Hierarchical Clustering: $O(1)$-Approximation for Well-Clustered Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Hierarchical clustering studies a recursive partition of a dataset into clusters of successively smaller size, and is a fundamental problem in data analysis. In this work we study the cost function for hierarchical clustering introduced by Dasgupta [13], and present two polynomial-time approximation algorithms: Our first result is an $O(1)$-approximation algorithm for graphs of high conductance. Our simple construction bypasses complicated recursive routines of finding sparse cuts known in the literature (e.g., [7, 12]). Our second and main result is an $O(1)$-approximation algorithm for a wide family of graphs that exhibit a well-defined structure of clusters. This result generalises the previous state-of-the-art [11], which holds only for graphs generated from stochastic models. The significance of our work is demonstrated by the empirical analysis on both synthetic and real-world datasets, on which our presented algorithm outperforms the previous state-of-the-art and several classical linkage heuristics.

1 Introduction

Hierarchical clustering (HC) studies a recursive partition of a dataset into clusters of successively smaller size, via an effective binary tree representation. As a basic technique, hierarchical clustering has been employed as a standard package in data analysis, and has comprehensive applications in practice. While traditionally HC-trees were constructed through bottom-up (agglomerative) heuristics, which lacked a clearly-defined objective function, Dasgupta [13] has recently introduced a simple objective function to measure the quality of a particular hierarchical clustering and his work has inspired a number of research on this topic [3, 7, 8, 9, 11, 12, 21, 24]. Consequently, there has been a significant interest in studying efficient HC algorithms that not only work in practice, but also have proven theoretical guarantees with respect to Dasgupta's cost function.

Our contribution. In this work, we present two new approximation algorithms for constructing HC trees that can be rigorously analysed with respect to Dasgupta's cost function. For our first result, we present a simple HC construction of an input graph G *entirely* based on the degree sequence of $V(G)$, and we show that the approximation guarantee of our constructed tree is with respect to the conductance of G, which will be defined formally in Section 2. The striking fact of this result is that, for any n-vertex graph G with m edges and conductance $\Omega(1)$ (a.k.a. expander graphs), an $O(1)$-approximate HC tree of G can be very easily constructed in $O(m + n \log n)$ time, although obtaining such result for general graphs is impossible under the Small Set Expansion (SSE) Hypothesis [7]. Our theorem is in line with a sequence of results for problems that are naturally linked to the Unique Games and Small Set Expansion problems: it has been shown that such problems are much easier to solve once the input instance exhibits the high conductance property [4, 5, 17, 19]. However, to the best of our knowledge our result is the first of this type for hierarchical clustering.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

While our first result presents a very interesting theoretical fact, we further study whether we can extend this $O(1)$-approximation construction to a much wider family of graphs occurring in practice. Specifically, we look at *well-clustered graphs*, i.e., the graphs in which vertices within each cluster are better connected than vertices between different clusters and the total number of clusters is constant. This includes a wide range of graphs occurring in practice with a clear cluster-structure, and have been extensively studied over the past two decades (e.g., [14, 16, 23, 26]). As our second and main result, we present a polynomial-time $O(1)$-approximation algorithm that constructs an HC tree for a well-clustered graph. Given that the class of well-clustered graphs includes graphs with clusters of different sizes and asymmetrical internal structure, our result significantly improves the previous state-of-the-art [11], which only holds for graphs generated from stochastic models. At the technical level, the design of our algorithm is based on the graph decomposition algorithm presented in [14], which is designed to find a good partition of a well clustered graph. However, our analysis suggests that, in order to obtain an $O(1)$-approximation algorithm, directly applying their decomposition isn't sufficient for our purpose. To overcome this bottleneck, we refine the output decomposition via a *pruning* technique, and carefully merge the refined parts to construct our final HC tree. From our point of view, our presented stronger graph decomposition procedure might have applications in other settings as well.

To demonstrate the significance and potential practical impact of our work, we compare our algorithm against the previous state-of-the-art and well-known linkage heuristics on both synthetic and real-world datasets. Although our algorithm's performance is marginally better than the previous state-of-the-art for the graphs generated from the stochastic block models (SBM), the cost of our algorithm's output is up to 50% lower than the one from the previous state-of-the-art when the clusters of the input graph have different sizes and some cliques are embedded into a cluster. Finally, the outperformence of our algorithm over the previous state-of-the-art is further confirmed by our experiments on 5 classical real-world datasets for hierarchical clustering.

