$ and each $\widehat{D}_j = \min\{\max\{\widehat{D}'_j, 0\}, n-1\}$, and $\widehat{D}'_j$ is the output of the algorithm in Corollary 25 with input G, $H = G_j^{(s)}$, $\varepsilon/8$, $k = W$, $\delta = 1/(8W)$. ▷ For consistency, reuse stored value of $\widehat{D}_j$ if previously estimated.

6: **end for**

7: For any $j \in \hat{J}_i := \{\hat{j}_i, \ldots, \hat{j}_{i+1} - 1\}$, let $\overline{D}_j = \overline{D}_{\hat{j}_i} = B_i$.

8: Set $\widehat{\mathrm{cost}^{(s)}}(G) \leftarrow \frac{1}{2}\sum_{j=1}^{W}(\hat{c}_j^{(s)} + n - 1) \cdot \overline{D}_j$.

9: **return** $\widehat{\mathrm{cost}^{(s)}}(G)$ and the sequence $(\hat{j}_1, \ldots, \hat{j}_t)$.

Now we first give the following guarantee on the estimates $\overline{D}_j$'s.

▶ **Lemma 29.** *Assume that for* all $j \in [W]$*, the inequality* $\left|\widehat{D}_j - D_j\right| \le T_j$ *holds. Then it holds that for all* $j \in [W]$*,*

$$\left|\overline{D}_j - D_j\right| \le 5\varepsilon \cdot \max\left\{\frac{n}{W}, \min\{D_j, n - D_j\}\right\}.$$

Proof sketch. By Lemma 28 and Lemma 17, $B_{i+2} \le D_j \le B_{i-1}$; substituting $\overline{D}_j = B_i$ and bounding the endpoint gaps in each of the five regimes of D_j gives $|\overline{D}_j - D_j| \le 5\varepsilon \max\{n/W, \min\{D_j, n - D_j\}\}$. The detailed case analysis is deferred to the full version. ◀

A.2.2.2 Analysis of Algorithm 6

Now we give the analysis on the approximation ratio of Algorithm 6 as stated in Theorem 9.

Proof sketch of Theorem 9. Estimating $A_j = c_j^{(s)} + n - 1$ to additive error εn (via Corollary 25) and D_j via Lemma 29, the product error obeys $|\hat{A}_j\overline{D}_j - A_jD_j| \le 31\varepsilon A_jD_j + 20\varepsilon n^2/W$. Summing and using $\mathrm{cost}^{(s)}(G) = \frac{1}{2}\sum_j A_jD_j \ge n^2/4$ gives $(1+\varepsilon)$ after rescaling. The two estimator families cost $\tilde{O}(Wd/\varepsilon^3)$ in total. See details in the full version. ◀