Hierarchical Clustering: Objective Functions and Algorithms

VINCENT COHEN-ADDAD, Sorbonne Université, UPMC Univ Paris 06, CNRS, LIP6, France
VARUN KANADE, University of Oxford, UK
FREDERIK MALLMANN-TRENN, Massachusetts Institute for Technology, USA
CLAIRE MATHIEU, Université Paris Diderot 07, CNRS, IRIF, France

Hierarchical clustering is a recursive partitioning of a dataset into clusters at an increasingly finer granularity. Motivated by the fact that most work on hierarchical clustering was based on providing algorithms, rather than optimizing a specific objective, Dasgupta framed similarity-based hierarchical clustering as a combinatorial optimization problem, where a "good" hierarchical clustering is one that minimizes a particular cost function [23]. He showed that this cost function has certain desirable properties: To achieve optimal cost, disconnected components (namely, dissimilar elements) must be separated at higher levels of the hierarchy, and when the similarity between data elements is identical, all clusterings achieve the same cost.

We take an axiomatic approach to defining "good" objective functions for both similarity- and dissimilarity-based hierarchical clustering. We characterize a set of *admissible* objective functions having the property that when the input admits a "natural" ground-truth hierarchical clustering, the ground-truth clustering has an optimal value. We show that this set includes the objective function introduced by Dasgupta.

Equipped with a suitable objective function, we analyze the performance of practical algorithms, as well as develop better and faster algorithms for hierarchical clustering. We also initiate a beyond worst-case analysis of the complexity of the problem and design algorithms for this scenario.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**; **Approximation algorithms analysis**; **Randomness, geometry and discrete structures**; **Theory and algorithms for application domains**; **Machine learning theory**;

Additional Key Words and Phrases: Hierarchical clustering, stochastic block model, PCA

ACM Reference format:
Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. 2019. Hierarchical Clustering: Objective Functions and Algorithms. *J. ACM* 66, 4, Article 26 (June 2019), 42 pages.
https://doi.org/10.1145/3321386

Most results in this article, with most proofs missing, have appeared in extended abstracts [21] and [20].
Varun Kanade was supported in part by The Alan Turing Institute under the EPSRC grant EP/N510129/1. Frederik Mallmann-Trenn was supported in part by NSF Award Numbers CCF-1461559, CCF-0939370, and CCF-18107.
Authors' addresses: V. Cohen-Addad, LIP6, Sorbonne Université. 4 Place Jussieu, 75005 Paris, France; email: vcohenad@gmail.com; V. Kanade, Department of Computer Science, University of Oxford, Parks Road, Oxford OX2 6QE, UK; email: varunk@cs.ox.ac.uk; F. Mallmann-Trenn, 32 Vassar St, Cambridge, MA 02139, USA; email: mallmann@mit.edu; C. Mathieu, IRIF, Case 7014, Université de Paris Diderot, 75205 Paris Cedex 13, France; email: clairemmathieu@gmail.com.

0004-5411/2019/06-ART26 $15.00
https://doi.org/10.1145/3321386

1 INTRODUCTION

A *hierarchical clustering* is a recursive partitioning of a dataset into successively smaller clusters. The input is a weighted graph whose edge weights represent pairwise similarities or dissimilarities between data points. A hierarchical clustering is represented by a rooted tree where each leaf represents a data point and each internal node represents a cluster containing its descendant leaves. Computing a hierarchical clustering is a fundamental problem in data analysis; it is routinely used to analyze, classify, and pre-process large datasets. A hierarchical clustering provides useful information about data that can be used, e.g., to divide a digital image into distinct regions of different granularities, to identify communities in social networks at various societal levels, or to determine the ancestral tree of life. Developing robust and efficient algorithms for computing hierarchical clusterings is of importance in several research areas, such as machine learning, big-data analysis, and bioinformatics.

Compared to flat partition-based clustering (the problem of dividing the dataset into k parts), hierarchical clustering has received significantly less attention from a theory perspective. Partition-based clustering is typically framed as minimizing a well-defined objective such as k-means, k-medians, and so on, and (approximation) algorithms to optimize these objectives have been a focus of study for at least three decades. However, hierarchical clustering has been studied at a more procedural level in terms of algorithms used in practice. Such algorithms can be broadly classified into two categories, *agglomerative* heuristics that build the candidate cluster tree bottom up, e.g., average-linkage, single-linkage, and complete-linkage, and *divisive* heuristics that build the tree top-down, e.g., bisection k-means, recursive sparsest cut, and so on. Dasgupta [23] identified the lack of a well-defined objective function as one of the reasons why the theoretical study of hierarchical clustering has lagged behind that of partition-based clustering.

Our goal is to provide a comprehensive study of algorithmic approaches to hierarchical clustering. We start by analyzing and defining suitable cost functions and then provide a worst-case analysis of the standard heuristics, then move beyond the worst-case analysis to design efficient algorithms that achieve good approximation guarantees.

Defining a Good Objective Function. What is a "good" output tree for hierarchical clustering? Let us suppose that the edge weights represent similarities (similar data points are connected by edges of high weight).[1] Dasgupta [23] frames hierarchical clustering as a combinatorial optimization problem, where a good output tree is a tree that minimizes some cost function; but which function should that be? Each (binary) tree node is naturally associated to a cut that splits the cluster of its descendant leaves into the cluster of its left subtree on one side and the cluster of its right subtree on the other, and Dasgupta defines the objective to be the sum, over all tree nodes, of the total weight of edges crossing the cut multiplied by the cardinality of the node's cluster. In what sense is this good? Dasgupta argues that it has several attractive properties: (1) If the graph is disconnected, i.e., data items in different connected components have nothing to do with one another, then the hierarchical clustering that minimizes the objective function begins by first pulling apart the connected components from one another; (2) when the input is a (unit-weight) clique, then no particular structure is favored and all binary trees have the same cost; and (3) the cost function also behaves in a desirable manner for data containing a planted partition. Very recently, Moseley and Wang [38] use the "dual" version of Dasgupta's objective in an attempt to explain the behavior of popular heuristics. The dual version also satisfies the properties of Dasgupta's cost function, so which cost function should we use and are there other "interesting" cost functions?

In this article, we take an axiomatic approach to defining a "good" cost function. We remark that in many applications, for example, in phylogenetics, there exists an unknown "ground-truth"

[1] This entire discussion can equivalently be phrased in terms of dissimilarities without changing the essence.

hierarchical clustering—the actual ancestral tree of life—from which the similarities are generated (possibly with noise), and the goal is to infer the underlying ground-truth tree from the available data. In this sense, a cluster tree is good insofar as it is isomorphic to the (unknown) ground-truth cluster tree, and thus a natural condition for a "good" objective function is one such that for inputs that admit a "natural" ground-truth cluster tree, the value of the ground-truth tree is optimal. We provide a formal definition of inputs that admit a ground-truth cluster tree in Section 2.2.

We consider, as potential objective functions, the class of all functions that sum, over all the nodes of the tree, the total weight of edges crossing the associated cut times some function of the cardinalities of the left and right clusters (this includes the class of functions considered by Dasgupta [23]). In Section 3, we characterize the "good" objective functions in this class and call them *admissible* objective functions. We prove that for any ground-truth input, the ground-truth tree has optimal cost (w.r.t. to the objective function) *if and only if* the objective function (1) is symmetric (independent of the left-right order of children), (2) is increasing in the cardinalities of the child clusters, and (3) for (unit-weight) cliques has the same value for all binary trees (Theorem 3.2). Both Dasgupta's and Moseley and Wang's objective functions are admissible in terms of the criteria described above.

Algorithmic Results. The objective functions identified in Section 3 allow us to (1) compare quantitatively the performances of algorithms used in practice and (2) design better and faster approximation algorithms. In several applications, such as in those arising in bioinformatics, the data comes as similarity graphs: The higher the weight of an edge, the higher the similarity between the two elements. In other cases, such as in image processing, the input is a set of points in a Euclidean space where two points are similar if they are near each other. One can see this as a dissimilarity graph: The higher the weight of an edge between two elements (given by, e.g., the Euclidean distance), the higher the dissimilarity. Our approach to defining cost function allows us to define meaningful objective functions for both types of inputs. From an algorithmic perspective, the approaches differ; while optimizing these objective functions is NP-hard,[2] the objective functions behave differently in terms of hardness of approximation. For example, obtaining a constant factor approximation for Dasgupta's cost function for dissimilarity graphs is beyond current techniques while we show that the popular average-linkage heuristics achieves a 2/3-approximation for a meaningful objective function in dissimilarity graphs.

Algorithms for Similarity Graphs. Dasgupta [23] shows that the *recursive ϕ-approximate sparsest cut* algorithm, which recursively splits the input graph using a ϕ-approximation to the sparsest cut problem, outputs a tree whose cost is at most $O(\phi \log n \cdot \text{OPT})$, where OPT is the cost of the optimal tree. Roy and Pokutta [41] recently gave an $O(\log n)$-approximation by providing a linear programming relaxation for the problem and providing a clever rounding technique. Charikar and Chatziafratis [17] showed that the recursive ϕ-sparsest cut (or approximate balanced cut) algorithm of Dasgupta gives an $O(\phi)$-approximation. In Section 4.1, we obtain an independent proof showing that the ϕ-approximate sparsest cut algorithm is an $O(\phi)$-approximation (Theorem 4.1).[3] Our proof is quite different from the proof of Reference [17]. Our proof also shows that using an approximation to the minimum balanced cut would lead to the same result. Combined with the celebrated result of Arora et al. [2], this yields an $O(\sqrt{\log n})$-approximation. All of the results stated here apply to Dasgupta's objective function.

[2]For the objective function proposed in his work, Dasgupta [23] shows that finding a cluster tree that minimizes the cost function is NP-hard. This directly applies to the admissible objective functions for the dissimilarity setting as well. Thus, the focus turns to developing approximation algorithms.

[3]Our analysis shows that the algorithm achieves a 6.75ϕ-approximation and the analysis of Reference [17] yields a 8ϕ-approximation guarantee. This minor difference is of limited impact, since the best approximation guarantee for sparsest cut is $O(\sqrt{\log n})$.

Algorithms for Dissimilarity Graphs. Many of the algorithms commonly used in practice, e.g., linkage-based methods, assume that the input is provided in terms of pairwise dissimilarity (e.g., points that lie in a metric space). As a result, it is of interest to understand how they fare when compared using admissible objective functions for the dissimilarity setting. When the edge weights of the input graph represent dissimilarities, the picture is considerably different from an approximation perspective. For the analogue of Dasgupta's objective function in the dissimilarity setting—which is also admissible, we show that the average-linkage algorithm (see Algorithm 2) achieves a 2/3-approximation (Theorem 4.3). This stands in contrast to other practical heuristic-based algorithms, which may have an approximation guarantee as bad as $\Omega(n^{1/4})$ (Theorem B.3). Thus, using this objective-function-based approach, one can conclude that the average-linkage algorithm is the more robust of the practical algorithms, perhaps explaining its success in practice. We also provide a new, simple algorithm, the locally densest cut algorithm,[4] which we show gives a 2/3-approximation (Theorem 4.6). While dividing recursively using a random cut also gives a 2/3-approximation, we show that the performance of standard heuristics such as single-linkage or bisection 2-center could be arbitrarily bad. Thus it seems that average-linkage and locally densest cut are more robust approaches.

Structured Inputs and Beyond Worst-Case Analysis. The recent work of Roy and Pokutta [41] and Charikar and Chatziafratis [17] have shown that obtaining constant approximation guarantees w.r.t. Dasgupta's cost function for worst-case inputs is beyond current techniques (see Section 1.1). Thus, to obtain better approximation guarantees and algorithms that could have a high impact in practice, it is required to go beyond the worst-case scenario. A natural way to analyze a problem beyond the worst case is to consider a suitable random input model. More precisely, we introduce a random graph model and a semi-random graph model that are based on the notion of a *hierarchical stochastic block model*, which is a natural extension of the stochastic block model. Even in the case of random graphs, the linkage algorithms may perform quite poorly, mainly because ties may be broken unfavorably at early stages, when the clusters are singleton nodes; these choices cannot be easily compensated later on in the algorithm. We thus introduce the LINKAGE++ algorithm, which first uses a *seeding step* using a standard SVD approach to build clusters of a significant size. Then, we show that using these clusters as a starting point, the classic single-linkage approach achieves a $(1 + \varepsilon)$-approximation for the problem (cf. Theorem 5.6). We also consider the semi-random hierarchical stochastic block model and show that by computing recursively an $O(1)$-approximation to the problem of computing a (roughly) balanced cut produces an $O(1)$-approximation to the hierarchical clustering problem. To do so, we harness an algorithm introduced by Makarychev et al. [36] for the Small-Set Expansion problem in a semi-random version of the stochastic block model (cf. Theorem 6.1).

1.1 Related Work

The recent article of Dasgupta [23] served as the starting point of this work. Dasgupta [23] defined an objective function for hierarchical clustering and thus formulated the question of constructing a cluster tree as a combinatorial optimization problem. Dasgupta also showed that the resulting problem is NP-hard and that the recursive ϕ-sparsest-cut algorithm achieves an $O(\phi \log n)$-approximation. Dasgupta's results have been improved in two subsequent articles. Roy and Pokutta [41] wrote an integer program for the hierarchical clustering problem using a combinatorial characterization of the ultrametrics induced by Dasgupta's cost function. They also

[4] We say that a cut (A, B) is locally dense if moving a vertex from A to B or from B to A does not increase the density of the cut. One could similarly define locally sparsest cut.

provided an LP relaxation using spreading metrics and a rounding algorithm based on sphere/region-growing that yields an $O(\log n)$-approximation. Finally, they show that no polynomial size SDP can achieve a constant factor approximation for the problem and that under the Small Set Expansion (SSE) hypothesis, no polynomial-time algorithm can achieve a constant factor approximation.

Charikar and Chatziafratis [17] also gave a proof that the problem is hard to approximate within any constant factor under the Small Set Expansion hypothesis. They also proved that the recursive ϕ-sparsest cut algorithm produces a hierarchical clustering with cost at most $O(\phi\text{OPT})$; their techniques appear to be significantly different from ours. Additionally, they introduce a spreading metric SDP relaxation for the hierarchical clustering problem introduced by Dasgupta that has integrality gap $O(\sqrt{\log n})$ and a spreading metric LP relaxation that yields an $O(\log n)$-approximation to the problem. Recently, Moseley and Wang [38] use the "dual" version of Dasgupta's objective: Their goal is maximizing a fixed quantity minus Dasgupta's cost function. While the NP-hardness of the problem is preserved, hardness of approximation does not hold anymore. Thus, they show that average-linkage "naturally" results in a 1/3-approximation.[5] Very recently, Chatziafratis et al. [19] studied the problem of hierarchical clustering with constraints. In addition, they also give two algorithms that achieve a 2/3-approximation for dissimilarity graphs.

Finally, Charikar et al. [18] study average-linkage in the symmetric "dual" version of Dasgupta's objective as well as in the original dissimilarity setting. They show that the analysis for average-linkage is tight in both settings (with approximation factors of 1/3 and 2/3, respectively). On the positive side, they provide two new algorithms based on semi-definite programming with better approximation factors.

On Hierarchical Clustering More Broadly. There is an extensive literature on hierarchical clustering and its applications. It would be impossible to discuss most of it here; for some applications the reader may refer to, e.g., References [16, 27, 30, 42]. Algorithms for hierarchical clustering have received a lot of attention from a practical perspective. For a definition and overview of *agglomerative* algorithms (such as average-linkage, complete-linkage, and single-linkage) see, e.g., Refeerence [28], and for *divisive algorithms* see, e.g., Reference [43].

Most previous theoretical work on hierarchical clustering aimed at evaluating the cluster tree output by the linkage algorithms using the traditional objective functions for partition-based clustering, e.g., considering k-median or k-means cost of the clusters induced by the top levels of the tree, e.g., References [24, 34, 40]. Previous work also proved that average-linkage can be useful to recover an underlying partition-based clustering when it exists under certain stability conditions [8, 10]. The approach of this article is different: We aim at associating a cost or a value to each hierarchical clustering and finding the best hierarchical clustering with respect to these objective functions.

In Section 3, we take an axiomatic approach toward *objective functions*. Axiomatic approaches toward a *qualitative analysis of algorithms* for clustering have been taken before; for example, the celebrated result of Kleinberg [31] (see also Reference [44]) showed that there is no algorithm satisfying three natural axioms simultaneously. This approach was applied to hierarchical clustering algorithms by Carlsson and Mémoli [15] who showed that in the case of hierarchical clustering one gets a positive result, unlike the impossibility result of Kleinberg. Their focus was on finding an ultrametric (on the data points) that is the closest to the metric (in which the data lies) in terms

[5]It is fairly easy to see that if n is the total number of nodes in the graph and $w(e)$ is the weight associated with edge e, then the quantity $n \cdot \sum_{e \in E} w(e)$ is an upper bound on the total cost of the objective function introduced by Dasgupta. Moseley and Wong simply consider the problem of maximizing $n \cdot \sum_{e \in E} w(e) - \text{cost}_D(T; G)$, where $\text{cost}_D(T; G)$ denote Dasgupta's cost of a cluster tree T on the graph G.

of the Gromov-Hausdorff distance. Our approach is completely different as we focus on defining objective functions and use these for *quantitative analyses of algorithms*.

Our condition for inputs to have a ground-truth cluster tree, and especially their δ-adversarially perturbed versions, is in the same spirit as that of the stability condition of Bilu and Linial [12] or Bilu et al. [11]: The input induces a natural clustering to be recovered whose cost is optimal. It bears some similarities with the "strict separation" condition of Balcan et al. [10], while we do not require the separation to be strict, we do require some additional hierarchical constraints. There are a variety of stability conditions that aim at capturing some of the structure that real-world inputs may exhibit, e.g., References [4, 7, 10, 39]. Some of them induce a condition under which an underlying clustering can be mostly recovered (e.g., References [6, 7, 12] for deterministic conditions and, e.g., References [1, 9, 14, 22, 25] for probabilistic conditions). Imposing other conditions allows one to bypass hardness-of-approximation results for classical clustering objectives (such as k-means), and design efficient approximation algorithms, e.g., References [3, 5, 33]. Eldridge et al. [26] also investigate the question of understanding hierarchical cluster trees for random graphs generated from graphons. Their goal is quite different from ours—they consider the "single-linkage tree" obtained using the graphon as the ground-truth tree and investigate how a cluster tree that has low *merge distortion* with respect to this *single-linkage tree* can be obtained.[6] This is quite different from the approach taken in our work, which is primarily focused on understanding performance with respect to admissible cost functions.

2 PRELIMINARIES

2.1 Notation

An undirected weighted graph $G = (V, E, w)$ is defined by a finite set of vertices V, a set of edges $E \subseteq \{\{u, v\} \mid u, v \in V\}$, and a weight function $w : E \to \mathbb{R}_+$, where $\mathbb{R}_+$ denotes non-negative real numbers. We will only consider graphs with non-negative weights in this article. To simplify notation (and since the graphs are undirected), we let $w(u, v) = w(v, u) = w(\{u, v\})$. When the weights on the edges are not pertinent, we simply denote graphs as $G = (V, E)$. When G is clear from the context, we denote $|V|$ by n and $|E|$ by m. We define $G[U]$ to be the subgraph induced by the nodes of U.

A *cluster tree* or *hierarchical clustering* T for graph G is a rooted binary tree with exactly $|V|$ leaves, each of which is labeled by a distinct vertex $v \in V$.[7] Given a graph $G = (V, E)$ and a cluster tree T for G, for nodes $u, v \in V$ we denote by $\mathrm{LCA}_T(u, v)$ the lowest common ancestor (furthest from the root) of u and v in T.

For any internal node N of T, we denote the subtree of T rooted at N by T_N.[8] Moreover, for any node N of T, define $V(N)$ to be the set of leaves of the subtree rooted at N. Additionally, for any two trees T_1, T_2, define the *union* of T_1, T_2 to be the tree whose root has two children C_1, C_2 such that the subtree rooted at C_1 is T_1 and the subtree rooted at C_2 is T_2.

Finally, given a weighted graph $G = (V, E, w)$, for any set of vertices $A \subseteq V$, let $w(A) = \sum_{a,b \in A} w(a, b)$ and for any set of edges E_0, let $w(E_0) = \sum_{e \in E_0} w(e)$. Finally, for any sets of vertices $A, B \subseteq V$, let $w(A, B) = \sum_{a \in A, b \in B} w(a, b)$.

[6] This is a simplistic characterization of their work. However, a more precise characterization would require introducing a lot of terminology from their article, which is not required in this article.

[7] In general, one can look at trees that are not binary. However, it is common practice to use binary trees in the context of hierarchical trees. Also, for results presented in this article nothing is gained by considering trees that are not binary.

[8] For any tree T, when we refer to a subtree T' (of T) rooted at a node N, we mean the connected subgraph containing all the leaves of T that are descendants of N.