Related work. Our work fits in a line of research initiated by Dasgupta [13], who introduced a cost function to measure the quality of an HC tree. Dasgupta proved that a recursive application of the algorithm for the Sparest Cut problem achieves an $O(\log^{3/2} n)$-approximation, which has been subsequently improved to $O(\log n)$ and $O(\sqrt{\log n})$ by [24] and [7, 12]. It is known to be NP-hard to find an optimal HC tree [13] , and SSE-hard to achieve an $O(1)$-approximation for a general input instance [7] . Cohen-Addad et al. [11] studied a hierarchical extension of the SBM and showed that a certain SVD projection algorithm together with several linkage heuristics achieves an $O(1 + o(1))$-approximation with high probability. We emphasise that our notion of well-clustered graphs generalises the SBM variant studied by [11] and does not assume the rigid hierarchical structure of the clusters.

For another line of related work, Moseley and Wang [21] studied the dual objective function and proved that `Average Linkage` achieves a $(1/3)$-approximation for the new cost function. Notice that, although this new cost function has received significant attention very recently [3, 8, 9, 10, 25], achieving an $O(1)$-approximation is tractable under this alternative cost function: this suggests the fundamental difference on the hardness of the problem under different cost functions, and is our reason to entirely focus on the Dasgupta's cost function in this work.

2 Preliminaries

Graph notation. Throughout the paper, we always assume that $G = (V, E, w)$ is an undirected graph with n vertices, m edges and weight function $w : V \times V \to \mathbb{R}_{\geq 0}$. For any edge $e = \{u, v\} \in E$ we write w_e to indicate the *similarity* weight between u and v. For a vertex $u \in V$, we denote its degree by $d_u \triangleq \sum_{v \in V} w_{uv}$ and we assume that $w_{\max}/w_{\min} = O(\mathrm{poly}(n))$, where $w_{\min}(w_{\max})$ is the minimum (maximum) edge weight. We will use $d_{\min}, d_{\max}, d_{\mathrm{avg}}$ for the minimum, maximum and average degrees in G, where $d_{\mathrm{avg}} \triangleq \sum_{u \in V} d_u/n$. For a nonempty subset $S \subset V$, we define $G[S]$ to be the induced subgraph on S and we denote by $G\{S\}$ the subgraph $G[S]$, where self loops are added to vertices $v \in S$ such that their degrees in G and $G\{S\}$ are the same. For any two subsets $S, T \subset V$, we define the *cut value* $w(S, T) \triangleq \sum_{e \in E(S,T)} w_e$, where $E(S, T)$ is the set of edges between S and T. For any set $S \subset V$, the *volume* of S is $\mathrm{vol}(S) \triangleq \sum_{u \in S} d_u$ and we write $\mathrm{vol}(G)$

when referring to $\mathrm{vol}(V(G))$. We further define the conductance of S by

$$\Phi_G(S) \triangleq \frac{w(S, V \setminus S)}{\min\{\mathrm{vol}(S), \mathrm{vol}(V \setminus S)\}},$$

and $\Phi_G \triangleq \min_{\substack{S \subset V \\ S \neq \emptyset}} \Phi_G(S)$. We call G an expander graph if $\Phi_G = \Omega(1)$. For a graph G, let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the diagonal matrix defined by $\mathbf{D}_{uu} = d_u$ for all $u \in V$. We define $\mathbf{A} \in \mathbb{R}^{n \times n}$, where $\mathbf{A}_{uv} = w_{uv}$ for all $u, v \in V$. The *normalised Laplacian matrix* of G is defined as $\mathcal{L} \triangleq \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. We will denote the eigenvalues of $\mathcal{L}$ by $0 = \lambda_1 \leq \cdots \leq \lambda_n \leq 2$.

Hierarchical clustering. A *hierarchical clustering (HC) tree* of a given graph G is a binary tree $\mathcal{T}$ with n leaf nodes such that each leaf corresponds to exactly one vertex $v \in V(G)$. Let $\mathcal{T}$ be an HC tree of some graph $G = (V, E, w)$ and let $N \in \mathcal{T}$ be an arbitrary internal node[1] of $\mathcal{T}$. We denote $\mathcal{T}[N]$ to be the subtree of $\mathcal{T}$ rooted at N and $\mathsf{leaves}(\mathcal{T}[N])$ to be the set of leaf nodes of $\mathcal{T}[N]$. Notice that each internal node $N \in \mathcal{T}$ naturally induces a unique cluster $C \subseteq V$ formed by the vertices corresponding to $\mathsf{leaves}(\mathcal{T}[N])$. We will abuse the notation and write $N \in \mathcal{T}$ for both the internal node of $\mathcal{T}$ and the corresponding subset of vertices in V.

To measure the quality of an HC tree $\mathcal{T}$ with similarity weights, Dasgupta [13] introduced the following cost function:

$$\mathrm{cost}_G(\mathcal{T}) \triangleq \sum_{e=\{u,v\} \in E} w_e \cdot |\mathsf{leaves}\,(\mathcal{T}[u \vee v])|\,,$$

where $u \vee v$ is the lowest common ancestor of u and v in $\mathcal{T}$. Trees that achieve a better hierarchical clustering have a lower cost. The intuition behind this is as follows: for pairs of vertices $u, v \in V$ that correspond to large weights w_{uv} (i.e., they are highly similar) a "good" HC tree would place them together, thus reflected in a small size $|\mathsf{leaves}(\mathcal{T}[u \vee v])|$. We denote by OPT_G the minimum cost in G, i.e., $\mathsf{OPT}_G = \min_{\mathcal{T}} \mathrm{cost}_G(\mathcal{T})$, and use $\mathcal{T}^*$ to refer to an optimal tree achieving the minimum. We say that an HC tree $\mathcal{T}$ is an α-approximate tree if $\mathrm{cost}_G(\mathcal{T}) \leq \alpha \cdot \mathsf{OPT}_G$. All omitted proofs in the paper are deferred to the Appendix.

3 Hierarchical clustering for graphs of high conductance

In this section we study hierarchical clustering for graphs with high conductance and prove that, for any input graph G with $\Phi_G = \Omega(1)$, an $O(1)$-approximate HC tree of G can be simply constructed based on the degree sequence of G. As a starting point, we show that $\mathrm{cost}_G(\mathcal{T})$ for any $\mathcal{T}$ can be upper bounded with respect to Φ_G and the degree distribution of $V(G)$.

Lemma 3.1. *It holds for any HC tree $\mathcal{T}$ of graph G that* $\mathrm{cost}_G(\mathcal{T}) \leq \frac{9}{4\Phi_G} \cdot \min\left\{\frac{d_{\mathrm{avg}}}{d_{\min}}, \frac{d_{\max}}{d_{\mathrm{avg}}}\right\} \cdot \mathsf{OPT}_G$.

While Lemma 3.1 holds for any graph G, it implies some interesting facts for expander graphs: first of all, when $G = (V, E, w)$ satisfies $d_{\max}/d_{\min} = O(1)$ and $\Phi_G = \Omega(1)$, Lemma 3.1 shows that any $\mathcal{T}$ is an $O(1)$-approximate tree. In addition, although Φ_G plays a crucial role in analysing $\mathrm{cost}_G(\mathcal{T})$ as for many other graph problems, Lemma 3.1 indicates that the degree distribution of G might also have significant impact. One could naturally ask the extend to which the degree distribution of $V(G)$ would influence the construction of OPT_G.

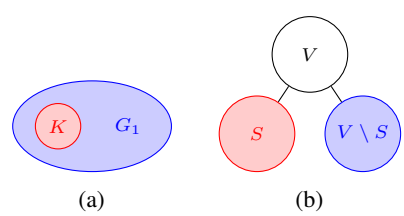


Figure 1: (a) Our considered graph G; (b) the tree that separates the vertices of high degrees from the others achieves $O(1)$-approximation.