2.2 Ultrametrics

Definition 2.1 (Ultrametric). A metric space (X, d) is an ultrametric if for every $x, y, z \in X$, $d(x, y) \leq \max\{d(x, z), d(y, z)\}$.

Similarity Graphs Generated from Ultrametrics. We say that a weighted graph $G = (V, E, w)$ is a *similarity graph generated from an ultrametric* if there exists an ultrametric (X, d), such that $V \subseteq X$, and for every $x, y \in V, x \neq y$, $e = \{x, y\}$ exists, and $w(e) = f(d(x, y))$, where $f : \mathbb{R}_+ \to \mathbb{R}_+$ is a non-increasing function.[9]

Dissimilarity Graphs Generated from Ultrametrics. We say that a weighted graph $G = (V, E, w)$ is a *dissimilarity graph generated from an ultrametric*, if there exists an ultrametric (X, d), such that $V \subseteq X$, and for every $x, y \in V, x \neq y$, $e = \{x, y\}$ exists, and $w(e) = f(d(x, y))$, where $f : \mathbb{R}_+ \to \mathbb{R}_+$ is a non-decreasing function.

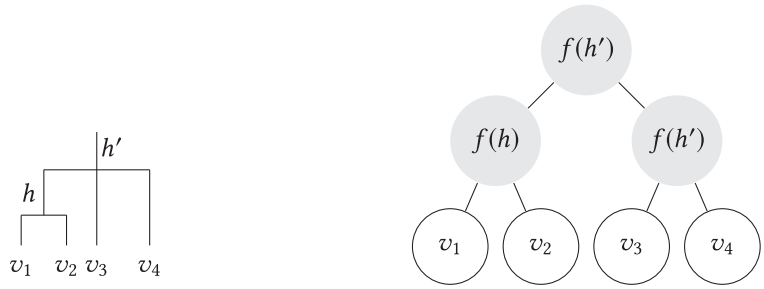
Minimal Generating Ultrametric. For a weighted undirected graph $G = (V, E, w)$ generated from an ultrametric (either similarity or dissimilarity), in general there may be several ultrametrics and corresponding functions f mapping distances in the ultrametric to weights on the edges, that generate the same graph. It is useful to introduce the notion of a minimal ultrametric that generates G. We focus on similarity graphs here; the notion of minimal generating ultrametric for dissimilarity graphs is easily obtained by suitable modifications. Let (X, d) be an ultrametric that generates $G = (V, E, w)$ and f the corresponding function mapping distances to similarities. Then we consider the ultrametric $(V, \widetilde{d})$ defined as follows: (i) $\widetilde{d}(u, u) = 0$ and (ii) for $u \neq v$,

$$\widetilde{d}(u, v) = \widetilde{d}(v, u) = \max_{u', v'} \{d(u', v') \mid f(d(u', v')) = f(d(u, v))\}. \tag{1}$$

It remains to be seen that $(V, \widetilde{d})$ is indeed an ultrametric. First, notice that by definition, $\widetilde{d}(u, v) \geq d(u, v)$ and hence clearly $\widetilde{d}(u, v) = 0$ if and only if $u = v$ as d is the distance in an ultrametric. The fact that $\widetilde{d}$ is symmetric is immediate from the definition. The only part remaining to check is the so called *isosceles triangles with longer equal sides* conditions—the ultrametric requirement that for any u, v, w, $d(u, v) \leq \max\{d(u, w), d(v, w)\}$ implies that all triangles are isosceles and the two sides that are equal are at least as large as the third side. Let $u, v, w \in V$, and assume without loss of generality that according to the distance d of (V, d), $d(u, w) = d(v, w) \geq d(u, v)$. From Equation (1) it is clear that $\widetilde{d}(u, w) = \widetilde{d}(v, w) \geq d(u, w)$. Also, from Equation (1) and the non-increasing nature of f it is clear that if $d(u, v) \leq d(u', v')$, then $\widetilde{d}(u, v) \leq \widetilde{d}(u', v')$. Thence, $(V, \widetilde{d})$ is an ultrametric. The advantage of considering the minimal ultrametric is the following: If $\mathcal{D} = \{\widetilde{d}(u, v) \mid u, v \in V, u \neq v\}$ and $\mathcal{W} = \{w(u, v) \mid u, v \in V, u \neq v\}$, then the restriction of f from $\mathcal{D} \to \mathcal{W}$ is actually a bijection. This allows the notion of a generating tree to be defined in terms of distances in the ultrametric or weights, without any ambiguity. Applying an analogous definition and reasoning yields a similar notion for the dissimilarity case.

Definition 2.2 (Generating Tree). Let $G = (V, E, w)$ be a graph generated by a minimal ultrametric (V, d) (either a similarity or dissimilarity graph). Let T be a rooted binary tree with $|V|$ leaves and $|V| - 1$ internal nodes; let $\mathcal{N}$ denote the internal nodes and L the set of leaves of T and let $\sigma : L \to V$ denote a bijection between the leaves of T and nodes of V. We say that T is a generating tree for G, if there exists a weight function $W : \mathcal{N} \to \mathbb{R}_+$, such that for $N_1, N_2 \in \mathcal{N}$, if N_1 appears on the path from N_2 to the root, $W(N_1) \leq W(N_2)$ ($W(N_1) \geq W(N_2)$ in the dissimilarity case). Moreover for every $x, y \in V$, $w(\{x, y\}) = W(LCA_T(\sigma^{-1}(x), \sigma^{-1}(y)))$.

[9] In some cases, we will say that $e = \{x, y\} \notin E$, if $w(e) = 0$. This is fine as long as $f(d(x, y)) = 0$.



(a) Dendrogram on 4 nodes. (b) Generating tree equivalent to dendrogram

Fig. 1. Dendrogram and equivalent generating tree.

The notion of a generating tree defined above more or less corresponds to what is referred to as a *dendrogram* in the machine-learning literature, e.g., Reference [15]. More formally, a dendrogram is a rooted tree (not necessarily binary), where the leaves represent the datapoints. Every internal node in the tree has associated with it a height function h, which is the distance between any pairs of datapoints for which it is the least common ancestor. It is a well-known fact that a set of points in an ultrametric can be represented using a dendrogram, e.g., Reference [15]. A dendrogram can easily be modified to obtain a generating tree in the sense of Definition 2.2: an internal node with k children is replace by an arbitrary binary tree with k leaves and the children of the nodes in the dendrogram are attached to these k leaves. The height h of this node is used to give the weight $W = f(h)$ to all the $k - 1$ internal nodes added when replacing this node. Figure 1 shows this transformation.

Ground-Truth Inputs.

Definition 2.3 (Ground-Truth Input). We say that a graph G is a *ground-truth input* if it is a similarity or dissimilarity graph generated from an ultrametric. Equivalently, there exists a tree T that is generating for G.

Motivation. We briefly describe the motivation for defining graphs generated from an ultrametric as ground-truth inputs. We will focus the discussion on similarity graphs, though essentially the same logic holds for dissimilarity graphs. As described earlier, there is a natural notion of a *generating tree* associated with graphs generated from ultrametrics. This tree itself can be viewed as a cluster tree. The clusters obtained using the generating tree have the property that any two nodes in the same cluster are at least as similar to each other as they are to points outside this cluster; and this holds at every level of granularity. Furthermore, as observed by Carlsson and Mémoli [15], many practical hierarchical clustering algorithms, such as the linkage-based algorithms, actually output a dendrogram equipped with a height function that corresponds to an ultrametric embedding of the data. While their work focuses on algorithms that find embeddings in ultrametrics, our work focuses on finding cluster trees. We remark that these problems are related but also quite different.

Furthermore, our results show that the linkage algorithms (and some other practical algorithms), recover a generating tree when given as input graphs that are generated from an ultrametric. Finally, we remark that relaxing the notion further leads to instances where it is hard to define a "natural" ground-truth tree. Consider a similarity graph generated by a *tree-metric* rather than an ultrametric, where the tree is the caterpillar graph on 5 nodes (see Figure 2(a)). Then, it is hard to argue that the tree shown in Figure 2(b) is not a more suitable cluster tree. For instance, D and E are more similar to each other than D is to B or A. In fact, it is not hard to show that by choosing a

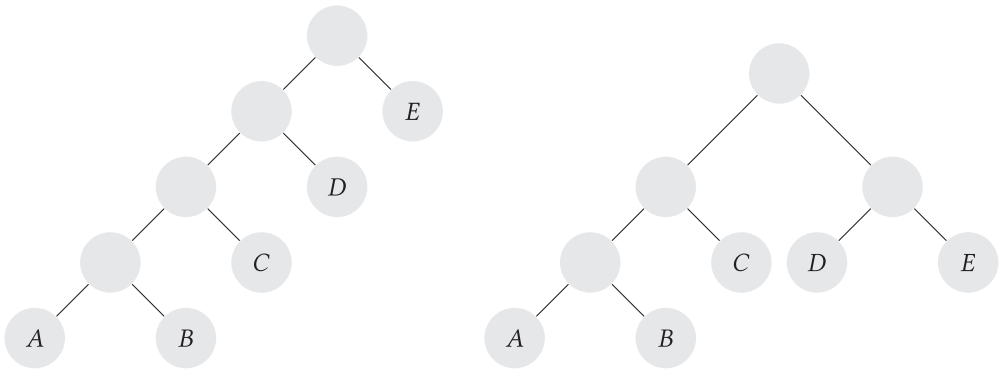


Fig. 2. (a) Caterpillar tree on five nodes with unit-weight edges used to define a tree metric on nodes A, B, C, D, E. (b) A hierarchical clustering tree for A, B, C, D, E.

suitable function f mapping distances from this tree metric to similarities, Dasgupta's objective function is minimized by the tree shown in Figure 2(b) rather than the "generating" tree in Figure 2(a).

3 QUANTIFYING OUTPUT VALUE: AN AXIOMATIC APPROACH

3.1 Admissible Cost Functions

Let us focus on the similarity case; in this case, we use *cost* and *objective* interchangeably. Let $G = (V, E, w)$ be an undirected weighted graph and let T be a cluster tree for graph G. We consider cost functions for cluster trees that capture the quality of the hierarchical clustering produced by T.

The Axiom. A natural property we would like the cost function to satisfy is that a cluster tree T has minimum cost if and only if T is a generating tree for G. Indeed, the objective function can then be used to indicate whether a given tree is generating and so whether it is an underlying ground-truth hierarchical clustering. Hence, the objective function acts as a "guide" for finding the correct hierarchical classification. Note that there may be multiple trees that are generating for the same graph. For example, if $G = (V, E, w)$ is a clique with every edge having the same weight, then every tree is a generating tree. In these cases, all generating trees are *valid* ground-truth hierarchical clusterings.

Following the recent work of Dasgupta [23], we adopt an approach in which a cost is assigned to each internal node of the tree T that corresponds to the quality of the split at that node and restrict the search space for such cost functions. For an internal node N in a clustering tree T, let $A, B \subseteq V$ be the leaves of the subtrees rooted at the left and right child of N, respectively. We define the cost Γ of the tree T as the sum of the cost at every internal node N in the tree, and at an individual node N we consider cost functions γ of the form

$$\Gamma(T) = \sum_{N} \gamma(N), \tag{2}$$

$$\gamma(N) = \left(\sum_{x \in A,\, y \in B} w(x, y) \right) \cdot g(|A|, |B|). \tag{3}$$

We remark that Dasgupta [23] defined $g(a, b) = a + b$.

Definition 3.1 (Admissible Cost Function). We say that a cost function γ of the form of Equations (2) and (3) is *admissible* if it satisfies the condition that for all similarity graphs $G = (V, E, w)$ generated from a minimal ultrametric (V, d), a cluster tree T for G achieves the minimum cost if and only if it is a generating tree for G.

Remark 1. Analogously, for the dissimilarity setting we define *admissible value functions* to be the functions of the form of Equations (2) and (3) that satisfy the following: For all dissimilarity graph G generated from a minimal ultrametric (V, d), a cluster tree T for G achieves the maximum value if and only if it is a generating tree for G.

Remark 2. The RHS of Equation (3) has linear dependence on the weight of the cut (A, B) in the subgraph of G induced by the vertex set $A \cup B$ as well as on an arbitrary function of the number of leaves in the subtrees of the left and right child of the internal node creating the cut (A, B). For the purpose of hierarchical clustering this form is fairly natural and indeed includes the specific cost function introduced by Dasgupta [23]. We could define the notion of admissibility for other forms of the cost function similarly and it would be of interest to understand whether they have properties that are desirable from the point of view of hierarchical clustering.

3.2 Characterizing Admissible Cost Functions

In this section, we give an almost complete characterization of admissible cost functions of the form of Equation (3). The following theorem shows that cost functions of this form are admissible if and only if they satisfy three conditions: all cliques have the same cost, symmetry and monotonicity.

Theorem 3.2. *Let γ be a cost function of the form of Equation (3) and let g be the corresponding function used to define γ. Then γ is admissible if and only if it satisfies the following three conditions:*

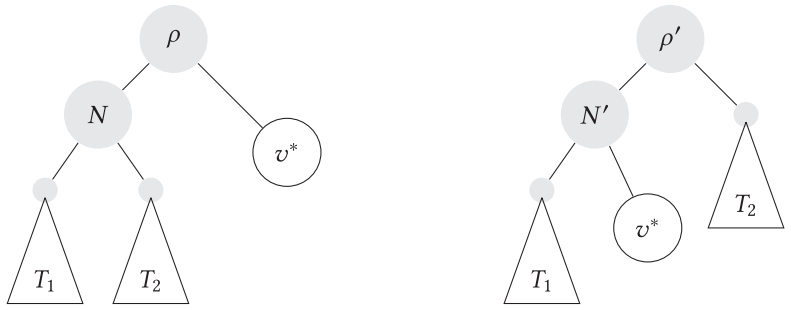
(1) Let $G = (V, E, w)$ be a clique, i.e., for every $x, y \in V$, $e = \{x, y\} \in E$ and $w(e) = 1$ for every $e \in E$. Then the cost $\Gamma(T)$ for every cluster tree T of G is identical.
(2) For every $n_1, n_2 \in \mathbb{N}$, $g(n_1, n_2) = g(n_2, n_1)$.
(3) For every $n_1, n_2 \in \mathbb{N}$, $g(n_1 + 1, n_2) > g(n_1, n_2)$.

Proof. We first prove the only if part and then the if part.
Only If Part: Suppose that γ is indeed an admissible cost function. We prove that all three conditions must be satisfied by γ.

1. All cliques have same cost. We observe that a clique $G = (V, E, w)$ can be generated from an ultrametric. Indeed, let $X = V$ and let $d(u, v) = d(v, u) = 1$ for every $u, v \in X$ such that $u \neq v$ and $d(u, u) = 0$. Clearly, for $f : \mathbb{R}_+ \to \mathbb{R}_+$ that is non-increasing and satisfying $f(1) = 1$, (V, d) is a minimal ultrametric generating G.

Let T be any binary rooted tree with leaves labeled by V, i.e., a cluster tree for graph G. For any internal node N of T define $W(N) = 1$ as the weight function. This satisfies the definition of generating tree (Definition 2.2). Thus, every cluster tree T for G is generating and hence, by the definition of admissibility all of them must be optimal, i.e., they all must have exactly the same cost.

2. $g(n_1, n_2) = g(n_2, n_1)$. This part follows more or less directly from the previous part. Let G be a clique on $n_1 + n_2$ nodes. Let T be any cluster tree for G, with subtrees T_1 and T_2 rooted at the left and right child of the root respectively, such that T_1 contains n_1 leaves and T_2 contains n_2 leaves. The number of edges, and hence the total weight of the edges, crossing the cut induced by the root node of T is $n_1 \cdot n_2$. Let $\widetilde{T}$ be a tree obtained by making T_2 be rooted at the left child of the root and T_1 at the right child. Clearly $\widetilde{T}$ is also a cluster tree for G and induces the same



(a) Tree T (b) Tree T'

Fig. 3. Trees T and T' used to show monotonicity of g.

cut at the root node, hence using the property that all cliques have the same cost, $\Gamma(T) = \Gamma(\widetilde{T})$. But $\Gamma(T) = n_1 \cdot n_2 \cdot g(n_1, n_2) + \Gamma(T_1) + \Gamma(T_2)$ and $\Gamma(\widetilde{T}) = n_1 \cdot n_2 \cdot g(n_2, n_1) + \Gamma(T_1) + \Gamma(T_2)$. Thence, $g(n_1, n_2) = g(n_2, n_1)$.

3. $g(n_1 + 1, n_2) > g(n_1, n_2)$. Consider a graph on $n_1 + n_2 + 1$ nodes generated from an ultrametric as follows. Let $V_1 = \{v_1, \ldots, v_{n_1}\}$, $V_2 = \{v'_1, \ldots, v'_{n_2}\}$ and consider the ultrametric $(V_1 \cup V_2 \cup \{v^*\}, d)$ defined by $d(x, y) = 1$ if $x \neq y$ and $x, y \in V_1$ or $x, y \in V_2$, $d(x, y) = 2$ if $x \neq y$ and $x \in V_1, y \in V_2$ or $x \in V_2, y \in V_1$, $d(v^*, x) = d(x, v^*) = 3$ for $x \in V_1 \cup V_2$, and $d(u, u) = 0$ for $u \in V_1 \cup V_2 \cup \{v^*\}$. It can be checked easily by enumeration that this is indeed an ultrametric. Furthermore, if $f : \mathbb{R}_+ \to \mathbb{R}_+$ is non-increasing and satisfies $f(1) = 2$, $f(2) = 1$ and $f(3) = 0$, i.e., $w(\{u, v\}) = 2$ if u and v are both either in V_1 or V_2, $w(\{u, v\}) = 1$ if $u \in V_1$ and $v \in V_2$ or the other way around, and $w(\{v^*, u\}) = 0$ for $u \in V_1 \cup V_2$, then $(V_1 \cup V_2, \{v^*\}, d)$ is a minimal ultrametric generating G.

Now consider two possible cluster trees defined as follows: Let T_1 be an arbitrary tree on nodes V_1, T_2 and arbitrary tree on nodes V_2. T is obtained by first joining T_1 and T_2 using internal node N and making this the left subtree of the root node ρ and the right subtree of the root node is just the singleton node v^*. T' is obtained by first creating a tree by joining T_1 and the singleton node v^* using internal node N', this is the left subtree of the root node ρ' and T_2 is the right subtree of the root node. (See Figure 3(a) and (b).)

Now it can be checked that T is generating by defining the following weight function. For every internal node M of T_1, let $W(M) = 1$, similarly for every internal node M of T_2, let $W(M) = 1$, define $W(N) = 2$ and $W(\rho) = 3$. Now, we claim that T' cannot be a generating tree. This follows from the fact that for a node $u \in V_1, v \in V_2$, the root node $\rho' = \mathrm{LCA}_{T'}(u, v)$, but it is also the case that $\rho' = \mathrm{LCA}_{T'}(v^*, v)$ (recall that $\mathrm{LCA}_{T'}(u, v)$ denotes the lowest common ancestor of u, v in T', as defined in Section 2). Thus, it cannot possibly be the case that $W(\rho) = w(\{u, v\})$ and $W(\rho) = w(\{v^*, v\})$ as $w(\{u, v\}) \neq w(\{v^*, v\})$. By definition of admissibility, it follows that $\Gamma(T) < \Gamma(T')$, but $\Gamma(T) = \Gamma(T_1) + \Gamma(T_2) + n_1 \cdot n_2 \cdot g(n_1, n_2)$. The last term arises from the cut at node N; the root makes no contribution as the cut at the root node ρ has weight 0. However, $\Gamma(T') = \Gamma(T_1) + \Gamma(T_2) + n_1 \cdot n_2 \cdot g(n_1 + 1, n_2)$. There is no cost at the node N', since the cut has size 0; however, at the root node the cost is now $n_1 \cdot n_2 \cdot g(n_1 + 1, n_2)$ as the left subtree at the root contains $n_1 + 1$ nodes. It follows that $g(n_1 + 1, n_2) > g(n_1, n_2)$.

If Part: For the other direction, we first use the following observation. By condition 1 in the statement of the theorem, every clique on n nodes has the same cost irrespective of the tree used for hierarchical clustering; let $\kappa(n)$ denote said cost. Let $n_1, n_2 \geq 1$, then we have

$$n_1 \cdot n_2 \cdot g(n_1, n_2) = \kappa(n_1 + n_2) - \kappa(n_1) - \kappa(n_2). \tag{4}$$

We will complete the proof by induction on $|V|$. The minimum number of nodes required to have a cluster tree with at least one internal node is 2. Suppose $|V| = 2$, then there is a unique (up to interchanging left and right children) cluster tree; this tree is also generating and hence by definition any cost function is admissible. Thus, the base case is covered rather easily.

Now, consider a graph $G = (V, E, w)$ with $|V| = n > 2$. Let T^* be a tree that is generating. Suppose that T is any other tree. Let ρ^* and ρ be the root nodes of the trees, respectively. Let V_L^* and V_R^* be the nodes on the left subtree and right subtree of ρ^*; similarly V_L and V_R in the case of ρ. Let $A = V_L^* \cap V_L$, $B = V_L^* \cap V_R$, $C = V_R^* \cap V_L$, $D = V_R^* \cap V_R$. Let a, b, c, and d denote the sizes of A, B, C, and D, respectively.

We will consider the case when all of $a, b, c, d > 0$; the proof is similar and simpler in case some of them are 0. Let $\widetilde{T}$ be a tree with root $\widetilde{\rho}$ that has the following structure: Both children of the root are internal nodes, all of A appears as leaves in the left subtree of the left child of the root, B as leaves in the right subtree of the left child of the root, C as leaves in the left subtree of the right child of the root and D as leaves in the right subtree of the right child of the root. We assume that all four subtrees for the sets A, B, C, D are generating and hence by induction optimal. We claim that the cost of $\widetilde{T}$ is at least as much as the cost of T^*. To see this note that $V_L^* = A \cup B$. Thus, the left subtree of ρ^* is optimal for the set V_L^* (by induction), whereas that of $\widetilde{\rho}$ may or may not be. Similarly, for all the nodes in V_R^*. The only other thing left to account for is the cost at the root. But since ρ^* and $\widetilde{\rho}$ induce exactly the same cut on V, the cost at the root is the same. Thus, $\Gamma(\widetilde{T}) \geq \Gamma(T^*)$. Furthermore, equality holds if and only if $\widetilde{T}$ is also generating for G.

Let W^* denote the weight function for the generating tree T^* such that for all $u, v \in V$, $W^*(\mathrm{LCA}_{T^*}(u, v)) = w(\{u, v\})$. Let ρ_L^* and ρ_R^* denote the left and right children of the root ρ^* of T^*. For all $u_a \in A, u_b \in B$, $w(\{u_a, u_b\}) \geq W^*(\rho_L^*)$. Let

$$x = \frac{1}{ab} \sum_{u_a \in A,\, u_b \in B} w(\{u_a, u_b\})$$

denote the average weight of the edges going between A and B; it follows that $x \geq W^*(\rho_L^*)$. Similarly, for all $u_c \in C, u_d \in D$, $w(\{u_c, u_d\}) \geq W^*(\rho_R^*)$. Let

$$y = \frac{1}{cd} \sum_{u_c \in C,\, u_d \in D} w(\{u_c, u_d\})$$

denote the average weight of the edges going between C and D; it follows that $y \geq W^*(\rho_R^*)$. Finally, for every $u \in A \cup B, u' \in C \cup D$, $w(\{u, u'\}) = W^*(\rho^*)$; denote this common value by z. By the definition of generating tree, we know that $x \geq z$ and $y \geq z$.

Now consider the tree T. Let T_L and T_R denote the left and right subtrees of ρ. By induction, it must be that T_L splits A and C as the first cut (or at least that's one possible tree, if multiple cuts exist); similarly, T_R first cuts B and D. Both T and $\widetilde{T}$ have subtrees containing only nodes from A, B, C, and D. The costs for these subtrees are identical in both cases (by induction). Thus, we have

$$\begin{aligned}
\Gamma(T) - \Gamma(\widetilde{T}) &= zac \cdot g(a,c) + zbd \cdot g(b,d) + (xab + ycd + z(ad + bc)) \cdot g(a + c, b + d) \\
&\quad - xab \cdot g(a,b) + y \cdot cdg(c,d) - z(a + b)(c + d) \cdot g(a + b, c + d) \\
&= (x - z)ab(g(a + c, b + d) - g(a,b)) + (y - z)cd(g(a + c, b + d) - g(c,d)) \\
&\quad + z((a + c)(b + d) \cdot g(a + c, b + d) + ac \cdot g(a,c) + bd \cdot g(b,d)) \\
&\quad - z((a + b)(c + d) \cdot g(a + b, c + d) + ab \cdot g(a,b) + cd \cdot g(c,d)).
\end{aligned}$$

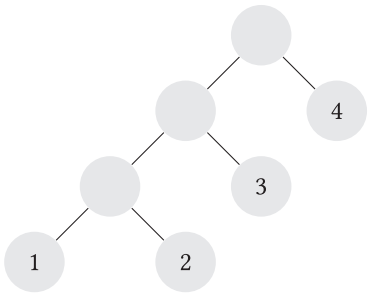


Fig. 4. The caterpillar cluster tree for a clique with 4 nodes.

Using Equation (4), we get that the last two expressions above both evaluate to $z(\kappa(a+b+c+d)-\kappa(a)-\kappa(b)-\kappa(c)-\kappa(d))$ but have opposite signs. Thus, we get

$$\Gamma(T)-\Gamma(\widetilde{T})=(x-z)ab(g(a+c,b+d)-g(a,b))+(y-z)cd(g(a+c,b+d)-g(c,d)).$$

It is clear that the above expression is always non-negative and is 0 if and only if $x=z$ and $y=z$. If it is the latter case and it is also the case that $\Gamma(\widetilde{T})=\Gamma(T^*)$, then it must actually be the case that T is a generating tree. □

3.2.1 Characterizing g that Satisfy Conditions of Theorem 3.2. Theorem 3.2 give necessary and sufficient conditions on g for cost functions of the form of Equation (3) to be admissible. However, it leaves open the question of the existence of functions satisfying the criteria and also characterizing the functions g themselves. The fact that such functions exist already follows from the work of Dasgupta [23], who showed that if $g(n_1,n_2)=n_1+n_2$, then all cliques have the same cost. Clearly, g is monotone and symmetric and thus satisfies the condition of Theorem 3.2.

To give a more complete characterization, we define g as follows: Suppose $g(\cdot,\cdot)$ is symmetric, we define $g(n,1)$ for all $n\geq 1$ so that $g(n,1)/(n+1)$ is non-decreasing.[10] We consider a particular cluster tree for a clique that is defined using a caterpillar graph, i.e., a cluster tree where the right child of any internal node is a leaf labeled by one of the nodes of G and the left child is another internal node, except at the very bottom. Figure 4 shows a caterpillar cluster tree for a clique on four nodes. The cost of the clique on n nodes, say, $\kappa(n)$, using this cluster tree is given by

$$\kappa(n)=\sum_{i=0}^{n-1} i\cdot g(i,1).$$

Now we enforce the condition that all cliques have the same cost by defining $g(n_1,n_2)$ for $n_1,n_2>1$ suitably, in particular,

$$g(n_1,n_2)=\frac{\kappa(n_1+n_2)-\kappa(n_1)-\kappa(n_2)}{n_1\cdot n_2}. \tag{5}$$

Thus it only remains to be shown that g is strictly increasing. We show that for $n_2\leq n_1$, $g(n_1+1,n_2)>g(n_1,n_2)$. To show this it suffices to show that

$$n_1(\kappa(n_1+n_2+1)-\kappa(n_1+1)-\kappa(n_2))-(n_1+1)(\kappa(n_1+n_2)-\kappa(n_1)-\kappa(n_2))>0.$$

[10]The function proposed by Dasgupta [23] is $g(n,1)=n+1$, so this ratio is always 1.

Thus, consider

$$
\begin{aligned}
&n_1(\kappa(n_1+n_2+1)-\kappa(n_1+1)-\kappa(n_2))-(n_1+1)(\kappa(n_1+n_2)-\kappa(n_1)-\kappa(n_2))\\
&=n_1(\kappa(n_1+n_2+1)-\kappa(n_1+n_2)-\kappa(1)-\kappa(n_1+1)+\kappa(n_1)+\kappa(1))-(\kappa(n_1+n_2)-\kappa(n_1)-\kappa(n_2))\\
&=n_1(n_1+n_2)g(n_1+n_2,1)-n_1^2g(n_1,1)-(\kappa(n_1+n_2)-\kappa(n_1)-\kappa(n_2))\\
&\geq n_1(n_1+n_2)g(n_1+n_2,1)-n_1^2g(n_1,1)-\sum_{i=n_1}^{n_1+n_2-1} i\cdot g(i,1)\\
&\geq \frac{g(n_1+n_2,1)}{n_1+n_2+1}\cdot\left(n_1(n_1+n_2)(n_1+n_2+1)-n_1^2(n_1+1)-\sum_{i=n_1}^{n_1+n_2-1} i(i+1)\right)>0.
\end{aligned}
$$

Above we used the fact that $g(n,1)/(n+1)$ is non-decreasing in n and some elementary calculations. This shows that the objective function proposed by Dasgupta [23] is by no means unique. Only in the last step do we get an inequality where we use the condition that $g(n,1)/(n+1)$ is increasing. Whether this requirement can be relaxed further is also an interesting direction.

3.2.2 Characterizing Objective Functions for Dissimilarity Graphs. When the weights of the edges represent dissimilarities instead of similarities, one can consider objective functions of the same form as Equation (3). As mentioned in Remark 1, the difference in this case is that the goal is to maximize the objective function and hence the definition of admissibility now requires that generating trees have a value of the objective that is strictly larger than any tree that is not generating.

The characterization of admissible objective functions as given in Theorem 3.2 for the similarity case continues to hold in the case of dissimilarities. The proof follows in the same manner by appropriately switching the direction of the inequalities when required.

4 WORST-CASE

We now provide approximation algorithms for Dasgupta's objective function for the similarity setting (Section 4.1) and its analogue for the dissimilarity setting (Section 4.2), i.e., we treat each edge weight as a measure of dissimilarity and consider the problem of maximizing the cost defined by Dasgupta. Note that Dasgupta showed that the problem of finding an optimal solution w.r.t. to his cost function for both the similarity and dissimilarity setting is NP-hard [23]. In addition, Charikar and Chatziafratis [17] and Roy and Pokutta [41] showed that the similarity version of the problem does not admit an $O(1)$-approximation algorithm assuming the SSE-hypothesis.

4.1 Similarity-Based Inputs: Approximation Algorithms

4.1.1 Analysis of the Recursive Sparsest-cut Algorithm w.r.t. Dasgupta's Cost Function. In this section, we analyze the recursive ϕ-sparsest-cut algorithm (see Algorithm 1) that was described previously in Referencee [23]. Recall the cost function introduced by Dasgupta [23]: The cost of tree T is $\text{cost}(T)=\sum_{N\in T}\text{cost}(N)$, where for each node N of T with children N_1, N_2, $\text{cost}(N)=w(V(N_1),V(N_2))\cdot V(N)$. The goal is to find a tree T^* minimizing $\text{cost}(T^*)$. We show that the ϕ-sparsest-cut algorithm achieves a 6.75ϕ-approximation. Charikar and Chatziafratis [17] proved an 8ϕ approximation for Dasgupta's function.

The ϕ-sparsest-cut algorithm (Algorithm 1) constructs a binary tree top-down by recursively finding cuts using a ϕ-approximate sparsest cut algorithm, where the *sparsest-cut* problem asks for a set A minimizing the *sparsity* $w(A,V\setminus A)/(|A||V\setminus A|)$ of the cut $(A,V\setminus A)$.

Theorem 4.1. *For any graph $G=(V,E)$, and weight function $w:E\to\mathbb{R}_+$, the ϕ-sparsest-cut algorithm (Algorithm 1) outputs a solution of cost at most $\frac{27}{4}\phi OPT$.*

ALGORITHM 1: Recursive ϕ-Sparsest-Cut Algorithm for Hierarchical Clustering

1: **Input**: An edge weighted graph $G = (V, E, w)$.
2: $\{A, V \setminus A\} \leftarrow$ cut with sparsity $\leq \phi \cdot \min_{S \subset V} w(S, V \setminus S)/(|S||V \setminus S|)$
3: Recurse on $G[A]$ and on $G[V \setminus A]$ to obtain trees T_A and $T_{V \setminus A}$
4:
5: **return** The tree whose root has two children, T_A and $T_{V \setminus A}$.

Proof. Let $G = (V, E)$ be the input graph and n denote the total number of vertices of G. Let T denote the tree output by the algorithm and T^* be any arbitrary tree. We will prove that $\text{cost}(T) \leq \frac{27}{4}\phi\text{cost}(T^*)$.

Recall that for an arbitrary tree T_0 and node N of T_0, the vertex corresponding to the leaves of the subtree rooted at N is denoted by $V(N)$. Consider the node N_0 of T^* that is the first node reached by the walk from the root that always goes to the child tree with the higher number of leaves, stopping when the subtree of T^* rooted at N_0 contains fewer than $2n/3$ leaves. The *balanced cut* (BC) of T^* is the cut $(V(N_0), V - V(N_0))$. For a given node N with children N_1, N_2, we say that the cut induced by N is the sum of the weights of the edges that have one extremity in $V(N_1)$ and the other in $V(N_2)$.

Let $(A \cup C, B \cup D)$ be the cut induced by the root node u of T, where A, B, C, D are such that $(A \cup B, C \cup D)$ is the balanced cut of T^*. Since $(A \cup C, B \cup D)$ is a ϕ-approximate sparsest cut:

$$\frac{w(A \cup C, B \cup D)}{|A \cup C| \cdot |B \cup D|} \leq \phi \frac{w(A \cup B, C \cup D)}{|A \cup B| \cdot |C \cup D|}.$$

By definition of N_0, $A \cup B$ and $C \cup D$ both have size in $[n/3, 2n/3]$, so the product of their sizes is at least $(n/3)(2n/3) = 2n^2/9$; developing $w(A \cup B, C \cup D)$ into four terms, we obtain

$$\begin{aligned} w(A \cup C, B \cup D) &\leq \phi \frac{9}{2n^2} |A \cup C||B \cup D|(w(A, C) + w(A, D) + w(B, C) + w(B, D)) \\ &\leq \phi \frac{9}{2} \left[\frac{|B \cup D|}{n} w(A, C) + w(A, D) + w(B, C) + \frac{|A \cup C|}{n} w(B, D) \right], \end{aligned}$$

and so the cost induced by node u of T^* satisfies

$$n \cdot w(A \cup C, B \cup D) \leq \frac{9}{2}\phi|B \cup D|w(A, C) + \frac{9}{2}\phi|A \cup C|w(B, D) + \frac{9}{2}\phi n(w(A, D) + w(B, C)).$$

To account for the cost induced by u, we thus assign a charge of $(9/2)\phi|B \cup D|w(e)$ to each edge e of (A, C), a charge of $(9/2)\phi|A \cup C|w(e)$ to each edge e of (B, D), and a charge of $(9/2)\phi n w(e)$ to each edge e of (A, D) or (B, C).

When we do this for every node u of T, how much does each edge get charged?

Lemma 4.2. *Let $G = (V, E)$ be a graph on n nodes. We consider the above charging scheme for T and T^*. Then, an edge $(v_1, v_2) \in E$ gets charged at most $(9/2)\phi \min((3/2)|V(LCA_{T^*}(v_1, v_2))|, n)w(e)$ overall, where $LCA_{T^*}(v_1, v_2)$ denotes the lowest common ancestor of v_1 and v_2 in T^*.* □

We temporarily defer the proof and first see how Lemma 4.2 implies the theorem. Observe (as in Reference [23]) that $\text{cost}(T^*) = \sum_{\{u,v\} \in E} |V(\text{LCA}_{T^*}(u, v))|w(u, v)$. Using Lemma 4.2, the sum of charges assigned is

$$\text{cost}(T) \leq \frac{9}{2}\phi \sum_{\{v_1, v_2\} \in E} \frac{3}{2}|V(\text{LCA}_{T^*}(v_1, v_2))|w(v_1, v_2) = \frac{27}{4}\phi\text{cost}(T^*).$$

Proof of Lemma 4.2. The lemma is proved by induction on the number of nodes of the graph. The base case follows immediately, and we proceed with the inductive step. For the inductive step, consider the cut $(A \cup C, B \cup D)$ induced by the root node u of T.

- Consider the edges that cross the cut. First, observe that edges of (A, B) or of (C, D) never get charged at all. Second, an edge $e = \{v_1, v_2\}$ of (A, D) or of (B, C) gets charged $(9/2)\phi nw(e)$ when considering the cost induced by node u, and does not get charged when considering any other node of T. In T^*, edge e is separated by the cut $(A \cup B, C \cup D)$ induced by N_0, so the least common ancestor of v_1 and v_2 is the parent node of N_0 (or above), and by definition of N_0 we have $|V(\mathrm{LCA}_{T^*}(v_1, v_2))| \geq 2n/3$, hence the lemma holds for e.
- An edge $e = \{v_1, v_2\}$ of $G[A] \cup G[C]$ does not get charged when considering the cut induced by node u. Apply Lemma 4.2 to $G[A \cup C]$ for the tree $T^*_{A\cup C}$ defined as the subtree of T^* induced by the vertices of $A \cup C$.[11] By induction, the overall charge to e due to the recursive calls for $G[A \cup C]$ is at most $(9/2)\phi \min((3/2)|V(\mathrm{LCA}_{T^*_{A\cup C}}(v_1, v_2))|, |A \cup C|)w(e)$. By definition of $T^*_{A\cup C}$, we have $|V(\mathrm{LCA}_{T^*_{A\cup C}}(v_1, v_2))| \leq |V(\mathrm{LCA}_{T^*}(v_1, v_2))|$, and $|A \cup C| \leq n$, so the lemma holds for e.
- An edge $\{v_1, v_2\}$ of (A, C) gets a charge of $(9/2)\phi|B \cup D|w(e)$ plus the total charge to e coming from the recursive calls for $G[A \cup C]$ and the tree $T^*_{A\cup C}$. By induction the latter is at most

$$(9/2)\phi \min((3/2)|V(\mathrm{LCA}_{T^*_{A\cup C}}(v_1, v_2))|, |A \cup C|)w(e) \leq (9/2)\phi|A \cup C|w(e).$$

  Overall, the charge to e is at most $(9/2)\phi nw(e)$. Since the cut induced by node u_0 of T^* separates v_1 from v_2, we have $|V(\mathrm{LCA}_{T^*}(v_1, v_2))| \geq 2n/3$, hence the lemma holds for e. For edges of (B, D) or of $G[B] \cup G[D]$, a symmetric argument applies. □

We complete our study of classical algorithms for hierarchical clustering by showing that the standard agglomerative heuristics can perform poorly when measured using Dasgupta's cost function (see Theorems B.1, B.2). Thus, the recursive sparsest-cut-based approach seems to be more reliable in the worst case for inputs given as a similarity graph. To understand better the success of the agglomerative heuristics, we restrict our attention random graphs (Section 5) and to inputs given as dissimilarity graphs and show that in these contexts agglomerative heuristics are efficient.

4.2 Dissimilarity-Based Inputs: Approximation Algorithms

In this section, we consider inputs given as dissimilarity graphs and focus on the problem of maximizing the analogue of Dasgupta's function: Find T maximizing the value function: $\mathrm{val}(T) = \sum_{N\in T} \mathrm{val}(N)$, where for each node N of T with children N_1, N_2, $\mathrm{val}(N) = w(V(N_1), V(N_2)) \cdot V(N)$. This optimization problem is NP-hard [23], and hence we focus on approximation algorithms.

We show (Theorem 4.3) that average-linkage achieves a 2/3 approximation for the problem. We then introduce a simple algorithm based on *locally densest cuts* and show (Theorem 4.6) that it also achieves a $2/3 - \varepsilon$ approximation for the problem.

Chatziafratis, Nizadeh, and Charikar [19] showed that the divisive heuristic that recursively splits the graph randomly into two parts also yields a 2/3-approximation. Thus our results regarding the performance of average-linkage and locally densest cut should be viewed in contrast to that of other popular heuristics; we show that single-linkage and bisection 2-means could perform quite poorly and do not achieve a constant factor approximation. Thus, the outcome of this section

[11] Note that $T^*_{A\cup C}$ is not necessarily the optimal tree for $G[A \cup C]$, which is why the lemma was stated in terms of every tree T^*, not just on the optimal tree.

is that average-linkage and locally densest cut are simple popular heuristics that seem to be more reliable than single-linkage or bisection 2-means in the worst-case setting.

We start with the following elementary upper bound on OPT.

Fact 1. *For any graph $G = (V, E)$, and weight function $w : E \to \mathbb{R}_+$, we have $OPT \leq n \cdot \sum_{e \in E} w(e)$.*

4.2.1 Average-Linkage for the Dissimilarity Setting. We show that average-linkage yields a 2/3-approximation in the dissimilarity setting.