To study this question, we look at the following graph G where all edges have unit weight: (i) let $G_1 = (V, E_1)$ be a constant-degree expander graph of n vertices with $\Phi_{G_1} = \Omega(1)$, e.g., the

[1] We will always use *(internal) nodes* for the nodes of $\mathcal{T}$ and *vertices* for the elements of the vertex set V.

ones presented in [15]; (ii) we choose $n^{2/3}$ vertices from V to form S, and let $K = (S, S \times S)$ be a complete graph defined on S; (iii) let $G \triangleq (V, E_1 \cup (S \times S))$ be the union of G_1 and K, see Figure 1(a) for illustration. By construction, we know that $\Phi_G = \Omega(1)$, and the degrees of G satisfy $d_{\max} = \Theta(n^{2/3})$, $d_{\min} = \Theta(1)$, and $d_{\text{avg}} = \Theta(n^{1/3})$. Therefore, the ratio between $\text{cost}_G(\mathcal{T})$ for any HC tree $\mathcal{T}$ and OPT_G could be as high as $\Theta(n^{1/3})$. On the other side, it's not difficult to show that the tree $\mathcal{T}^*$ illustrated in Figure 1(b), which first separates the set S of high-degree vertices from $V \setminus S$ at the top of the tree, actually $O(1)$-approximates OPT_G[2].

This example suggests that grouping vertices of similar degrees first would potentially help reduce $\text{cost}_G(\mathcal{T})$ for our constructed $\mathcal{T}$. This motivates us to study the following simple Algorithm 1 to construct an HC tree. We highlight that the output of Algorithm 1 is uniquely determined by the ordering of the vertices of G according to their degrees, which can be computed in $O(n \cdot \log n)$ time.

Algorithm 1: `HCwithDegrees`$(G\{V\})$

Input: $G = (V, E, w)$ with the ordered vertices such that $d_{v_1} \geq \ldots \geq d_{v_n}$
; **Output:** An HC tree $\mathcal{T}_{\text{deg}}(G)$;
if $|V| = 1$ **then**
  return the single vertex in V as the tree;
else
  $i_{\max} := \lfloor \log_2(|V| - 1) \rfloor$; $r := 2^{i_{\max}}$; $A := \{v_1, \ldots, v_r\}$; $B := V \setminus A$;
  Let $\mathcal{T}_1 =$ `HCwithDegrees`$(G\{A\})$; $\mathcal{T}_2 =$ `HCwithDegrees`$(G\{B\})$;
  return $\mathcal{T}_{\text{deg}}$ as the union of $\mathcal{T}_1$ and $\mathcal{T}_2$.

Theorem 1. *Given any graph $G = (V, E, w)$ with conductance Φ_G as input, Algorithm 1 runs in $O(m + n \log n)$ time, and returns an HC tree $\mathcal{T}_{\text{deg}}$ of G that satisfies $\text{cost}_G(\mathcal{T}_{\text{deg}}) = O\left(1/\Phi_G^4\right) \cdot \mathsf{OPT}_G$.*

Theorem 1 guarantees that, when the input G satisfies $\Phi_G = \Omega(1)$, the output of Algorithm 1, denoted by $\mathcal{T}_{\text{deg}}$, achieves an $O(1)$-approximation. Moreover, as the high-conductance property can be determined in nearly-linear time by computing $\lambda_2(\mathcal{L}_G)$ and applying the Cheeger inequality, Algorithm 1 presents a very simple construction of an $O(1)$-approximate HC tree once the input G is known to have high conductance.

Proof sketch of Theorem 1. The key notion employed in our proof is that of the *dense branch* defined as follows: for any HC tree $\mathcal{T}$ of G, the dense branch is the maximal path in $\mathcal{T}$, $(A_0, A_1, \ldots, A_k)$ for some $k \in \mathbb{Z}_{\geq 0}$, such that the following hold: (1) A_0 is the root of $\mathcal{T}$; (2) for any $1 \leq i \leq k$, A_i is the child of A_{i-1} with the *higher* volume; (3) the final node A_k has the property that $\text{vol}(A_k) > \text{vol}(G)/2$ and both children of A_k have volume at most $\text{vol}(G)/2$.