Theorem 4.3. *For any graph $G = (V, E)$, and weight function $w : E \to \mathbb{R}_+$, the average-linkage algorithm (Algorithm 2) outputs a solution of value at least $\frac{2}{3} n \sum_{e \in E} w(e) \geq \frac{2}{3} OPT$.*

ALGORITHM 2: Average-Linkage Algorithm for Hierarchical Clustering (dissimilarity setting)

1: **Input:** Graph $G = (V, E)$ with edge weights $w : E \mapsto \mathbb{R}_+$
2: Create n singleton trees.
3: **while** there are at least two trees **do**
4: Take trees with roots N_1 and N_2 minimizing $\sum_{x \in V(N_1), y \in V(N_2)} w(x, y)/(|V(N_1)||V(N_2)|)$
5: Create a new tree with root N and children N_1 and N_2
6: **return** the resulting binary tree T

We recall the definitions of the previous section: When two trees are chosen at Step 4 of Algorithm 2, we say that they are *merged*. We say that all the trees considered at the beginning of an iteration of the while loop are the trees that are *candidates for the merge* or simply the *candidate trees*.

We first show the following lemma and then prove the theorem.

Lemma 4.4. *Let T be the output tree and A, B be the two children of the root. Then, the following holds:*

$$|V(A)|w(V(A), V(B)) \geq 2|V(B)|w(V(A)).$$

Proof. First, observe that if $|V(A)| = 1$, then $w(V(A)) = 0$, and the statement holds trivially. Assume that $|V(A)| > 1$ and let $a = |V(A)|(|V(A)| - 1)/2$.

For any node N_0 of T, let $\text{child}_1(N_0)$ and $\text{child}_2(N_0)$ be the two children of N_0. We first consider the subtree T_A of T rooted at A. We have

- $w(V(A)) = \sum_{A_0 \in T_A} w(V(\text{child}_1(A_0)), V(\text{child}_2(A_0)))$,
- $a = \sum_{A_0 \in T_A} |V(\text{child}_1(A_0))| \cdot |V(\text{child}_2(A_0))|$.

By using an averaging argument, there exists $A' \in T_A$ with children A_1, A_2 such that

$$\frac{w(V(A_1), V(A_2))}{|V(A_1)| \cdot |V(A_2)|} \geq \frac{w(V(A))}{a}. \tag{6}$$

We now consider the iteration of the while loop at which the algorithm merged the trees A_1 and A_2. Let $A_1, A_2, \ldots, A_k$ and $B_1, B_2, \ldots, B_\ell$ be the trees that were candidate for the merge at that iteration and such that $V(A_i) \cap V(B) = \emptyset$ and $V(B_i) \cap V(A) = \emptyset$. Observe that the sets of leaves of those trees form a partition of the sets $V(A)$ and $V(B)$, so we have

$$w(V(A), V(B)) = \sum_{i,j} w(V(A_i), V(B_j)),$$

$$|V(A)| \cdot |V(B)| = \sum_{i,j} |V(A_i)| \cdot |V(B_j)|.$$

By an averaging argument again, there exists A_i, B_j such that

$$\frac{w(V(A_i), V(B_j))}{|V(A_i)| \cdot |V(B_j)|} \leq \frac{w(V(A), V(B))}{|V(A)| \cdot |V(B)|}. \tag{7}$$

Now, since the algorithm merged A_1, A_2 rather than A_i, B_j, by combining Equations (6) and (7), we have

$$\frac{w(V(A))}{a} \leq \frac{w(V(A_1), V(A_2))}{|V(A_1)| \cdot |V(A_2)|} \leq \frac{w(V(A_i), V(B_j))}{|V(A_i)| \cdot |V(B_j)|} \leq \frac{w(V(A), V(B))}{|V(A)| \cdot |V(B)|}.$$

Substituting the value of a completes the proof. □

Proof of Theorem 4.3. We proceed by induction on the number of the nodes n in the graph. For any $n < 3$, the tree is unique and so the output optimal. Let A, B be the children of the root of the output tree T. By induction,

$$\text{val}(T) \geq (|V(A)| + |V(B)|) \cdot w(V(A), V(B)) + 2\frac{|V(A)|}{3} w(V(A)) + 2\frac{|V(B)|}{3} w(V(B)). \tag{8}$$

Applying Lemma 4.4 to A and B implies

$$(|V(A)| + |V(B)|) \cdot w(V(A), V(B)) \geq 2(|V(B)| w(V(A)) + |V(A)| w(V(B))).$$

Dividing both sides by 3 and plugging it into Equation (8) yields

$$\text{val}(T) \geq 2\frac{|V(A)| + |V(B)|}{3} w(V(A), V(B)) + 2\frac{|V(A)| + |V(B)|}{3} (w(V(A)) + w(V(B))).$$

Observing that $n = |V(A)| + |V(B)|$ and combining $\sum_{e \in E} w(e) = w(V(A), V(B)) + w(V(A)) + w(V(B))$ with Fact 1 completes the proof. □

4.2.2 A Simple Local-Search-based Approximation Algorithm for Worst-Case Inputs. In this section, we introduce a very simple algorithm (Algorithm 4) that achieves a similar approximation guarantee. The algorithm follows a divisive approach by recursively computing locally densest cuts using a local search heuristic (see Algorithm 3). This approach is similar to the recursive-sparsest-cut algorithm of Section 4.1. Here, instead of trying to solve the densest-cut problem (and so being forced to use approximation algorithms), we solve the simpler problem of computing a locally densest cut. This yields both a very simple local-search-based algorithm that has a good approximation guarantee.

We use the notation $A \oplus x$ to denote the set obtained by adding x to A if $x \notin A$ and by removing x from A if $x \in A$. We say that a cut (A, B) is an *ε/n locally densest cut* if for any x,

$$\frac{w(A \oplus x, B \oplus x)}{|A \oplus x| \cdot |B \oplus x|} \leq \left(1 + \frac{\varepsilon}{n}\right) \frac{w(A, B)}{|A||B|}.$$

The following local search algorithm computes an ε/n locally densest cut.

Theorem 4.5. *Algorithm 3 computes an ε/n locally densest cut in time $\widetilde{O}(n(n+m)/\varepsilon)$.*

ALGORITHM 3: Local Search for Densest Cut

1: **Input:** Graph $G = (V, E)$ with edge weights $w : E \mapsto \mathbb{R}_+$
2: Let (u, v) be an edge of maximum weight
3: $A \leftarrow \{v\}, B \leftarrow V \setminus \{v\}$
4: **while** $\exists x$: $\frac{w(A \oplus x, B \oplus x)}{|A \oplus x| \cdot |B \oplus x|} > (1 + \varepsilon/n) \frac{w(A,B)}{|A||B|}$ **do**
5: $\quad A \leftarrow A \oplus x, B \leftarrow B \oplus x$
6: **return** the cut (A, B)

Proof. The proof is straightforward and given for completeness. By definition, the algorithm computes an ε/n locally densest cut so we only need to argue about the running time. The weight of the cut is initially at least $w_{\max}$, the weight of the maximum edge weight, and in the end at most $mw_{\max}$. Since the weight of the cut increases by a factor of $(1+\varepsilon/n)$ at each iteration, the total number of iterations of the while loop is at most $\log_{1+\varepsilon/n}(mw_{\max}/w_{\max}) = \widetilde{O}(n/\varepsilon)$. Each iteration takes time $O(m+n)$, so the running time of the algorithm is $\widetilde{O}(n(m+n)/\varepsilon)$. □

ALGORITHM 4: Recursive Locally Densest Cut for Hierarchical Clustering

1: **Input:** Graph $G = (V, E)$, with edge weights $w : E \mapsto \mathbb{R}_+$, $\varepsilon > 0$
2: Compute an ε/n locally densest cut (A, B) using Algorithm 3
3: Recurse on $G[A]$ and $G[B]$ to obtain rooted trees T_A and T_B.
4: **return** the tree T whose root node has two children with subtrees T_A and T_B.

Theorem 4.6. *Algorithm 4 returns a tree of value at least*

$$\frac{2n}{3}(1-\varepsilon)\sum_e w(e) \geq \frac{2}{3}(1-\varepsilon)OPT,$$

in time $\widetilde{O}(n^2(n+m)/\varepsilon)$.

The proof relies on the following lemma.

Lemma 4.7. *Let* (A, B) *be an* ε/n *locally densest cut. Then,*

$$(|A|+|B|)w(A,B) \geq 2(1-\varepsilon)(|B|w(A)+|A|w(B)).$$

Proof. First, assume $|A| > 1$ and let $v \in A$. By definition of the algorithm,

$$(1+\varepsilon/n)\frac{w(A,B)}{|A||B|} \geq \frac{w(A\setminus\{v\}, B\cup\{v\})}{(|A|-1)(|B|+1)}.$$

Rearranging,

$$\frac{(|A|-1)(|B|+1)}{|A||B|}(1+\varepsilon/n)w(A,B) \geq w(A\setminus\{v\}, B\cup\{v\}) = w(A,B)+w(v,A)-w(v,B).$$

Summing over all vertices of A, we obtain

$$|A|\frac{(|A|-1)(|B|+1)}{|A||B|}(1+\varepsilon/n)w(A,B) \geq |A|w(A,B)+2w(A)-w(A,B).$$

Rearranging and simplifying,

$$(|A|-1)(|B|+1)\frac{\varepsilon}{n}w(A,B)+(|A|-1)(1+\varepsilon/n)w(A,B) \geq 2|B|w(A).$$

Since $|B|+1 \leq n$, this gives

$$|A|w(A,B) \geq 2(1-\varepsilon)|B|w(A).$$

Notice that if $|A| = 1$, then the above inequality can be obtained trivially as $w(A) = 0$.
Proceeding similarly with B and summing the two inequalities yields the lemma. □

Proof of Theorem 4.6. We first show the approximation guarantee. We proceed by induction on the number of vertices. The base case is trivial. By inductive hypothesis,

$$\text{val}(T) \geq nw(A,B) + \frac{2}{3}\cdot(1-\varepsilon)(|A|w(A)+|B|w(B)),$$

where $n = |A| + |B|$. Lemma 4.7 implies

$$nw(A,B) = (|A| + |B|)w(A,B) \geq 2(1-\varepsilon)(|B|w(A) + |A|W(B)).$$

Hence,

$$\frac{|A| + |B|}{3} w(A,B) \geq \frac{2}{3}(1-\varepsilon)(|B|w(A) + |A|w(B)).$$

Therefore,

$$\mathrm{val}(T) \geq \frac{2n}{3}(1-\varepsilon)(w(A,B) + w(A) + w(B)) = (1-\varepsilon)\frac{2n}{3}\sum_e w(e).$$

To analyze the running time, observe that by Theorem 4.5, a recursive call on a graph $G' = (V', E')$ takes time $\widetilde{O}(|V'|(|V'| + |E'|)/\varepsilon)$ and that the number of recursive calls is $O(n)$. □

Remark 3. In Appendix B, we show that other commonly used algorithms, single-linkage, or bisection 2-Center, can perform arbitrarily poorly (see Theorem B.3). Hence, average-linkage is more robust for dissimilarity inputs.

5 A GENERAL HIERARCHICAL STOCHASTIC BLOCK MODEL

Lyzinski et al. [35] studied a Hierarchical Stochastic Block Model (HSBM) and gave an algorithm that recovers the "ground-truth" hierarchical clustering when the hidden clusters have linear size and the ratio between the minimum edge probability and the maximum edge probability is $O(1)$. Krishamurthy et al. [32] provide *active learning algorithms* that guarantee exact recovery of the ground-truth tree under certain conditions.

We introduce a generalization of HSBM and give a $(1 + o(1))$-approximation algorithm to recover a near-optimal hierarchical clustering under a more general setting. Our algorithm is very similar to the widely used linkage approach and remains easy to implement and parallelize. The take-home message is that, on "structured inputs" the linkage (agglomerative) heuristics perform provably well, taking a step toward explaining their success in practice.

The graphs generated from our model are a noisy version of a "ground-truth hierarchical clustering tree" (see Definition 5.2). For a motivating example, the tree of life has a natural associated hierarchical clustering, but because of extinct species, the input is imperfect and noisy. Our definition uses the notion of a generating tree (Definition 2.2) which can be associated to an ultrametric (and therefore to a "natural" hierarchical clustering). Each edge of the graph thus generated has a certain probability of being present, which only depends on the underlying ground-truth tree and the least common ancestor two endpoints.

Definition 5.1 (Hierarchical Stochastic Model (HSM)). Let $\widetilde{T}$ be a generating tree for an n-vertex graph $\bar{G}$, called the expected graph, such that all weights are in $[0, 1]$. A hierarchical stochastic model is a random graph G such that for every two vertices u and v, the edge $\{u, v\}$ is present independently with probability $w(\{u, v\}) = W(\mathrm{LCA}_T(\sigma^{-1}(u), \sigma^{-1}(v)))$, where w and W are the weights functions associated with $\widetilde{T}$ as per Definition 2.2. In words, the probability of an edge being present is given by the weight of the lowest common ancestor of the corresponding vertices in $\widetilde{T}$.

Definition 5.2 (Hierarchical Stochastic Block Model (HSBM)). A hierarchical stochastic model is a k-hierarchical stochastic block model (k-HSBM) if T contains k disjoint subtrees, spanning all leaves, such that W is constant inside each of the k subtrees.

We let $p_{\min}$ denote the weight of the root node of $\widetilde{T}$ for both HSM and HSBM. $\widetilde{T}$ is called a *ground-truth tree* (see Figure 5 for an illustration). For any tree T, whenever there is a possibility of ambiguity, we use $\mathrm{cost}(T; G)$ and $\mathrm{cost}(T; \bar{G})$ to denote the the costs of the cluster tree T for graphs

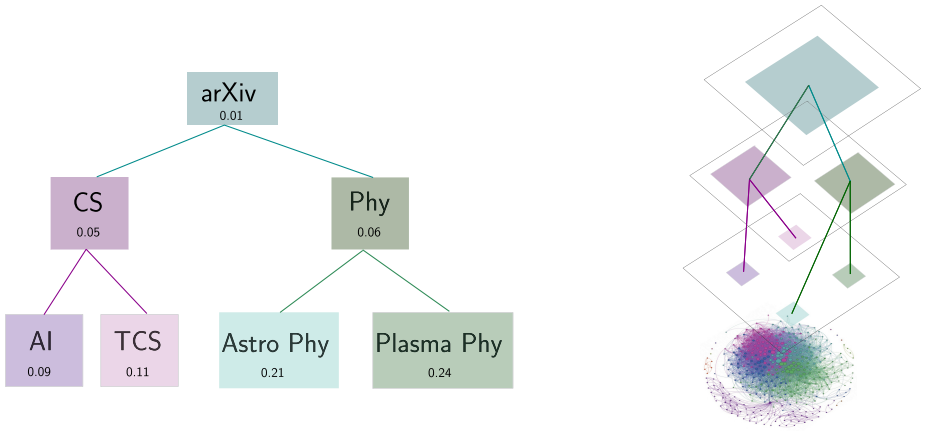


Fig. 5. An illustration of an HSBM. Nodes represent authors on arXiv.org, and there is an edge between two nodes if the two authors have a joint publication. Typically, authors of the same community have more edges within their community and sub-communties. This is reflected by the increase of the connection probability of a from the root toward the leaves.

G and for $\bar{G}$, respectively: Observe that $\text{cost}(T;\bar{G}) = \mathbb{E}[\,\text{cost}(G;T)\,]$, where the expectation is taken with respect to the randomness over the edges in G.

5.1 Objective Functions and Ground-Truth Tree

In this section, we show that the cost of the ground-truth tree (which is optimal for the expected graph $\bar{G}$) is near-optimal for the realized graph. All results in this section assume that the cost function is the one introduced by Dasgupta [23]; under some smoothness assumption on the cost functions the results can be generalized to other cost functions with (possibly) slightly worse constant factors.

In Proposition 5.4 we show that the tree T minimizing the expected cost for a graph G generated from an HSM is minimized for a ground-truth tree (note that the ground-truth tree might not be unique, but they are all of the same cost). Furthermore, we show in Theorem 5.5 that, under mild assumptions, even in the realized graph, the cost induced by the ground-truth tree is within a $(1 + o(1))$ factor of the cost of the optimal tree.

For the proof of Proposition 5.4, we will make use of a slightly generalized version of the Hoeffding bound (see Reference [29]).

Proposition 5.3 ([29]). *Let $X = \sum_{i=1}^{m} X_i$ be a sum of m independent random variables with $a_i \leq X_i \leq b_i$ for all i. Then for any $t > 0$:*

$$\mathbb{P}[\,|X - \mathbb{E}[\,X\,]| \geq t\,] \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{m}(b_i - a_i)^2}\right). \tag{9}$$

Proposition 5.4. *Let G be a graph generated according to an HSM (see Definition. 5.1). Then a tree T is a ground-truth tree for G if and only if*

$$\mathbb{E}[\,\textit{cost}(T)\,] = \min_{T'} \mathbb{E}[\,\textit{cost}(T')\,].$$

Proof. Let $\bar{G}$ be the expected graph. By linearity of expectation, for any tree T (not necessarily the ground-truth one) $\mathbb{E}[\,\mathrm{cost}(T;G)\,] = \mathrm{cost}(T;\bar{G})$. By the definition of admissibility, $\mathrm{cost}(T;\bar{G}) = \min_{T'} \mathrm{cost}(T';\bar{G})$ if an only if T is generating (see Definition 3.1). □

The following shows that the ground-truth tree is a $(1+o(1))$-approximation of the optimal solution.

Theorem 5.5. *Let $\widetilde{T}$ be a ground-truth tree for a graph G generated from an HSM (see Definition. 5.1). If $p_{\min} = \omega(\sqrt{\log n/n})$, then, with high probability,*

$$cost(\widetilde{T};G) \le (1+o(1)) \min_{T'} cost(T';G) = (1+o(1))OPT.$$

Proof. It suffices to show that with high probability, the following holds: For every binary tree T with n leaves labeled by the vertices of G,

$$|\mathrm{cost}(T;G) - \mathbb{E}[\,\mathrm{cost}(T;G)\,]| \le o(\mathbb{E}[\,\mathrm{cost}(T)\,]). \tag{10}$$

Indeed, if Equation (10) holds, then $\mathrm{cost}(\widetilde{T};G) \le (1+o(1))\mathbb{E}[\,\mathrm{cost}(\widetilde{T};G)\,]$; we know from Proposition 5.4 that $\mathbb{E}[\,\mathrm{cost}(\widetilde{T};G)\,] = \min_{T'} \mathbb{E}[\,\mathrm{cost}(T';G)\,]$ and by Equation (10) again for any tree T', $\mathbb{E}[\,\mathrm{cost}(T';G)\,] \le (1+o(1))\mathrm{cost}(T';G)$.

To prove Equation (10), we observe that the number of possible cluster trees (including labelings of the leaves to vertices of G) is bounded by $2^{c\cdot n\log n}$, for some constant c. Fix a cluster tree T'. It suffices to prove that for any fixed $c > 0$,

$$\mathbb{P}[\,|\mathrm{cost}(T';G) - \mathbb{E}[\,\mathrm{cost}(T';G)\,]| \ge o(\mathbb{E}[\,\mathrm{cost}(T';G)\,])\,] \le \exp(-2cn\log n).$$

We can write

$$\mathrm{cost}(T';G) = \sum_{N\in T'} (|V(\mathrm{child}_1(N))| + |V(\mathrm{child}_2(N))|) \sum_{\substack{i\in V(\mathrm{child}_1(N))\\ j\in V(\mathrm{child}_2(N))}} \mathbf{1}_{(i,j)\in E} = \sum_{1\le i<j\le n} Z_{i,j},$$

where $Z_{i,j} = |V(\mathrm{LCA}_{T'}(i,j))| \cdot \mathbf{1}_{(i,j)\in E}$. Observe that the random variables $Z_{i,j}$ are independent. We apply Proposition 5.3 with $t = \mathbb{E}[\,\mathrm{cost}(T';G)\,] \cdot \sqrt{6c}\sqrt{\frac{\log n}{n}}/p_{\min} = o(\mathbb{E}[\,\mathrm{cost}(T';G)\,])$ and derive:

$$\mathbb{P}\left[\,|\mathrm{cost}(T';G) - \mathbb{E}[\,\mathrm{cost}(T';G)\,]| \ge \mathbb{E}[\,\mathrm{cost}(T')\,] \cdot \frac{\sqrt{6c}\cdot\sqrt{\frac{\log n}{n}}}{p_{\min}}\right]$$
$$\le \exp\left(-\frac{2\left(\mathbb{E}[\,\mathrm{cost}(T';G)\,]\cdot\frac{\sqrt{6c}\cdot\sqrt{\frac{\log n}{n}}}{p_{\min}}\right)^2}{\sum_{i<j}|V(\mathrm{LCA}_{T'}(i,j))|^2}\right).$$

Now, assume that the following claim holds.

Claim 1. *Let $\kappa(n)$ denote the cost of a unit weight clique. Then:*

(1) $\mathbb{E}[\,cost(T';G)\,] \ge \kappa(n)\cdot p_{\min}$
(2) $\sum_{i<j}|V(LCA_{T'}(i,j))|^2 \le n\cdot\kappa(n)$.