At a very high level, the proof of Theorem 1 starts with any optimal HC tree $\mathcal{T}_0$ of G and constructs trees $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ and $\mathcal{T}_4$ such that (i) $\text{cost}_G(\mathcal{T}_i)$ can be upper bounded with respect to $\text{cost}_G(\mathcal{T}_{i-1})$ for every $1 \leq i \leq 4$, (ii) the final constructed $\mathcal{T}_4$ is exactly the tree $\mathcal{T}_{\text{deg}}(G)$, the output of Algorithm 1. Combining these two facts allows us to upper bound $\text{cost}_G(\mathcal{T}_{\text{deg}})$ with respect to OPT_G.

Step 1: Regularisation. Let $(A_0, \ldots, A_{k_0})$ be the dense branch of any optimal $\mathcal{T}_0$, and let B_i be the sibling of A_i, for all $1 \leq i \leq k_0$. Let $i_{\min} = \lfloor \log_2(|A_{k_0}|) \rfloor$ and $i_{\max} = \lfloor \log_2(|V| - 1) \rfloor$. For simplicity, we assume that the dense branch has size at least 2 and that $|A_1| \geq 2^{i_{\max}}$, which ensures that $i_{\min} \leq i_{\max}$. In general, those assumptions are likely to happen for any optimal $\mathcal{T}_0$, but for completeness we will deal with them in the Appendix. The goal of this step is to adjust the dense branch of $\mathcal{T}_0$, such that the resulting tree $\mathcal{T}_1$ satisfies the following condition: for all $i \in [i_{\min}, i_{\max}]$, there is a node of size exactly 2^i along the dense branch of $\mathcal{T}_1$. We achieve this by applying a sequence of operations, each of which creating a new node of exact size 2^i for some suitable i. Concretely, let $i \in [i_{\min}, i_{\max}]$ be the largest integer such that there is no node of size 2^i on the dense branch of $\mathcal{T}_0$. As $|A_{k_0}| \geq 2^{i_{\min}}$, there is some node A_j on the dense branch such that $|A_j| > 2^i$ and $|A_{j+1}| < 2^i$. We adjust the branch at A_j as follows: (i) we consider a partition of $B_{j+1} = B_{j+1}^1 \cup B_{j+1}^2$ such that $|A_{j+1}| + |B_{j+1}^2| = 2^i$; (ii) we replace the node A_j by some newly created node A_j^1 that has children

[2] To see this, notice that $\text{cost}(\mathcal{T}^*) \leq \text{cost}(\mathcal{T}^*[S]) + n \cdot \text{vol}(G_1) = \Theta(n^2)$, as the complete subgraph $G[S]$ induces a cost of $\Theta((n^{2/3})^3)$ [13]. The subgraph $G[S]$ also implies that $\mathsf{OPT}_G = \Omega(n^2)$.