Using Claim 1 and the fact that $\kappa(n) \ge n^3/6$:

$$\exp\left(-\frac{2\left(\mathbb{E}[\,\mathrm{cost}(T';G)\,]\cdot\frac{\sqrt{6c}\cdot\sqrt{\frac{\log n}{n}}}{p_{\min}}\right)^2}{\sum_{i<j}|V(\mathrm{LCA}_{T'}(i,j))|^2}\right) \le \exp\left(-\frac{2\cdot\kappa(n)\cdot 6c\cdot\log n}{n^2}\right) \le \exp(-2c\cdot n\log n).$$ □

Note that the assumption on $p_{\min}$ is necessary for our proof technique, which relies on taking Union bound over all possible generating trees. It remains an interesting open question whether this assumption is necessary. Furthermore, this proof is the "bottleneck" for the choice of $p_{\min}$.

We now turn to the proof of Claim 1.

Proof of Claim 1. Let $\bar{G}$ denote the expected graph. $\mathbb{E}[\,\text{cost}(T';G)\,]$ is equal to $\text{cost}(T';\bar{G})$, which is a linear function of the edge weights of $\bar{G}$. However, the clique has the same cost for every tree, so consider its cost for tree T'. The minimum edge weights is $p_{\min}$ for $\bar{G}$, the edge weight is 1 for every edge of the clique; hence, the first statement.

For the second statement we write

$$\sum_{i<j} |V(\text{LCA}_{T'}(i,j))|^2 \leq n \sum_{i<j} |V(\text{LCA}_{T'}(i,j))| = n\kappa(n). \qquad \square$$

5.2 Algorithm Linkage++, a $(1+\varepsilon)$-Approximation Algorithm in the HSBM

We start with some useful notation for a k-HSBM: $n_1, n_2, \ldots, n_k$, the sizes of the k *bottom-level clusters*, $p_1, p_2, \ldots, p_k$, the value of W inside each cluster $p_{\max} = \max_i p_i$, and $n_{\min}$, the size of the smallest of the k bottom-level clusters. We extend σ to be the correspondence between clusters and the roots of the k subtrees.

For a graph G, we denote by $\mathbf{u}$ the adjacency vector of vertex u, i.e., the column of the adjacency matrix corresponding to the vertex u. Given a graph generated from a k-HSBM (or from the planted partition model) with k bottom-level clusters, we say that is satisfies the *separation condition* if there exists a universal constant c such that for each pair of vertices u, v belonging to different bottom-level clusters (or simply clusters in the planted partition), they satisfy

$$\|\mathbb{E}[\,\mathbf{u}\,] - \mathbb{E}[\,\mathbf{v}\,]\|_2^2 \geq c \cdot k \cdot ((p_{\max} + \omega(\log^6 n/n)) \cdot n/n_{\min} + \log(n)), \tag{11}$$

where $\mathbb{E}[\,\mathbf{u}\,]$ is the entrywise expectation. We establish the following theorem.

Theorem 5.6. *Let G be a graph generated from a k-HSBM (see Definition 5.2). If G satisfies the separation condition (11), $p_{\min} = \omega(\sqrt{\log n/n})$, and $n_{\min} \geq n^{1/4} \cdot \log^{1/4} n$, then, with high probability, Algorithm 5 outputs a tree T that satisfies cost$(T;G) = (1+o(1))OPT$.*

We consider a simple algorithm, called linkage++, which works in two phases (see Algorithm 5)

(1) Perform a singular value decomposition on the adjacency matrix to embed nodes in Euclidean space. Then apply single-linkage using the Euclidean distance until we obtain k clusters.
(2) Consider these as bottom-level clusters and apply single-linkage using the edge density between these clusters in the input graph G to finish building the hierarchical clustering tree.

The idea of using spectral techniques for graph partitioning goes back at least to Bopanna [13]. We use a result of McSherry [37] that considers the planted partition model.

Theorem 5.7 ([37], Observation 11 and a simplification of Theorem 12). *Let G be a random graph generated from the planted partition model satisfying the separation condition and where each of the k clusters contains at least $n_{\min}$ vertices.*

Then, the algorithm of Reference [37, Theorem 12] with inputs G, k maps V to points $\{\zeta(1), \ldots, \zeta(n)\}$ in a k-dimensional subspace of $\mathbb{R}^n$ such that the following holds: with probability at least $1 - 1/n^3$ over the random graph G and with probability $1/k$ over the random bits of the algorithm, there exists $\eta > 0$ such that for any two vertices u and v:

ALGORITHM 5: Linkage++

1: **Input:** Unweighted graph $G = (V, E)$ on n vertices.
2: **Parameter:** An integer k.
3: **Output:** A Hierarchical Clustering Tree of G.
4: Let $\zeta(1), \ldots, \zeta(n)$ denote the n points corresponding to vertices of V, spanning a k-dimensional subspace of $\mathbb{R}^n$, that are obtained by applying Theorem 5.7 with parameters G, k.
5: Run the single-linkage algorithm on points $\{\zeta(1), \ldots, \zeta(n)\}$, using Euclidean distance, until there are exactly k clusters $C_1^\zeta, \ldots, C_k^\zeta$.
6: Define similarity between clusters by $\text{sim}(C_i^\zeta, C_j^\zeta) = w(C_i^\zeta, C_j^\zeta)/(|C_i^\zeta||C_j^\zeta|)$, where $w(C_i^\zeta, C_j^\zeta)$ is the number of edges of G between the corresponding vertices.
7: **while** there is more than one cluster **do**
8: Let A, B denote the two clusters maximizing $\text{sim}(A, B)$.
9: Define cluster $D = A \cup B$ and let $\text{sim}(D, E) = \max(\text{sim}(A, E), \text{sim}(B, E))$ for all clusters E.
10: Add D to and remove A and B from the set of clusters.
11: The sequence of merges in the while-loop (Steps 7 to 10) induces a hierarchical clustering tree T'_k with k leaves $\{C_1^\zeta, \ldots, C_k^\zeta\}$. For an internal node N of T'_k created by merging $A, B \subseteq \{C_1^\zeta, \ldots, C_k^\zeta\}$, define the weight, $W'(N) = \text{sim}(A, B) = \max_{C_i^\zeta \in A, C_j^\zeta \in B} \text{sim}(C_i^\zeta, C_j^\zeta)$. Replace each leaf of T'_k by an arbitrary binary tree on $|C_k^\zeta|$ leaves labeled according to the corresponding vertices of V to obtain a hierarchical tree T.
12: **Repeat** the algorithm $2k \log n$ times and output the tree T that minimizes $\text{cost}(T; G)$.

(1) if u and v are in the same cluster then $\|\zeta(u) - \zeta(v)\|_2^2 \leq \eta$ and
(2) if u and v are in different clusters then $\|\zeta(u) - \zeta(v)\|_2^2 > 2\eta$.

Our analysis relies on analyzing the probability of occurrence of two events $\mathcal{E}_1$ and $\mathcal{E}_2$.

Event $\mathcal{E}_1$. Fix one of the $2k \log n$ iterations of Lines 4–11 of Algorithm 5. We define event $\mathcal{E}_1$ as the event that after Line 4 of the algorithm, the resulting vectors satisfy the conclusion of Theorem 5.7, i.e.,

(1) if u and v are in the same cluster then $\|\zeta(u) - \zeta(v)\|_2^2 \leq \eta$ and
(2) if u and v are in different clusters then $\|\zeta(u) - \zeta(v)\|_2^2 > 2\eta$.

We let $\psi(u) \in [k]$ denote the cluster containing vertex u. From the above theorem, we deduce the following lemma.

Lemma 5.8. *Let G be generated by a k-HSBM. Assume event $\mathcal{E}_1$ holds. Then, the k clusters obtained after Step 5 correspond exactly to the k hidden bottom-level clusters.*

Proof. By event $\mathcal{E}_1$, any linkage algorithm, e.g., single-linkage, performing merges starting from the set $\{\zeta(1), \ldots, \zeta(n)\}$ until there are k clusters will merge clusters at a distance of at most η and hence, the clusters obtained after Step 5 correspond to the assignment ψ. □

Let $C_1^*, \ldots, C_k^*$ be the hidden bottom-level clusters, i.e., $C_i^* = \{v | \psi(v) = i\}$. Let π denote the permutation $\pi : [k] \to [k]$ such that C_j^ζ corresponds to $C_{\pi(j)}^*$. Let $V(C_i^*)$ denote the set of vertices in the bottom-level clusters.

Event $\mathcal{E}_2$. We define $\mathcal{E}_2$ as the event that the graph G satisfies that for any pair of bottom-level clusters C_i, C_j, $|w(C_i, C_j) - E_{(C_i,C_j)}| < \varepsilon^2 E_{(C_i,C_j)}$, where $E_{(C_i,C_j)} = |V(C_i)| \cdot |V(C_j)| \cdot W(\mathrm{LCA}_{\widetilde{T}}(C_i, C_j))$.

γ-approximate Ground-Truth Tree. We say that a tree $T = (\mathcal{N}, \mathcal{E})$ is a *γ-approximate ground-truth tree* for G and $\widetilde{T}$ if there exists a weight function $W' : \mathcal{N} \mapsto \mathbb{R}_+$ such that for any two vertices u, v, we have that

(1) $\gamma^{-1} W'(\mathrm{LCA}_T(u, v)) \le W(\mathrm{LCA}_{\widetilde{T}}(u, v)) \le \gamma W'(\mathrm{LCA}_T(u, v))$ and
(2) for any node N of T and any node N' descendant of N in T, $W'(N) \le W'(N')$.

Lemma 5.9. *Let G be generated according to a k-HSBM. Assume that event $\mathcal{E}_2$ occurs and that the clusters obtained after Step 5 correspond to the assignment ψ. Then the output tree T of Algorithm 5 is a $(1 + \varepsilon/3)$-approximate ground-truth tree.*

Proof. The hypothesis of lemma assumes that the bottom level clusters are identified correctly as per the assignment ψ. Let $C_1^*, \ldots, C_k^*$ denote these bottom-level clusters. We focus on the tree T_k' (cf. Algorithm 5), which has leaves given by bottom level clusters. Let $\widetilde{T}_k'$ be the *generating* tree which is truncated to have exactly k leaves also given by the k bottom level clusters.

Since $\mathcal{E}_2$ occurs, for as sufficiently small ε, we have that for every pair $i, j \in [k]$:

$$\frac{W(\mathrm{LCA}_{\widetilde{T}'}(C_i^*, C_j^*))}{1 + \varepsilon/3} \le \mathrm{sim}(C_i^*, C_j^*) \le (1 + \varepsilon/3) W(\mathrm{LCA}_{\widetilde{T}'}(C_i^*, C_j^*)). \tag{12}$$

Let $\mathcal{N}$ denote the set of all nodes of T (including internal nodes). For any internal node $N \in \mathcal{N}$, define $\Lambda(N)$ to denote the set of leaves in the subtree rooted at N. Recall that W' is defined as follows (cf. Algorithm 5)—for every internal node $N \in \mathcal{N}$, if N_L and N_R denote its left and right children, then:

$$W'(N) = \max_{C_i^* \in \Lambda(N_L), C_j^* \in \Lambda(N_R)} \mathrm{sim}(C_i^*, C_j^*).$$

Our goal is to show that for every $i, j \in [k]$

$$W(\mathrm{LCA}_{\widetilde{T}'}(C_i^*, C_j^*))/(1 + \varepsilon/3) \le W'(\mathrm{LCA}_{T'}(C_i^*, C_j^*)) \le (1 + \varepsilon/3) W(\mathrm{LCA}_{\widetilde{T}'}(C_i^*, C_j^*)). \tag{13}$$

First, we prove the LHS of Inequality (13),

$$W'(\mathrm{LCA}_{T'}(C_i^*, C_j^*)) \ge \mathrm{sim}(C_i^*, C_j^*) \ge W(\mathrm{LCA}_{\widetilde{T}'}(C_i^*, C_j^*))/(1 + \varepsilon/3).$$

Thus it only remains to prove the RHS of Inequality (13). Consider an arbitrary pair of bottom-level clusters C_i^*, C_j^*, and let $N = \mathrm{LCA}_{T_k'}(C_i^*, C_j^*)$ be the least common ancestor of C_i^* and C_j^* in T_k'. Let N_L and N_R be the left and right child of N in T_k'. Similarly, let $\widetilde{N} = \mathrm{LCA}_{\widetilde{T}'}(C_i^*, C_j^*)$ be the least common ancestor of C_i^* and C_j^* in $\widetilde{T}_k'$, and let $\widetilde{N}_L$ and $\widetilde{N}_R$ be the left and right child of $\widetilde{N}$ in $\widetilde{T}_k'$. We consider the the following two cases:

Case 1: $\Lambda(N_L) \subseteq \Lambda(\widetilde{N}_L)$ and $\Lambda(N_R) \subseteq \Lambda(\widetilde{N}_R)$. Observe that in this case the clusters being merged are $A = \Lambda(N_L)$ and $B = \Lambda(N_R)$ and let $C_l^* \in \Lambda(N_L)$ and $C_k^* \in \Lambda(N_R)$ be such that $\mathrm{sim}(A, B) = \mathrm{sim}(C_l^*, C_k^*)$. Without loss of generality, we have that $C_i^* \in \Lambda(N_L)$ and $C_j^* \in \Lambda(N_R)$ and as a result, we get

$$W'(N) = \mathrm{sim}(A, B) = \mathrm{sim}(C_l^*, C_k^*) \le (1 + \varepsilon/3) W(\widetilde{N}),$$

where the last inequality follows from the fact that $\mathrm{LCA}_{\widetilde{T}'}(C_l^*, C_k^*) = \widetilde{N}$ and as $\mathcal{E}_2$ holds.

Case 2: $\exists C_l^* \in \Lambda(N_L), C_l^* \notin \Lambda(\widetilde{N}_L)$ or $\exists C_k^* \in \Lambda(N_R), C_k^* \notin \Lambda(\widetilde{N}_R)$. Without loss of generality, let $C_i^* \in \Lambda(N_L)$ and suppose that there exists $C_l^* \in \Lambda(N_L), C_l^* \notin \Lambda(\widetilde{N}_L)$. Then there must exist a descendant of N_L (possibly including N_L itself), which for the first time merges sets A and B satisfying: $\exists C_{i'}^* \in A, C_{i'}^* \in \Lambda(\widetilde{N}_L)$ and $\exists C_{l'}^* \in B, C_{l'}^* \notin \Lambda(\widetilde{N}_L)$, such that $\text{sim}(A,B) = \text{sim}(C_{i'}^*, C_{l'}^*)$. Then note that by definition, $\text{LCA}_{\widetilde{T}'}(C_{i'}^*, C_{l'}^*)$ must be an ancestor of $\widetilde{N}$ (possibly including $\widetilde{N}$ itself).

Thus, we have

$$W'(N) \le \text{sim}(C_{i'}^*, C_{l'}^*) \le (1+\varepsilon/3)W(\text{LCA}_{\widetilde{T}'_k}(C_{i'}^*, C_{l'}^*)) \le (1+\varepsilon/3)W(\widetilde{N}).$$

The proof follows by observing that T is obtained from T'_k by simply replacing the leaves of T'_k by arbitrary binary subtrees for each of the bottom-level clusters. □

The following lemma allows us to bound the cost of an approximate ground-truth tree.

Lemma 5.10. *Let G be a graph generated according to a k-*HSBM*. Let T be a γ-approximate ground-truth tree. Then, cost$(T;G) \le \gamma(1+o(1))OPT$.*

Proof. Let $\widetilde{T}$ be a ground-truth tree for G. Let $\bar{G}$ be the expected graph associated to $\widetilde{T}$ and G.

By Theorem 3.2 the cost of $\widetilde{T}$ is optimal for $\bar{G}$. Furthermore, by Theorem 5.5, we have that the cost of $\widetilde{T}$ on $\bar{G}$ and the cost of $\widetilde{T}$ are within a factor of $(1+o(1))$. We thus need to show that $\text{cost}(T;G) \le \gamma(1+o(1))\text{cost}(\widetilde{T};G)$.

Recall that T is the γ-approximate ground-truth tree and we define a new graph $\bar{G}_T$ which has the same set of vertices V as $\bar{G}$: For each pair of vertices $u, v \in V$ we create an edge in $\bar{G}_T$ with weight $w_{\bar{G}_T}(u,v) = W'(\text{LCA}_T(u,v))$. By definition of W', it follows that T is generating for $\bar{G}_T$, and so applying Theorem 3.2, we obtain that the cost of T for $\bar{G}_T$, denoted by $\text{cost}(T;\bar{G}_T)$, is less than the cost of $\widetilde{T}$ for $\bar{G}_T$, $\text{cost}(\widetilde{T};\bar{G}_T)$.

Now recall that the cost of a given tree T, for a given graph G' can be rewritten as follows: $\text{cost}(T;G') = \sum_{u,v} w_{G'}(u,v) \cdot |V(\text{LCA}_T(u,v))|$, where $w_{G'}(u,v)$ is the weight of the edge u,v in G'.

Thus, since by definition of T, we have $\gamma^{-1}W'(\text{LCA}_T(a,b)) \le W(\text{LCA}_{\widetilde{T}}(a,b)) \le \gamma W'(\text{LCA}_T(a,b))$.

Hence,

$$\begin{aligned}\text{cost}(T;\bar{G}) &= \sum_{u,v} w_{\bar{G}}(u,v) \cdot |V(\text{LCA}_T(u,v))| \le \sum_{u,v} \gamma \cdot w_{\bar{G}_T}(u,v) \cdot |V(\text{LCA}_T(u,v))| \\ &= \gamma \cdot \text{cost}(T;\bar{G}_T) \le \gamma \cdot \text{cost}(\widetilde{T};\bar{G}_T).\end{aligned}$$

Similarly, one can who that $\text{cost}(\widetilde{T};\bar{G}_T) \le \gamma \cdot \text{cost}(\widetilde{T};\bar{G})$. Combining the two yields the lemma. □

Proof of Theorem 5.6. We first analyze the probability of event $\mathcal{E}_1$. By Theorem 5.7, for one execution of Line 4 of Algorithm 5, $\mathcal{E}_1$ holds with probability at least $1 - 1/n^3$ over the random graph G and probability at least $1/k$ over the random bits of the algorithm. Since Line 12 of Algorithm 5 repeats the execution $10 \cdot k \cdot \log n$ times with independent random bits, the probability that event $\mathcal{E}_1$ holds for at least one of the $10 \cdot k \cdot \log n$ executions is at least $1 - e^{-10\log n} - 1/n^2$.

We next analyze the probability of event $\mathcal{E}_2$. Fix two bottom-level clusters. Thanks to our assumptions on $p_{\min}$ and $n_{\min}$, the expected number of edges of the cut between those two clusters is at least $n_{\min}^2 p_{\min} = \omega(\log n)$. Applying Chernoff bounds, the value of the cut is concentrated around its expectation with a probability of at least $1 - 2^{-4\log n}$. Taking a union bound over all $\binom{k}{2}$ pairs of bottom-level clusters, using that $k \le n$, we get that the probability for $\mathcal{E}_2$ to hold for some choice of $\varepsilon = o(1)$ is at least $1 - n^2 2^{-4\log n} \ge 1 - 1/n^2$.

Consider an execution during which events $\mathcal{E}_1$ and $\mathcal{E}_2$ both occur. Combining Lemmas 5.8, 5.9, and 5.10 imply the conclusion of Theorem 5.6. □

We note that k might not be known in advance. However, different values of k can be tested and an $O(1)$-estimate on k is enough for the proofs to hold. Thus, it is possible to run Algorithm 5 $O(\log n)$ times with different "guesses" for k and take the minimum-cost tree of these runs.

6 SEMI-RANDOM MODELS

The notation for random graphs used in this section is described in detail in Section 5.

6.1 Algorithm for Semi-Random Model using SDP Relaxations

We show that in random and *semi-random* (defined below) graph models generated according to an HSM, an SDP-based algorithm can be used to guarantee an $O(1)$-approximation with high-probability in a regime beyond that proved in Theorem 5.6. The proof of the following result follows using the technique of Makarychev et al. [36] to obtain $O(1)$-approximations to problems such as sparsest cut and small-set expansion (SSE) in random and semi-random settings combined with Theorem 4.1 that shows that approximations to (roughly) balanced min-cut problems can be used to obtain an equivalent approximation ratio for the problem of finding a minimum cost hierarchical cluster tree.

To generate a random graph, $G = (V, E)$, we use an HSM (Definition 5.1). The semi-random model simply considers a random graph generated as above and an adversary is allowed to remove edges from G, but not add any. Note that in either case the comparison is to the cost of the generating tree on the graph $\bar{G}$ (cf. Definition 5.1).

6.2 Algorithm in Semi-Random Model using SDP Relaxations

This section is dedicated to the proof of the following theorem.

Theorem 6.1. *Let G be a graph generated from the HSM (Definition 5.1) with $p_{\min} = \Omega(\log n/n^{2/3})$. Then, there exists a randomized polynomial time algorithm that with probability $1 - o(1)$ outputs a tree T such that*

$$cost(T; G) = O(OPT(\bar{G})), \tag{14}$$

where $OPT(\bar{G})$ denotes the value of the optimal tree for $\bar{G}$ and we note that $OPT(\bar{G}) = cost(\widetilde{T}; \bar{G})$, where $\widetilde{T}$ is the generating tree. Furthermore, the above holds even in the semi-random case, i.e., when an adversary is allowed to remove any subset of the edges from G.

Similar to Section 5, the assumption on $p_{\min}$ is crucial to our proof techniques; it remains an interesting open question whether the assumption is necessary. It is worth mentioning, that the assumption on $p_{\min}$ in Theorem 6.1 is much weaker than the one in Section 5, where we prove a $(1 + o(1))$-approximation.

6.2.1 Background. In this section, we recall the work of Makarychev et al. [36]. Essentially all of this section is directly cited from this work and we only provide it in this article for completeness.

While we do not require to go into details, we define the crude SDP for Small-Set Expansion (SSE) used by Makarychev et al. [36] below. $\bar{u}$ denotes some vector representation corresponding to vertex u in the SDP. The reader may refer to SDP solutions φ occurring in the rest of this section to mean feasible solutions to the following SDP. Note that solving the SSE problem gives

a (roughly) balanced sparse cut when $\rho = \Theta(1)$,

$$\min \quad \frac{1}{2} \sum_{(u,v)\in E(G)} \|\bar{u} - \bar{v}\|^2$$

subject to

$$\text{for all } u \in V, \quad \sum_{v} \langle \bar{u}, \bar{v} \rangle \le \rho |V| \qquad \text{(Spreading Constraints)}$$

$$\text{for all } u, v, w \in V, \quad \|\bar{u} - \bar{v}\|^2 + \|\bar{v} - \bar{w}\|^2 \ge \|\bar{u} - \bar{v}\|^2 \qquad (\ell_2^2\text{-triangle inequalities})$$

$$\text{for all } u, v \in V, \quad \langle \bar{u}, \bar{v} \rangle \ge 0$$

$$\text{for all } u \in V, \quad \|\bar{u}\|^2 = 1.$$

Definition 6.2 (Heavy Set, $H_{\delta,\varphi}(M)$ [36]). Let V be a set of n vertices and $M \subseteq N$. Consider an SDP solution $\varphi : V \to \mathcal{H}$. We say that a vertex $u \in M$ is δ-heavy in M if the ℓ_2^2-ball of radius δ around $\varphi(u)$ contains at least $\delta^2 n$ vectors from $\varphi(M)$, i.e., $|\{v \in M | \varphi(v) \in \mathrm{Ball}(\varphi(u), \delta)\}| \ge \delta^2 n$. We denote the set of all vertices that are δ-heavy in M by $H_{\delta,\varphi}(M)$.

Definition 6.3 (Geometric Expansion [36]). A graph $G = (V, E)$ satisfies the geometric expansion property with cut value X at scale δ if for every SDP solution $\varphi : V \to \mathcal{H}$ satisfying $H_{\delta,\varphi}(V) = \emptyset$ (recall that $H_{\delta,\varphi}(V)$ is the set of δ-heavy vertices in V):

$$\left| \{(u, v) \in E | \, \|\varphi(u) - \varphi(v)\|^2 \le \delta/2\} \right| \le 2\delta^2 X.$$

A graph $G = (V, E)$ satisfies the geometric expansion property with cut value X up to scale 2^{-T} for $T \in \mathbb{N}$ if it satisfies the geometric expansion property for every $\delta \in \{2^{-t} | 1 \le t \le T\}$.

Theorem 6.4 (Theorem 3.4 from Reference [36]). *Let $G = (V, E)$ be a graph that satisfies the geometric expansion property with cut value X at scale up to $c\sqrt{\log |V|}$. Then, there exists a randomized polynomial time that with high probability outputs a partition L, R of V such that $|\mathrm{cut}(L, R)| = O(X)$ and $|L|, |R| \ge |V|/3$.*

6.2.2 Geometric Expansion of HSM. Let $\bar{G}_n = (\bar{V}_n, \bar{E}_n, w)$ be a graph generated according to an ultrametric, where for each $e \in \bar{E}_n$, $w(e) \in (0, 1)$. In this case, we allow $w(e)$ to depend on n—in particular it is possible that $w(e) \to 0$ as $n \to \infty$. Let $G = (V, E)$ be an unweighted random graph with $|V| = |\bar{V}_n| = n$ generated from $\bar{G}$ as follows. An edge (u, v) is added to G with probability $w((u, v))$, for the corresponding vertices $u, v \in \bar{V}_n$. Note that this is simply a hierarchical stochastic models (Definition 5.1).

For the rest of this discussion we assume that $V = \bar{V}_n$ as a natural bijection exists between the two vertex sets. Let $\widetilde{T}$ be a generating tree for $\bar{G}$. Let $U \subseteq V$ and let $\widetilde{T}|_U$ be the restriction of $\widetilde{T}$ to leaves in U (removing unnecessary leaves and reducing internal nodes as necessary). Let $N(U)$ be the root of $\widetilde{T}|_U$. Consider the following procedure where the nodes appear as leaves in the left and right subtrees of the root of $\widetilde{T}|_U$. Suppose we follow the convention that the left subtree is never any smaller than the right subtree in $\widetilde{T}|_U$. We say that the canonical node of $\widetilde{T}|_U$ is the first left node N_L encountered in a top-down traversal starting from $N(U)$ such that $2|U|/3 \ge V(N_L) \ge |N|/3$. We define $U_L = V(N_L)$, and $U_R = U \setminus U_L$. We say that (U_L, U_R) is the *canonical cut* of U. It is easy to see that such a cut always exists since the tree is binary and left subtrees are never smaller than right subtrees. Let $E_{\mathrm{rnd}} = \{(u, v) \,|\, u \in U_L, v \in U_R\}$.

Lemma 6.5. *For a random graph G generated as described in Theorem 6.1, with probability at least $1 - o(1)$, for every subset U of size at least $n^{2/3}\sqrt{\log n}$, the subgraph (U, E_{rnd}) is geometrically*

expanding with cut cost

$$X = C \cdot \max\{w(L,R), |U| \cdot D \cdot \log^2 D, |U| \cdot D \cdot \log n\} \tag{15}$$

up to scale $1/\sqrt{D}$. *Furthermore, the result also applies in the semi-random setting where an adversary may remove any subset of edges from the random graph* G.

Before we give the proof of Lemma 6.5, we show how it implies Theorem 6.1.

Proof of Theorem 6.1. We apply Theorem 4.1, where it is essentially established that provided one obtains a ϕ approximation to the 1/3-balanced min-cut problem (i.e., minimize cut subject to the constraint that both sides have at least 1/3 of the vertices being cut), the recursive algorithm gives a $O(\phi)$ approximation for minimizing Dasgupta's cost function.

We observe that $\text{cost}(\widetilde{T}; \bar{G}) = \Omega(n^3 \cdot p_{\min}) = \Omega(n^{7/3} \log n)$. Thus, we notice that once we obtain sets U of size $n_0 = n^{2/3}\sqrt{\log n}$, since there are at most n/n_0 of them, even if we use an arbitrary tree on any such U, together this can only add $O(\frac{n}{n_0} \cdot n_0^3) = O(n^{7/3} \cdot \log n)$. Thus, we only need to be able to obtain suitable approximations during the recursive procedure as long as $|U| \geq n^{2/3} \log n$. This is precisely given by using Lemma 6.5. Observe that in Equation (15), $w(L,R) = \Omega(|U|^2 \cdot p_{\min}) = \Omega(n^{2/3} \log^2 n)$, $|U|D \log^2 D = o(|U|D \log n)$, and $D|U| \log n = O(n^{2/3} \log^2 n)$. Thus, the algorithm of Reference [36] given by Theorem 6.4 returns a cut that is a constant factor approximation to the 1/3-balanced min-cut problem on the induced subgraph of $\bar{G}$ on the vertex set U. This observation together with a slight modification of the charging argument in the proof of Theorem 4.1 to account for the case where subgraphs have size less than $n^{2/3} \log n$ finishes the proof. □

In the following, we give the deferred proof of Lemma 6.5. The proof is essentially identical to that of Theorem 5.1 in Reference [36]. However, as there are some minor modifications, we are unable to cite their result directly and hence provide the entire proof here for completeness.

Proof of Lemma 6.5. Fix some subset U and let U_L, U_R be the canonical cut of U given by the generating tree $\widetilde{T}$ of $\bar{G}$. Let $E_{\text{all}} = \{(u,v) | u \in U_L, v \in U_R\}$ and let $E_{\text{rnd}} = E_{\text{all}} \cap E$, where E is the set of edges in the realized random graph $G = (V, E)$. As the adversary in the semi-random graph can only remove edges it suffices to show that (U, E_{rnd}) is geometrically expanding with high probability. We fix the parameter $\delta = 2^{-t}$ (where $1 \leq t \leq T$) and prove that the graph (U, E_{rnd}) is geometrically expanding with cut value X at scale δ. The probability that this fails to happen will be low enough for us to take a simple union bound over all the possible values of δ.

The condition $H_{\delta,\varphi}(U) = \emptyset$ implies that

$$|\{(u,v) \in U \times U \mid \|\varphi(u) - \varphi(v)\|^2 \delta\}| \leq \delta^2 n^2.$$

We need to bound the probability of the bad event, the existence of an SDP solution $\varphi : U \to \mathcal{H}$ such that

$$|\{(u,v) \in U \times U \mid \|\varphi(u) - \varphi(v)\|^2 \leq \delta\}| \leq \delta^2 n^2, \tag{16}$$

$$\left|\left\{(u,v) \in E_{\text{rnd}} \mid \|\varphi(u) - \varphi(v)\|^2 \leq \frac{\delta}{2}\right\}\right| \geq 2\delta^2 X. \tag{17}$$

Makarychev et al. [36] show that if φ satisfying Equations (16) and (17) exists, then provided $|E_{\text{rnd}}| \leq 2X$, there exists $\varphi' : U \to N_\delta$ satisfying:

$$\left|\left\{(u,v) \in U \times U \mid \|\varphi'(u) - \varphi'(v)\|^2 \leq \frac{3}{4}\delta\right\}\right| \leq \frac{5}{4}\delta^2 n^2, \tag{18}$$

$$\left|\left\{(u,v) \in E_{\text{rnd}} \mid \|\varphi'(u) - \varphi'(v)\|^2 \leq \frac{3}{4}\delta\right\}\right| \geq \frac{3}{4}\delta^2 X, \tag{19}$$

where $N_\delta \subset \mathcal{H}$ is a set of size $\exp(O(\log^2 \delta^{-1}))$.

The remainder of the proof is showing that the existence of φ' is a very low-probability event. First, as $|E_{\text{rnd}}| = w(U_L, U_R) \leq X$, $\mathbb{P}[|E_{\text{rnd}}| \geq 2X] \leq e^{-c_0 X}$. Note that if we fix a $\varphi' : U \to N_\delta$, the probability (over the random choice of E_{rnd}) that Equations (18) and (19) hold is at most $e^{-c_1 \delta^2 X}$, by using the Chernoff bound. Finally, we note that there are at most $|N_\delta|^{|U|}$ such φ', thus we can safely take a union bound provided $X/D \geq c_3 \cdot |U| \log^2 D$. Finally, there are $n^{|U|}$ subsets of size $|U|$ and again we can safely apply a union bound provided $X/d \geq c_4 |U| \log n$. The choice of X ensures that this happens. □

APPENDICES

A PERFECT GROUND-TRUTH INPUTS AND BEYOND

In this section, we focus on ground-truth inputs. We state and (re)prove results that when the input is a perfect ground-truth input, commonly used algorithms (single-linkage, average-linkage, and complete-linkage; as well as some divisive algorithms—the bisection 2-Center and sparsest-cut algorithms) yield a tree of optimal cost, hence (by Definition 3.1) a ground-truth tree. Several of these results are folklore (and straightforward when there are no ties), but we have been unable to pin down a reference, so we include them for completeness (Section A.1). We also introduce a faster optimal algorithm for "strict" ground-truth inputs (Section A.2). The proofs present no difficulty. The meat of this section is Section A.3, where we go beyond ground-truth inputs; we introduce δ-adversarially perturbed ground-truth inputs and design a simple, more robust algorithm that, *for any* admissible objective function, yields a δ-approximation.

ALGORITHM 6: Linkage Algorithm for Hierarchical Clustering (similarity setting)

Input: A graph $G = (V, E)$ with edge weights $w : E \mapsto \mathbb{R}_+$
Create n singleton trees. Root labels: $\mathcal{C} = \{\{v_1\}, \ldots, \{v_n\}\}$
Define $\text{sim} : \mathcal{C} \times \mathcal{C} \mapsto \mathbb{R}_+$: $\text{sim}(C_1, C_2) = \begin{cases} \frac{1}{|C_1||C_2|} \sum_{x \in C_1, y \in C_2} w((x,y)) & \textbf{Average-Linkage} \\ \max_{x \in C_1, y \in C_2} w((x,y)) & \textbf{Single-Linkage} \\ \min_{x \in C_1, y \in C_2} w((x,y)) & \textbf{Complete-Linkage} \end{cases}$.
while there are at least two trees **do**
    Take the two trees with root labels C_1, C_2 such that $\text{sim}(C_1, C_2)$ is maximum
    Create a new tree by making those two tree children of a new root node labeled $C_1 \cup C_2$
    Remove C_1, C_2 from $\mathcal{C}$, add $C_1 \cup C_2$ to $\mathcal{C}$, and update sim
return the resulting binary tree T

A.1 Perfect Ground-Truth Inputs Are Easy

In the following, we refer to the *tie breaking* rule of Algorithm 6 as the rule followed by the algorithm for deciding which of C_i, C_j or C_k, C_ℓ to merge, when $\max_{C_1, C_2 \in \mathcal{C}} \text{sim}(C_1, C_2) = \text{sim}(C_i, C_j) = \text{sim}(C_k, C_\ell)$.

Theorem A.1. [12] *Assume that the input is a (dissimilarity or similarity) ground-truth input. Then, for any admissible objective function, the agglomerative heuristics average-linkage, single-linkage, and complete-linkage (see Algorithm 6) return an optimal solution. This holds regardless of the tie breaking rule used by Algorithm 6.*

[12] This theorem may be folklore, at least when there are no ties, but we have been unable to find a reference.

Proof. We focus on the similarity setting; the proof for the dissimilarity setting is almost identical. We define the *candidate trees* after t iterations of the while loop to be sets of trees in C at that time. The theorem follows from the following statement, which we will prove by induction on t: If $C^t = \{C_1, \ldots, C_k\}$ denotes the set of clusters after t iterations, then there exists a generating tree T^t for G, such that the candidate trees are subtrees of T^t.

For the base case, note that initially each candidate tree contains exactly one vertex and the statement holds. For the general case, let C_1, C_2 be the two trees that constitute the tth iteration. By induction, there exists a generating tree T^{t-1} for G, and associated weights W^{t-1} (according to Definition 2.2) such that C_1 and C_2 are subtrees of T^{t-1}, rooted at nodes N^1 and N^2 of T^{t-1} respectively.

To define T^t, we start from T^{t-1}. Consider the path $P = \{N^1, N_1, N_2, \ldots, N_k, N^2\}$ joining N^1 to N^2 in T^{t-1} and let $N_r = \text{LCA}_{T^{t-1}}(N^1, N^2)$. If N_r is the parent of N^1 and N^2, then $T^t = T^{t-1}$, else do the following transformation: remove the subtrees rooted at N^1 and at N^2; create a new node N^* as the second child of N_k, and let N_1 and N_2 be its children. This defines T^t. To define W^t, extend W^{t-1} by setting $W^t(N^*) = W(N_r)$.

Claim 2. *For any $N_i, N_j \in P$, $W^{t-1}(N_i) = W^{t-1}(N_j)$.*

Thanks to the inductive hypothesis, with Claim 2 it is easy to verify that W^t certifies that T^t is generating for G. □

Proof of Claim 2. Fix a node N_i on the path from N_r to N^1 (the argument for nodes on the path from N_r to N^2 is similar). By induction $W^{t-1}(N_i) \geq W^{t-1}(N_r)$. We show that since the linkage algorithms merge the trees C_1 and C_2, we also have $W^{t-1}(N_i) \leq W^{t-1}(N_r)$ and so $W^{t-1}(N_i) = W^{t-1}(N_r)$, hence the claim. Let $w_0 = W^{t-1}(N_r)$.

By induction, for all $u \in C_1, v \in C_2$, $w(u, v) = w_0$, and thus $\text{sim}(C_1, C_2) = w_0$ in the execution of all the algorithms. Fix a candidate tree $C' \in C^t$, $C' \neq C_1, C_2$ and $C' \subseteq V(N_i)$. Since C is a partition of the vertices of the graph and since candidate trees are subtrees of T^{t-1}, such a cluster exists. Thus, for $u \in C_1, v \in C' w(u, v) = W^{t-1}(\text{LCA}_{T^{t-1}}(u, v)) = W^{t-1}(N_i) \geq w_0$, since N_i is a descendant of N_r.

It is easy to check that by their definitions, for any of the linkage algorithms, we thus have that $\text{sim}(C_1, C') \geq w_0 = \text{sim}(C_1, C_2)$. But since the algorithms merge the clusters with maximum pairwise similarity, it follows that $\text{sim}(C_1, C') \leq \text{sim}(C_1, C_2) = w_0$ and therefore, $W^{t-1}(N_i) \leq W^{t-1}(N_r)$ and so, $W^{t-1}(N_i) = W^{t-1}(N_r)$ and the claim follows. This is true no matter the tie breaking chosen for the linkage algorithms. □

Divisive Heuristics. We now focus on two well-known divisive heuristics: (1) the bisection 2-Center which uses a partition-based clustering objective (the k-Center objective) to divide the input into two (non necessarily equal-size) parts (see Algorithm 7), and (2) the recursive sparsest-cut algorithm, which can be implemented efficiently for ground-truth inputs (Lemma A.4).

ALGORITHM 7: Bisection 2-Center (similarity setting)

1: **Input:** A graph $G = (V, E)$ and a weight function $w : E \mapsto \mathbb{R}_+$
2: Find $\{u, v\} \subseteq V$ that maximizes $\min_x \max_{y \in \{u,v\}} w(x, y)$
3: $A \leftarrow \{x \mid w(x, u) \geq \max_{y \in \{u,v\}} w(x, y)\}$
4: $B \leftarrow V \setminus A$.
5: Apply Bisection 2-Center on $G[A]$ and $G[B]$ to obtain trees T_A,T_B respectively
6: **return** The union tree of T_A, T_B.

Loosely speaking, we show that this algorithm computes an optimal solution if the optimal solution is unique. More precisely, for any similarity graph G, we say that a tree T is *strictly generating* for G if there exists a weight function W such that for any nodes N_1, N_2, if N_1 appears on the path from N_2 to the root, then $W(N_1) < W(N_2)$ and for every $x, y \in V$, $w(x, y) = W(\mathrm{LCA}_T(x, y))$. In this case we say that the input is a strict ground-truth input. In the context of dissimilarity, an analogous notion can be defined and we obtain a similar result.

Theorem A.2. [13] *For any admissible objective function, the bisection 2-Center algorithm returns an optimal solution for any similarity or dissimilarity graph G that is a strict ground-truth input.*

Proof. We proceed by induction on the number of nodes in the graph. Consider a strictly generating tree T and the corresponding weight function W. Consider the root node N_r of T and let N_1, N_2 be the children of the root. Let (α, β) be the cut induced by the root node of T (i.e., $\alpha = V(N_1)$, $\beta = V(N_2)$). Define w_0 to be the weight of an edge between $u \in \alpha$ and $v \in \beta$ for any u, v (recall that since T is strictly generating all the edges between α and β are of same weight). We show that the bisection 2-Centers algorithm divides the graph into α and β. Applying the inductive hypothesis on both subgraphs yields the result.

Suppose that the algorithm locates the two centers in β. Then, $\min_{x \in \alpha} \max_{y \in \{u, v\}} w(x, y) = w_0$, since the vertices of α are connected by an edge of weight w_0 to the centers. Thus, the value of the clustering is w_0. Now, consider a clustering consisting of a center c_0 in α and a center c_1 in β. Then, for each vertex u, we have $\max_{c \in \{c_0, c_1\}} w(u, c) \geq \min(W(N_1), W(N_2)) > W(N_r) = w_0$, since T and W are strictly generating; hence, a strictly better clustering value. Therefore, the algorithm locates $x \in \alpha$ and $y \in \beta$. Finally, it is easy to see that the partitioning induced by the centers yields parts $A = \alpha$ and $B = \beta$. □

Remark 4. To extend our result to (non-strict) ground-truth inputs, one could consider the following variant of the algorithm (which bears similarities with the popular *elbow method* for partition-based clustering): Compute a k-Center clustering for all $k \in \{1, \ldots, n\}$ and partition the graph according to the k-Center clustering of the smallest $k > 1$ for which the value of the clustering increases. Mimicking the proof of Theorem A.2, one can show that the tree output by the algorithm is generating.