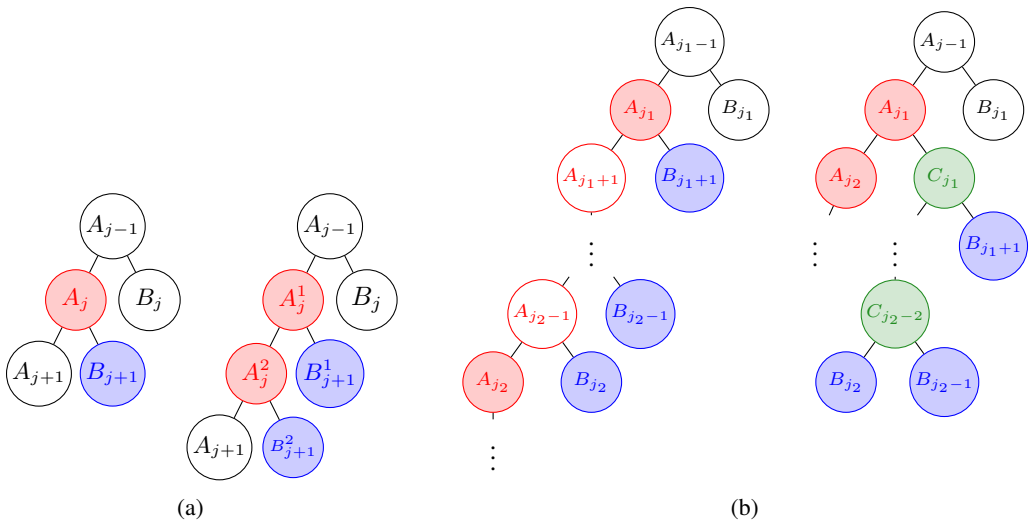


Figure 2: (a) With a proper partition of B_{j+1}, we have a new node A_j^2 of size 2^i; (b) The nodes between A_{j_1} and A_{j_2} (left) have size in $(2^{i-1}, 2^i)$; These nodes are compressed (right) and only the nodes of size some power of 2 remain in the dense branch.

B_{j+1}^1 and a new node A_j^2. (iii) the two children of A_j^2 will be A_{j+1} and B_{j+1}^2. This adjustment is illustrated in Figure 2(a), and we repeat this process until no such i exists any more. We call the resulting tree $\mathcal{T}_1$, and the following lemma gives an upper bound of $\mathrm{cost}_G(\mathcal{T}_1)$.

Lemma 3.2. *Our constructed tree $\mathcal{T}_1$ satisfies* $\mathrm{cost}_G(\mathcal{T}_1) \leq (1 + 4/\Phi_G) \cdot \mathrm{cost}_G(\mathcal{T}_0)$.

Step 2: Compression. With potential relabelling of the nodes, let $(A_0, \ldots, A_{k_1})$ be the dense branch of $\mathcal{T}_1$ with $|A_{k_1}| \geq 2^{i_{\min}}$, and B_i be the sibling of A_i. The objective of this step is to ensure that, by a sequence of adjustments, all the nodes along the dense branch are of size equal to some power of 2. Here, we describe how one such adjustment is performed, and refer the reader to Figure 2(b) for illustration. Let $i \in [i_{\min}, i_{\max}]$ be some index such that $|A_{j_1}| = 2^i$ and $|A_{j_2}| = 2^{i-1}$ for some $j_1 < j_2$. We compress the dense branch by removing all nodes between A_{j_1} and A_{j_2} as follows: The two children of A_{j_1} will be A_{j_2} and some new node C_{j_1}, which has children C_{j_1+1} and B_{j_1+1}; The two children of C_{j_1+1} will be some new node C_{j_1+2} and B_{j_1+2}, etc. The last node C_{j_2-1} has children B_{j_2-1} and B_{j_2}. In addition, we perform one more such adjustment to remove all nodes A_j of size $2^{i_{\max}} < |A_j| < n$ and ensure all nodes on the dense branch have size 2^i for some $i \in [i_{\min}, i_{\max}]$. We call the resulting tree $\mathcal{T}_2$, and the following lemma summaries the properties of $\mathcal{T}_2$.

Lemma 3.3. *Our constructed tree $\mathcal{T}_2$ satisfies that* $\mathrm{cost}_G(\mathcal{T}_2) \leq 2 \cdot \mathrm{cost}_G(\mathcal{T}_1)$.

Step 3: Matching. Let $(A_0, \ldots, A_{k_2})$ be the dense branch of $\mathcal{T}_2$, for some $k_2 \in \mathbb{Z}^+$, and B_i be the sibling of A_i. In this step we transform $\mathcal{T}_2$ into $\mathcal{T}_3$, such that $\mathcal{T}_3$ is isomorphic to $\mathcal{T}_{\deg}(G)$, which ensures that $\mathcal{T}_3$ and $\mathcal{T}_{\deg}(G)$ have the same structure. To achieve this, for every $1 \leq i \leq k_2$ we simply replace each $\mathcal{T}_2[B_i]$ with $\mathcal{T}_{\deg}(G\{B_i\})$. We further replace $\mathcal{T}_2[A_{k_2}]$ with $\mathcal{T}_{\deg}(G\{A_{k_2}\})$. We call the resulting tree $\mathcal{T}_3$, and bound its cost by the following lemma:

Lemma 3.4. *Our constructed tree $\mathcal{T}_3$ satisfies that* $\mathrm{cost}(\mathcal{T}_3) \leq (1 + 4/\Phi_G)\,\mathrm{cost}(\mathcal{T}_2)$.