We now turn to the recursive sparsest-cut algorithm (i.e., the recursive ϕ-sparsest-cut algorithm of Section 4.1, for $\phi = 1$). The recursive sparsest-cut consists of recursively partitioning the graph according to a sparsest cut of the graph. We show (1) that this algorithm yields a tree of optimal cost and (2) that computing a sparsest cut of a similarity graph generated from an ultrametric can be done in linear time. Finally, we observe that the analogous algorithm for the dissimilarity setting consists of recursively partitioning the graph according to the densest cut of the graph and achieves similar guarantees (and similarly the densest cut of a dissimilarity graph generated from an ultrametric can be computed in linear time).

Theorem A.3. [14] *For any admissible objective function, the recursive sparsest-cut (respectively densest-cut) algorithm computes a tree of optimal cost if the input is a similarity (respectively dissimilarity) ground-truth input.*

Proof. The proof is by induction and presents no difficulty; it may be easier to recreate it than to read it.

Let T be a generating tree and W be the associated weight function. Let N_r be the root of T, N_1, N_2 the children of N_r, and $(\alpha = V(N_1), \beta = V(N_2))$ the induced root cut. Since T is generating,

[13] This theorem may be folklore, but we have been unable to find a reference.
[14] This Theorem may be folklore, at least when there are no ties, but we have been unable to find a reference.

all the edges between α and β are of same weight w, which is therefore also the sparsity of (α, β). For every edge (u, v) of the graph, $w(u, v) = W(\mathrm{LCA}_T(u, v)) \geq w$, so every cut has sparsity at least w, so (α, β) has minimum sparsity.

Now consider the tree T^* computed by the algorithm, and let (γ, δ) denote the sparsest cut used by the algorithm at the root (in case of ties it might not be different from (α, β)). By induction, the algorithm on $G[\gamma]$ and $G[\delta]$ gives two generating trees T_γ and T_δ with associated weight functions W_γ and W_δ. To argue that T^* is generating, we define W^* as follows, where N_r^* denotes the root of T^*,

$$W^*(N) = \begin{cases} W_\gamma(N) & \text{if } N \in T_\gamma \\ W_\delta(N) & \text{if } N \in T_\delta \\ w & \text{if } N = N_r^* \end{cases}.$$

By induction $w(u, v) = W(\mathrm{LCA}_T(u, v))$ if either both $u, v \in \gamma$, or both $u, v \in \delta$. For any $u \in \gamma, v \in \delta$, we have $w(u, v) = w = W(N_r^*) = W(\mathrm{LCA}_T(u, v))$. Finally, since $w \leq w(u, v)$ for any u, v, we have $W(N_r^*) = w \leq W(N)$, for any $N \in T^*$, and therefore T^* is generating. □

We then show how to compute a sparsest cut of a graph that is a ground-truth input.

Lemma A.4. *If the input graph is a ground-truth input, then the sparsest cut is computed in $O(n)$ time by the following algorithm: Pick an arbitrary vertex u, let $w_{\min}$ be the minimum weight of edges adjacent to u, and partition V into $A = \{x \mid w(u, x) > w_{\min}\}$ and $B = V \setminus A$.*

Proof. Let $w_{\min} = w(u, v)$. We show that $w(A, B)/(|A||B|) = w_{\min}$ and, since $w_{\min}$ is the minimum edge weight of the graph, that the cut (A, B) only contains edges of weight $w_{\min}$. Fix a generating tree T. Consider the path from u to the root of T and let N_0 be the first node on the (bottom-up) path such that $W(N_0) = w_{\min}$. For any vertex $x \in A$, we have that $w(u, x) > w_{\min}$. Hence by definition, we have that N_0 is an ancestor of $\mathrm{LCA}_T(u, x)$. Therefore, for any other node y such that $w(u, y) = w_{\min}$, we have $\mathrm{LCA}_T(u, y) = \mathrm{LCA}_T(x, y)$ and so, $w(x, y) = W(\mathrm{LCA}_T(x, y)) = W(\mathrm{LCA}_T(u, y)) = w_{\min}$. It follows that all the edges in the cut (A, B) are of weight $w_{\min}$ and so, the cut is a sparsest cut. □

A.2 A Near-Linear Time Algorithm

In this section, we propose a simple, optimal, algorithm for computing a generating tree of a ground-truth input. For any graph G, the running time of this algorithm is $O(n^2)$, and $\widetilde{O}(n)$ if there exists a tree T that is *strictly generating* for the input. For completeness we recall that for any graph G, we say that a tree T is strictly generating for G if there exists a weight function W such that for any nodes N_1, N_2, if N_1 appears on the path from N_2 to the root, then $W(N_1) < W(N_2)$ and for every $x, y \in V$, $w(x, y) = W(\mathrm{LCA}_T(x, y))$. In this case we say that the input is a *strict ground-truth input.*

The algorithm is described for the similarity setting but could be adapted to the dissimilarity case to achieve the same performances.

Theorem A.5. *For any admissible objective function, Algorithm 8 computes a tree of optimal cost in time $O(n \log^2 n)$ with high probability if the input is a strict ground-truth input or in time $O(n^2)$ if the input is a (non-necessarily strict) ground-truth input.*

Proof. We proceed by induction on the number of vertices in the graph. Let p be the first pivot chosen by the algorithm and let $B_1, \ldots, B_k$ be the sets defined by p at Step 4 of the algorithm, with $w(p, u) > w(v, p)$, for any $u \in B_i, v \in B_{i+1}$.

We show that for any $u \in B_i, v \in B_j$, $j > i$, we have $w(u, v) = w(p, v)$. Consider a generating tree T and define $N_1 = \mathrm{LCA}_T(p, u)$ and $N_2 = \mathrm{LCA}_T(p, v)$. Since T, h, σ is generating and

ALGORITHM 8: Fast and Simple Algorithm for Hierarchical Clustering on Perfect Data (similarity setting)

Input: A graph $G = (V, E)$ and a weight function $w : E \mapsto \mathbb{R}_+$
$p \leftarrow$ random vertex of V
Let $w_1 > \cdots > w_k$ be the edge weights of the edges that have p as an endpoint
Let $B_i = \{v \mid w(p, v) = w_i\}$, for $1 \le i \le k$.
Apply the algorithm recursively on each $G[B_i]$ and obtain a collection of trees $T_1, \ldots, T_k$
Define T_0^* as a tree with p as a single vertex
For any $1 \le i \le k$, define T_i^* to be the union of T_{i-1}^* and T_i
return T_k^*

$w(p, u) > w(p, v)$, we have that N_2 is an ancestor of N_1, by Definition 2.2. Therefore, $\mathrm{LCA}_T(u, v) = N_2$, and so $w(u, v) = W(N_2) = w(p, v)$. Therefore, combining the inductive hypothesis on any $G[B_i]$ and by Definition 2.2 the tree output by the algorithm is generating.

A bound of $O(n^2)$ for the running time follows directly from the definition of the algorithm. We now argue that the running time is $O(n \log^2 n)$ with high probability if the input is strictly generated from a tree T. First, it is easy to see that a given recursive call on a subgraph with n_0 vertices takes $O(n_0)$ time (excluding time required for further recursive calls). Now, observe that if at each recursive call the pivot partitions the n_0 vertices of its subgraph into buckets of size at most $2n_0/3$, then applying the master theorem implies a total running time of $O(n \log n)$. Unfortunately, there are trees where picking an arbitrary vertex as a pivot yields a single bucket of size $n - 1$.

Thus, consider the node N of T that is the first node reached by the walk from the root that always goes to the child tree with the higher number of leaves, stopping when the subtree of T rooted at N contains fewer than $2n/3$ but at least $n/3$ leaves. Since T is strictly generating we have that the partition into $B_1, \ldots, B_k$ induced by any vertex $v \in V(N)$ is such that any B_i contains less than $2n/3$ vertices. Indeed, for any u such that $\mathrm{LCA}_T(u, v)$ is an ancestor of N and $x \in V(N)$, we have that $w(u, v) < w(x, v)$, and so u and x belong to different parts of the partition $B_1, \ldots, B_k$.

Since the number of vertices in $V(N)$ is at least $n/3$, the probability of picking one of them is at least $1/3$. Therefore, since the pivots are chosen independently, after $c \log n$ recursive calls, the probability of not picking a vertex of $V(N)$ as a pivot is $1/n^{\Omega(c)}$ for some large-enough constant c. Taking the union bound yields the theorem. □

A.3 Beyond Structured Inputs

Since real-world inputs are unlikely to correspond exactly to our definition of ground-truth inputs introduced in Section 2, we introduce the notion of δ-adversarially perturbed ground-truth inputs. This notion aims at accounting for noise in the data. We then design a simple and arguably more reliable algorithm (a robust variant of Algorithm 8) that achieves a δ-approximation for δ-adversarially perturbed ground-truth inputs in $O(n(n + m))$ time. An interesting property of this algorithm is that its approximation guarantee is the same for any admissible objective function.

We first introduce the definition of δ-adversarially perturbed ground-truth inputs. For any real $\delta \ge 1$, we say that a weighted graph $G = (V, E, w)$ is a *δ-adversarially perturbed ground-truth input* if there exists an ultrametric (X, d), such that $V \subseteq X$, and for every $x, y \in V, x \ne y$, $e = \{x, y\}$ exists, and $f(d(x, y)) \le w(e) \le \delta f(d(x, y))$, where $f : \mathbb{R}_+ \to \mathbb{R}_+$ is a non-increasing function. This defines δ-adversarially perturbed ground-truth inputs for similarity graphs and an analogous definition applies for dissimilarity graphs.

We now introduce a robust, simple version of Algorithm 8 that returns a δ-approximation if the input is a δ-adversarially perturbed ground-truth inputs. Algorithm 8 was partitioning the input

graph based on a single, random vertex. In this slightly more robust version, the partition is built iteratively: Vertices are added to the current part if there exists at least one vertex in the current part or in the parts that were built before with which they share an edge of high-enough weight (see Algorithm 9 for a complete description).

ALGORITHM 9: Robust and Simple Algorithm for Hierarchical Clustering on δ-adversarially perturbed ground-truth inputs (similarity setting)

Input: A graph $G = (V, E)$, a weight function $w : E \mapsto \mathbb{R}_+$, a parameter δ
$p \leftarrow$ arbitrary vertex of V
$i \leftarrow 0$
$\tilde{V}_i \leftarrow \{p\}$
while $\tilde{V}_i \neq V$ **do**
  Let $p_1 \in \tilde{V}_i, p_2 \in V \setminus \tilde{V}_i$ s.t. (p_1, p_2) is an edge of maximum weight in the cut $(\tilde{V}_i, V \setminus \tilde{V}_i)$
  $w_i \leftarrow w(p_1, p_2)$
  $B_i \leftarrow \{u \mid w(p_1, u) = w_i\}$
  while $\exists u \in V \setminus (\tilde{V}_i \cup B_i)$ s.t. $\exists v \in B_i \cup \tilde{V}_i$, $w(u, v) \geq w_i$ **do**
    $B_i \leftarrow B_i \cup \{u\}$.
  $\tilde{V}_{i+1} \leftarrow \tilde{V}_i \cup B_i$
  $i \leftarrow i + 1$
Let $B_1, \ldots, B_k$ be the sets obtained
Apply the algorithm recursively on each $G[B_i]$ and obtain a collection of trees $T_1, \ldots, T_k$
Define T_0^* as a tree with p as a single vertex
For any $1 \leq i \leq k$, define T_i^* to be the union of T_{i-1}^* and T_i
return T_k^*

Theorem A.6. *For any admissible objective function, Algorithm 9 returns a δ-approximation if the input is a δ-adversarially perturbed ground-truth input.*

To prove the theorem we introduce the following lemma whose proof is temporarily deferred. The lemma states that the tree built by the algorithm is almost generating (up to a factor of δ in the edge weights).

Lemma A.7. *Let T be a tree output by Algorithm 9, let $\mathcal{N}$ be the set of internal nodes of T. For any node N with children N_1, N_2 there exists a function $\omega : \mathcal{N} \mapsto \mathbb{R}_+$, such that for any $u \in V(N_1), v \in V(N_2)$, $\omega(N) \leq w(u, v) \leq \delta\omega(N)$. Moreover, for any nodes N, N', if N' is an ancestor of N, we have that $\omega(N) \geq \omega(N')$.*

Assuming Lemma A.7, the proof of Theorem A.6 is as follows.

Proof of Theorem A.6. Let $G = (V, E)$, $w : E \mapsto \mathbb{R}_+$ be the input graph and T^* be a tree of optimal cost. By Lemma A.7, the tree T output by the algorithm is such that for any node N with children N_1, N_2 there exists a real $\omega(N)$, such that for any $u \in V(N_1), v \in V(N_2)$, $\omega(N) \leq w(u, v) \leq \delta\omega(N)$. Thus, consider the slightly different input graph $G' = (V, E, w')$, where $w' : E \mapsto \mathbb{R}_+$ is defined as follows. For any edge (u, v), define $w'(u, v) = \omega(\mathrm{LCA}_T(u, v))$. Since by Lemma A.7, for any nodes N, N' of T, if N' is an ancestor of N, we have that $\omega(N) \geq \omega(N')$ and by Definition 2.2, T is generating for G'. Thus, for any admissible cost function, we have that for G', $\mathrm{cost}(T; G') \leq \mathrm{cost}(T^*; G')$.

Finally, observe that for any edge e, we have $w'(e) \leq w(e) \leq \delta w'(e)$. It follows that $\text{cost}(T;G) \leq \delta \text{cost}(T;G')$ for any admissible cost function and $\text{cost}(T^*;G') \leq \text{cost}(T^*;G)$. Therefore, $\text{cost}(T;G) \leq \delta \text{cost}(T^*;G) = \delta \text{OPT}$. □

Proof of Lemma A.7. We proceed by induction on the number of vertices in the graph (the base case is trivial). Consider the first recursive call of the algorithm. We show the following claim.

Claim 3. *For any $1 \leq i \leq k$, for any $y \in \widetilde{V}_i$, $x \in B_i$, $w_i \geq w(x,y) \geq w_i/\delta$. Additionally, for any $x, y \in B_i$, $w(x,y) \geq w_i/\delta$.* □

We first argue that Claim 3 implies the lemma. Let T be the tree output by the algorithm. Consider the nodes on the path from p to the root of T; let N_i denote the node whose subtree is the union of T^*_{i-1} and T_i. By definition, $V(T^*_{i-1}) = \widetilde{V}_i$ and $V(T_i) = B_i$. Applying Claim 3 and observing that $w_i > w_{i+1}$ implies that the lemma holds for all the nodes on the path. Finally, since for any edge $\{u,v\}$, for $u, v \in B_i$, we also have $w(u,v) \geq w_i/\delta$, combining with the inductive hypothesis on B_i implies the lemma for all the nodes of the subtree T_i.

Proof of Claim 3. Let (X,d) and f be a pair of ultrametric and function that is generating for G. Fix $i \in \{1,\ldots,k\}$. For any vertex $x \in B_i$, let $\sigma(x)$ denote a vertex y that is in $\widetilde{V}_i$ or inserted to B_i before x and such that $w(y,x) \geq w_i$. For any vertex v, let $\sigma^i(x)$ denotes the vertex obtained by applying σ i times to x (i.e., $\sigma^2(x) = \sigma(\sigma(x))$). By definition of the algorithm, it holds that for any $x \in B_i$, $\exists s \geq 1$, such that $\sigma^s(x) \in \widetilde{V}_i$.

Fix $x \in B_i$. For any $y \in \widetilde{V}_i$, we have that $w(y,x) \leq w_i$, since, otherwise, the algorithm would have added x before.

Now, let $y \in \widetilde{V}_i$ or y be inserted to B_i prior to x. We show that $w(y,x) \geq w_i/\delta$. Observe that since X, d is an ultrametric, $d(x,y) \leq \max(d(x,\sigma(x)), d(\sigma(x),y))$.

We now "follow" σ by applying the function σ to $\sigma(x)$ and repeating until we reach $\sigma^\ell(x) = z \in \widetilde{V}_i$, for some ℓ. Combining this with the definition of an ultrametric, it follows that

$$d(x,y) \leq \max(d(x,\sigma(x)), d(\sigma(x),\sigma^2(x)), \ldots, d(\sigma(x)^{\ell-1}, z), d(z,y)).$$

If y was in $\widetilde{V}_i$, then we define $\hat{y} = y$. Otherwise y is also in B_i (and so was added to B_i before x). We then proceed similarly as for x and "follow" σ. In this case, let $\hat{y} = \sigma^k(y) \in \widetilde{V}_i$, for some k. Applying the definition of an ultrametric again, we obtain

$$d(x,y) \leq \max(d(x,\sigma(x)), d(\sigma(x),\sigma^2(x)), \ldots, d(\sigma^{\ell-1}, z), d(z,\hat{y}), d(y,\sigma(y)), \ldots, d(\sigma^{k-1}(y),\hat{y})).$$

Assume for now that $d(z,\hat{y})$ is not greater than the others. Applying the definition of a δ-adversarially perturbed input, we have that

$$\delta w(x,y) \geq \min(\ldots, w(\sigma^a(x), \sigma^{a+1}(x)), \ldots, w(\sigma^b(y), \sigma^{b+1}(y)), \ldots).$$

Following the definition of σ, we have for all v that $w(v,\sigma(v)) \geq w_i$. Therefore, we conclude $\delta w(x,y) \geq w_i$.

We thus turn to the case where $d(z,\hat{y})$ is greater than the others. Since both $z, \hat{y} \in \widetilde{V}_i$, we have that they belong to some B_{j_0}, B_{j_1}, where $j_0, j_1 < i$. We consider the minimum j such that a pair at distance at least $d(z,\hat{y})$ was added to $\widetilde{V}_j$. Consider such a pair $u, v \in \widetilde{V}_j$ satisfying $d(u,v) \geq d(z,\hat{y})$ and suppose w.l.o.g. that $v \in B_{j-1}$ (we could have either $u \in B_{j-1}$ or $u \in \widetilde{V}_{j-1}$). Again, we follow the path $\sigma(v), \sigma(\sigma(v)), \ldots$, until we reach $\sigma^{r_1}(v) \in \widetilde{V}_{j-1}$, for some r_1, and similarly for u: $\sigma^{r_2}(u) \in \widetilde{V}_{j-1}$, for some r_2. Applying the definition of an ultrametric this yields that

$$d(u,v) \leq \max(\ldots, d(\sigma^a(u), \sigma^{a+1}(u)), \ldots, d(\sigma^b(v), \sigma^{b+1}(v)), \ldots, d(\sigma^{r_1}(v), \sigma^{r_2}(u))). \quad (20)$$

Now the difference is that $\widetilde{V}_{j-1}$ does not contain any pair at distance at least $d(z, \hat{y})$. Therefore, we have $d(\sigma^{r_1}(v), \sigma^{r_2}(u)) < d(z, \hat{y})$. Moreover, recall that by definition of u, v, $d(z, \hat{y}) \leq d(u, v)$. Thus, $d(\sigma^{r_1}(v), \sigma^{r_2}(u))$ is not the maximum in Equation (20), since it is smaller than the left-hand side. Simplifying Equation (20) yields

$$d(x, y) < d(z, \hat{y}) \leq d(u, v) \leq \max(\ldots, d(\sigma^a(u), \sigma^{a+1}(u)), \ldots, d(\sigma^b(v), \sigma^{b+1}(v)), \ldots).$$

By definition of a δ-adversarially perturbed input, we obtain $\delta w(x, y) \geq \min_\ell w(\sigma^\ell(b), \sigma^{\ell+1}(b)) \geq w_j$. Now, it is easy to see that for $j < i$, $w_i < w_j$ and therefore $\delta w(x, y) \geq w_i$.

We conclude that for any $y \in \widetilde{V}_i$, $x \in B_i$, $w_i \geq w(x, y) \geq w_i/\delta$ and for $x, y \in B_i$, we have that $w(x, y) \geq w_i/\delta$, as claimed. □

B WORST-CASE ANALYSIS OF COMMON HEURISTICS

The results presented in this section show that for both the similarity and dissimilarity settings, some of the widely used heuristics may perform badly. The proofs are neither difficult nor particularly interesting, but the results stand in sharp contrast to those for structured inputs and help motivate our study of inputs beyond worst case.

Similarity Graphs. We show that for very simple input graphs (i.e., unweighted trees), the linkage algorithms (adapted to the similarity setting, see Algorithm 6) may perform badly.

Theorem B.1. *There exists an infinite family of inputs on which the single-linkage and complete-linkage algorithms output a solution of cost* $\Omega(\frac{n}{\log n} \cdot OPT)$.

Proof. The family of inputs consists of the graphs that represent paths of length $n > 2$. More formally, Let G_n be a graph on n vertices such that $V = \{v_1, \ldots, v_n\}$ and that has the following edge weights. Let $w(v_{i-1}, v_i) = w(v_i, v_{i+1}) = 1$, for all $1 < i < n$ and for any $i, j, j \notin \{i-1, i, i+1\}$, define $w(v_i, v_j) = 0$. Dasgupta [23] showed that $\text{OPT}(G_n) = O(n \log n)$.

Complete-Linkage. We show that the complete-linkage algorithm could perform a sequence of merges that would induce a tree of cost $\Omega(n^2)$. The complete-linkage algorithm is defined as merging the two clusters C_i, C_j that maximize the $W(C_i, C_j) = \min_{u \in C_i, v \in C_j} w(u, v)$.

Observe that in the instance created $w(u, v) \in \{0, 1\}$. So, w.l.o.g. the first merges done by the algorithm will consist of vertices v_i, v_{i+1}, for i even since the similarity between these pairs of clusters is 1 and so maximum. Let $V_{i/2}$ denote the cluster containing v_i, v_{i+1}. Now, observe that for the resulting clusters, the similarity between them is 0, since $w(u, v) = 0$ for any u, v that are not adjacent on the line and so the order of merges could be arbitrary.

Thus, the sequence of merges done by the algorithm could result in a hierarchical clustering tree where the root node splits the nodes such that on the one side there are all the nodes in clusters V_j such that j is even and on the other side all the nodes in clusters V_j such that j is odd. Since the total number of edges between all the odd and even such clusters is $\Omega(n)$, the cost induced by the root node of the tree output is $\Omega(n^2)$.

Single-Linkage. Recall that the algorithm merges the two candidate clusters C_i, C_j that maximize $w(C_i, C_j) = \max_{u \in C_i, v \in C_j} w(u, v)$.

At the beginning, each cluster contains a single vertex and so, the algorithm could merge any two clusters $\{v_i\}, \{v_j\}$ for $j \in \{i-1, i+1\}$, since the edge weight is 1. Suppose w.l.o.g. that the algorithm merges v_1, v_2, let V_1 be this cluster. Then, observe that $w(V_1, v_3) = 1$ as well and so the next merge can be V_1, v_3. An immediate induction shows that the hierarchical clustering tree output by single-linkage could be a caterpillar: namely, the first split at the root being a cut $V - v_n, v_n$, the next split on $G[V - v_n]$ being $V - v_n - v_{n-1}, v_{n-1}$ and so on. As observed by Dasgupta [23],

the cost of this tree is $\Omega(n^2)$ and so the ratio between OPT and single-linkage could be as bad as $\Omega(n/\log n)$. □

Theorem B.2. *There exists an infinite family of inputs on which the average-linkage algorithm outputs a solution of cost $\Omega(n^{1/3}OPT)$.*

Proof. For any $n = 2^i$ for some integer i, we define a tree $T_n = (V, E)$ as follows. Let $k = n^{1/3}$. Let $P = (u_1, \ldots, u_k)$ be a path of length k (i.e., for each $1 \leq i < k$, we have an edge between u_i and u_{i+1}). For each u_i, we define a collection $P_i = \{P_1^i = (V_1^i, E_1^i), \ldots, P_k^i = (V_k^i, E_k^i)\}$ of k paths of length k and for each P_j^i we connect one of its extremities to u_i. Define $V_i = \{u_i\} \bigcup_j V_j^i$ and $V = \bigcup_i V_i$. □

Claim 4. $OPT(T_n) \leq 3n^{4/3}$

Proof. Consider the following non-binary solution tree T^*: Let the root have children $N_1, \ldots, N_k$ such that $V(N_i) = V_i$ and for each child N_i let it have children N_i^j such that $V(N_i^j) = V_i^j$. Finally, for each N_i^j let the subtree rooted at N_i^j be any tree.

We now analyze the cost of T^*. Observe that for each edge e in the path P, we have $|V(\mathrm{LCA}_{T^*}(e))| = n$. Moreover, for each edge e connecting a path P_j^i to u_i, we have $|V(\mathrm{LCA}_{T^*}(e))| = k^2 = n^{2/3}$. Finally, for each edge e whose both endpoints are in a path P_j^i, we have that $|V(\mathrm{LCA}_{T^*}(e))| \leq k = n^{1/3}$.

We now sum up over all edges to obtain the overall cost of T_n. There are $k = n^{1/3}$ edges in P; They incur a cost of $nk = n^{4/3}$. There are k^2 edges joining a vertex u_i to a path P_j^i; They incur a cost of $k^2 \cdot n^{2/3} = n^{4/3}$. Finally, there are k^3 edges whose both endpoints are in a path P_j^i; they incur a cost of at most $k^3 \cdot n^{1/3} = n^{4/3}$. Thus, the total cost of this tree is at most $3n^{4/3} \geq \mathrm{OPT}(T_n)$. □

We now argue that there exists a sequence of merges performed by the average-linkage algorithm that yields a solution of cost at least $n^{5/3}$.

Claim 5. *There exists a sequence of merges performed by average-linkage and an integer t such that the candidate trees at time t have leaves sets $\{\{u_1, \ldots, u_k\}\} \bigcup_{i,j} \{V_j^i\}$.*

Equipped with this claim, we can finish the proof of the proposition. Since there is no edge between V_j^i and $V_{j'}^{i'}$ for $i' \neq i$ or $j' \neq j$ the similarity between those trees in the algorithm will always be 0. However, the similarity between the tree $\widetilde{T}$ that has leaves set $\{u_1, \ldots, u_k\}$ and any other tree is positive (since there is one edge joining those two sets of vertices in T_n). Thus, the algorithm will merge $\widetilde{T}$ with some tree whose vertex set is exactly V_j^i for some i, j. For the same reasons, the resulting cluster will be merged with a cluster whose vertex set is exactly $V_{j'}^{i'}$, and so on. Hence, after $n/2k = k^2/2$ such merges, the tree $\widetilde{T}$ has a leaves set of size $k \cdot k^2/2 = n/2$. However, the number of edges from this cluster to the other candidate clusters is $k^2/2$ (since each other remaining clusters correspond to a vertex set V_j^i for some i, j) For each such edge e we have $|V(\mathrm{LCA}_T(e))| \geq n/2$. Since there are $k^2/2$ of them, the resulting tree has cost $\Omega(n^{5/3})$. Combining with Claim 4 yields the theorem.

We thus turn to the proof of Claim 5.

Proof of Claim 5. Given a graph G and a set of candidate trees C. Define G/C to be the graph resulting from the contraction of all the edges for which both endpoints belong to the same cluster. We show a slightly stronger claim. We show that for any graph G and candidate trees $\mathcal{V}$ such that

(1) All the candidate clusters in $\mathcal{V}$ have the same size; and
(2) There exists a bijection ϕ between vertices $v \in T_n$ and vertices in G/C.

There exists a sequence of merges and an integer t such that the candidate trees at time t have leaves sets $\{\{\phi(u_1), \ldots, \phi(u_k)\}\} \bigcup_{i,j}\{\phi(V_j^i)\}$ where $\phi(V_j^i) = \{\phi(v) \mid v \in V_j^i\}$.

This slightly stronger statement yields the claim by observing that T_n and the candidate trees at the start of the algorithm satisfy the conditions of the statement.

We proceed by induction on the number of vertices of the graph. Let $V_j^i = \{v_j^i(1), \ldots, v_j^i(k)\}$ such that $(v_j^i(\ell), v_j^i(\ell+1)) \in E_j^i$ for any $1 \leq \ell < k$, and $(v_j^i(k), u_i) \in E$.

We argue that the algorithm could perform a sequence of merges that results in the following set C of candidate trees. C contains candidate trees $U^i = \phi(u_{2i-1}) \cup \phi(u_{2i})$ for $1 \leq i < k/2$, and for each i, j, candidate trees $v_{i,j,\ell} = \phi(v_j^i(2\ell-1) \cup \phi(v_j^i(2\ell))$, for $1 \leq \ell < k/2$. Let s_0 be the number of vertices in each candidate tree.

At first, all the trees contain a single vertex and so, for each pair of adjacent vertices of the graph the similarity between their corresponding trees in the algorithm is $1/s_0$. For any non-adjacent pair of vertices, the corresponding trees have similarity 0. Thus, w.l.o.g. assume the algorithm first merges u_1, u_2. Then, the similarity between the newly created tree U^1 and any other candidate tree C is 0 if there is no edge between u_1 and u_2 and C or $1/(2s_0)$ if there is one (since U^1 contains now two vertices). For the other candidate trees the similarity is unchanged. Thus, the algorithm could merge vertices u_3, u_4. Now, observe that the similarity between U^2 and U^1 is at most $1/(4s_0)$. Thus, it is possible to repeat the argument assuming that the algorithm merges the candidate trees corresponding to u_5, u_6. Repeating this argument $k/2$ times yields that after $k/2$ merges, the algorithm has generated the candidates trees $U_1, \ldots, U_{k/2-1}$. The other candidate trees still contain a single vertex. Thus, the algorithm is now forced to merge candidate trees that contain single vertices that are adjacent (since their similarity is $1/s_0$ and any other similarity is $< 1/s_0$). Assume, w.l.o.g. assume that the algorithm merges $v_1^1(1), v_1^1(2)$. Again, applying a similar reasoning to each $v_1^1(2\ell-1), v_1^1(2\ell)$ yields the set of candidate clusters $v_{1,1,1}, \ldots, v_{1,1,k/2-1}$. Applying this argument to all sets V_j^i yields that the algorithm could perform a sequence of merges that results in the set C of candidate clusters described above.

Now, all the clusters have size $2s_0$ and there exists a bijection between vertices of G/C and $T_{n/2}$. Therefore, combining with the induction hypothesis yields the claim. □

Dissimilarity Graphs. We now show that single-linkage, and bisection 2-Center might return a solution that is arbitrarily bad compared to OPT in some cases. Hence, since average-linkage achieves a 2/3-approximation in the worst case it seems that it is more robust than the other algorithms used in practice.

Theorem B.3. *For each of the single-linkage, and bisection 2-Center algorithms, there exists a family of inputs for which the algorithm outputs a solution of value $O(OPT/n)$.*

Proof. We define the family of inputs for single-linkage and bisection 2-Center as follows. For any $n > 2$, the graph G_n consists of n vertices $V = \{v_1, \ldots, v_{n-1}, u\}$ and the edge weights are the following: For any $i, j \in \{1, \ldots, n-1\}$, $w(v_i, v_j) = 1$, for any $1 < i \leq n-1$, $w(v_i, u) = 1$, and $w(v_1, u) = W$ for some fixed $W \geq n^3$. Consider the tree T^* whose root induces a cut $(V \setminus \{u\}, \{u\})$. Then, the value of this tree (and so OPT) is at least nW, since $|V(\mathrm{LCA}_{T^*}(v_1, u))| = n$.

Single-Linkage. At the beginning, all the clusters are at distance 1 from each other except v_1 and u that are at distance W. Thus, suppose that the first merge generates a candidate tree C_1 whose leaves set is $\{v_1, v_2\}$. Now, since $w(v_2, u) = 1$, we have that all the clusters are at distance 1 from each other. Therefore, the next merge could possibly generate the cluster C_2 with leaves sets $\{u, v_1, v_2\}$. Assume w.l.o.g. that this is the case and let T be the tree output by the algorithm. We obtain $|V(\mathrm{LCA}_T(u, v_1))| = 3$ and so, since for any v_i, v_j, $|V(v_i, v_j)| \leq n$, we have $\mathrm{val}(T) \leq n^2 + 3W \leq 4W$, because $W > n^2$. Hence, $\mathrm{val}(T) = O(\mathrm{val}(T^*)/n)$.

Bisection 2-Center. It is easy to see that for any location of the two centers, the cost of the clustering is 1. Thus, suppose that the algorithm locates centers at v_2 and v_3 and that the induced partitioning is $\{v_1, v_2, u\}, V \setminus \{v_1, v_2, u\}$. It follows that $|V(\mathrm{LCA}_T(u, v_1))| \leq 3$ and so, $\mathrm{val}(T) \leq n^2 + 3W \leq 4W$, since $W > n^2$. Again, $\mathrm{val}(T) = O(\mathrm{val}(T^*)/n)$. □

ACKNOWLEDGMENTS

The authors are grateful to Sanjoy Dasgupta for sharing thoughtful comments at various stages of this project and thank P. J. Lamberson for making the depiction of the graph used in Figure 5 available to us.

REFERENCES

[1] Sanjeev Arora and Ravi Kannan. 2001. Learning mixtures of arbitrary Gaussians. In *Proceedings on the 33rd Annual ACM Symposium on Theory of Computing*. 247–257. DOI: https://doi.org/10.1145/380752.380808

[2] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. 2009. Expander flows, geometric embeddings and graph partitioning. *J. ACM* 56, 2 (2009). DOI: https://doi.org/10.1145/1502793.1502794

[3] Pranjal Awasthi, Avrim Blum, and Or Sheffet. 2010. Stability yields a PTAS for k-median and k-means clustering. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*. 309–318. DOI: https://doi.org/10.1109/FOCS.2010.36

[4] Pranjal Awasthi, Avrim Blum, and Or Sheffet. 2012. Center-based clustering under perturbation stability. *Inf. Process. Lett.* 112, 1–2 (2012), 49–54. DOI: https://doi.org/10.1016/j.ipl.2011.10.006

[5] Pranjal Awasthi and Or Sheffet. 2012. Improved spectral-norm bounds for clustering. In *Proceedings of the 15th International Workshop on Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques (APPROX'12) and Proceedings of the 16th International Workshop (RANDOM'12)*. 37–49. DOI: https://doi.org/10.1007/978-3-642-32512-0_4

[6] Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. 2009. Approximate clustering without the approximation. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*. 1068–1077.

[7] Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. 2013. Clustering under approximation stability. *J. ACM* 60, 2 (2013), 8. DOI: https://doi.org/10.1145/2450142.2450144

[8] Maria-Florina Balcan and Yingyu Liang. 2016. Clustering under perturbation resilience. *SIAM J. Comput.* 45, 1 (2016), 102–155. DOI: https://doi.org/10.1137/140981575

[9] Maria-Florina Balcan, Heiko Röglin, and Shang-Hua Teng. 2009. Agnostic clustering. In *Proceedings of the 20th International Conference Algorithmic Learning Theory (ALT'09)*. 384–398. DOI: https://doi.org/10.1007/978-3-642-04414-4_31

[10] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. 2008. A discriminative framework for clustering via similarity functions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC'08)*. ACM, New York, NY, 671–680. DOI: https://doi.org/10.1145/1374376.1374474

[11] Yonatan Bilu, Amit Daniely, Nati Linial, and Michael E. Saks. 2013. On the practically interesting instances of MAX-CUT. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS'13)*. 526–537. DOI: https://doi.org/10.4230/LIPIcs.STACS.2013.526

[12] Yonatan Bilu and Nathan Linial. 2012. Are stable instances easy? *Combin. Probab. Comput.* 21, 5 (2012), 643–660. DOI: https://doi.org/10.1017/S0963548312000193

[13] Ravi B. Boppana. 1987. Eigenvalues and graph bisection: An average-case analysis. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS'87)*. IEEE Computer Society, Los Alamitos, CA, 280–285. DOI: https://doi.org/10.1109/SFCS.1987.22

[14] S. Charles Brubaker and Santosh Vempala. 2008. Isotropic PCA and affine-invariant clustering. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS'08)*. 551–560. DOI: https://doi.org/10.1109/FOCS.2008.48

[15] Gunnar Carlsson and Facundo Mémoli. 2010. Characterization, stability and convergence of hierarchical clustering methods. *J. Mach. Learn. Res.* 11 (2010), 1425–1470.

[16] Rui M. Castro, Mark J. Coates, and Robert D. Nowak. 2004. Likelihood based hierarchical clustering. *IEEE Trans. Sign. Process.* 52, 8 (2004), 2308–2321.

[17] Moses Charikar and Vaggos Chatziafratis. 2017. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*. 841–854. DOI: https://doi.org/10.1137/1.9781611974782.53

[18] Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. 2019. Hierarchical clustering better than average-linkage. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. 2291–2304. DOI: https://doi.org/10.1137/1.9781611975482.139

[19] Vaggos Chatziafratis, Rad Niazadeh, and Moses Charikar. 2018. Hierarchical clustering with structural constraints. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*. 773–782.

[20] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. 2017. Hierarchical clustering beyond the worst-case. In *Advances in Neural Information Processing Systems*.

[21] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. 2018. Hierarchical clustering: Objective functions and algorithms. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 378–397. http://dl.acm.org/citation.cfm?id=3174304.3175293.

[22] Sanjoy Dasgupta. 1999. Learning mixtures of gaussians. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS'99)*. 634–644. DOI: https://doi.org/10.1109/SFFCS.1999.814639

[23] Sanjoy Dasgupta. 2016. A cost function for similarity-based hierarchical clustering. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC'16)*. ACM, New York, NY, 118–127. DOI: https://doi.org/10.1145/2897518.2897527

[24] Sanjoy Dasgupta and Philip M. Long. 2005. Performance guarantees for hierarchical clustering. *J. Comput. Syst. Sci.* 70, 4 (2005), 555–569.

[25] Sanjoy Dasgupta and Leonard J. Schulman. 2007. A probabilistic analysis of EM for mixtures of separated, spherical gaussians. *J. Mach. Learn. Res.* 8 (2007), 203–226.

[26] Justin Eldridge, Mikhail Belkin, and Yusu Wang. 2016. Graphons, mergeons, and so on!. In *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems*. 2307–2315.

[27] Joseph Felsenstein and Joseph Felenstein. 2004. *Inferring Phylogenies*. Vol. 2. Sinauer Associates Sunderland.

[28] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer Series in Statistics, Berlin.

[29] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *J. Am. Statist. Assoc.* 58, 301 (1963), 13–30.

[30] N. Jardine and R. Sibson. 1972. *Mathematical Taxonomy*. John Wiley & Sons.

[31] Jon Kleinberg. 2002. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems*, Vol. 15. 463–470.

[32] Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. 2012. Efficient active algorithms for hierarchical clustering. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*. Omnipress, 267–274. http://dl.acm.org/citation.cfm?id=3042573.3042611

[33] Amit Kumar and Ravindran Kannan. 2010. Clustering with spectral norm and the k-means algorithm. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*. 299–308. DOI: https://doi.org/10.1109/FOCS.2010.35

[34] Guolong Lin, Chandrashekhar Nagarajan, Rajmohan Rajaraman, and David P. Williamson. 2006. A general approach for incremental approximation and hierarchical clustering. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*. Society for Industrial and Applied Mathematics, 1147–1156.

[35] Vince Lyzinski, Minh Tang, Avanti Athreya, Youngser Park, and Carey E. Priebe. 2017. Community detection and classification in hierarchical stochastic blockmodels. *IEEE Trans. Netw. Sci. Eng.* 4, 1 (2017), 13–26.

[36] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. 2012. Approximation algorithms for semi-random partitioning problems. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC'12)*, Howard J. Karloff and Toniann Pitassi (Eds.). ACM, 367–384. DOI: https://doi.org/10.1145/2213977.2214013

[37] Frank McSherry. 2001. Spectral partitioning of random graphs. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*. 529–537. DOI: https://doi.org/10.1109/SFCS.2001.959929

[38] Benjamin Moseley and Joshua Wang. 2017. Approximation bounds for hierarchical clustering: Average linkage, bisecting K-means, and local search. In *Advances in Neural Information Processing Systems*, Vol. 30 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). 3097–3106.

[39] Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Chaitanya Swamy. 2012. The effectiveness of Lloyd-type methods for the k-means problem. *J. ACM* 59, 6 (2012), 28. DOI: https://doi.org/10.1145/2395116.2395117

[40] C. Greg Plaxton. 2003. Approximation algorithms for hierarchical location problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*. ACM, 40–49.

[41] Aurko Roy and Sebastian Pokutta. 2016. Hierarchical clustering via spreading metrics. In *Advances in Neural Information Processing Systems*. 2316–2324.

[42] Peter H. A. Sneath and Robert R. Sokal. 1962. Numerical taxonomy. *Nature* 193, 4818 (1962), 855–860.

[43] Michael Steinbach, George Karypis, and Vipin Kumar. 2000. A comparison of document clustering techniques. In *Proceedings of the ACM Knowledge Discovery and Data Mining Workshop on Text Mining (KDD'00)*.
[44] Reza Bosagh Zadeh and Shai Ben-David. 2009. A uniqueness theorem for clustering. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*. 639–646.

Received December 2017; revised March 2019; accepted March 2019