Step 4: Sorting. We assume that $(A_0, \ldots, A_{k_3})$ is the dense branch of T_3, for some $k_3 \in \mathbb{Z}^+$, and we extend the dense branch to $(A_0, \ldots, A_{i_{\max}})$ with the same property that for every $i \in [k_3, i_{\max}]$ A_i is the child of A_{i-1} with the higher volume, and let $\mathcal{S} \triangleq \{B_1, \ldots, B_{i_{\max}}, A_{i_{\max}}\}$. Recall that in $\mathcal{T}_{\deg}(G)$ the first $r = 2^{i_{\max}}$ vertices of the highest degrees, i.e., $\{v_1, \ldots, v_r\}$ belong to A_1, of which the first $2^{i_{\max}-1}$ vertices belong to A_2, and so on; however, this might not be the case for $\mathcal{T}_3$. Hence, in the final step, we prove that $\mathcal{T}_3$ can be transformed into $\mathcal{T}_{\deg}(G)$ without a significant increase of the total cost. In this step we will swap vertices between the internal nodes $B_1, B_2, \ldots, B_{i_{\max}}, A_{i_{\max}}$ in such a way that $A_{i_{\max}}$ will consists of v_1 and v_2, $B_{i_{\max}}$ will consists of v_3 and v_4, $B_{i_{\max}-1}$ will consists of v_5 up to v_8, etc. We call vertex u *misplaced* if the position of u is $\mathcal{T}_3$ is different from the one in $\mathcal{T}_{\deg}(G)$. To transform $\mathcal{T}_3$ into $\mathcal{T}_{\deg}(G)$ we perform a sequence of operations, each of

which consisting in a chain of swaps focusing on the vertex of the *highest* degree that is currently misplaced. For the sake of argument, we assume that $v_1 \notin A_{i_{\max}}$ is misplaced, and we apply the following operation to move v_1 to $A_{i_{\max}}$: (i) Let $v_1 \in B_{i_0}$ for some $i_0 \geq 1$, and let y be the vertex of the lowest degree among the vertices in $\mathcal{S} \setminus \{B_1, \ldots, B_{i_0}\}$. Say $y \in B_{i_1}$ for some $i_1 > i_0$, and swap v_1 with y while keeping the structure of the tree unchanged; (ii) Repeat the swap operation above until v_1 reaches its correct place $A_{i_{\max}}$. Once the above process is complete and v_1 reaches $A_{i_{\max}}$, we apply a similar chain of swaps to v_2 to ensure v_2 also reaches $A_{i_{\max}}$. Then, we sequentially apply the process to v_3 and v_4 to ensure they reach $B_{i_{\max}}$, and continue until there are no more misplaced vertices.

We call the resulting tree $\mathcal{T}_3^{'}$, and notice that every node $X \in \mathcal{S}$ in $\mathcal{T}_3^{'}$ contains the correct set of vertices. However, the positions of these vertices in $\mathcal{T}_3^{'}[X]$ might be different from the ones in $\mathcal{T}_{\deg}(G\{X\})$. To overcome this issue, we repeat Step 3 again to the tree $\mathcal{T}_3^{'}$, and this will introduce another factor of $(1+4/\Phi_G)$ to the total cost of the constructed tree. Importantly, the final constructed tree after this step is exactly the tree $\mathcal{T}_{\deg}(G)$ and we bound its cost by the following lemma:

Lemma 3.5. *It holds for $\mathcal{T}_{\deg}$ that* $\mathrm{cost}(\mathcal{T}_{\deg}) \leq (1 + 16/\Phi_G)\,(1 + 4/\Phi_G) \cdot \mathrm{cost}(\mathcal{T}_3)$.

Finally, by combining Lemmas 3.2, 3.3, 3.4 and 3.5 we prove the approximation guarantee of $\mathcal{T}_{\deg}$ in Theorem 1. The runtime of Algorithm 1 follows by a simple application of the master theorem.

4 Hierarchical clustering for well-clustered graphs

As we have shown that an $O(1)$-approximate tree can be easily constructed for expander graphs, we will now focus on a wider class of well-clustered graphs. Informally, a well-clustered graph is a collection of densely-connected components (clusters) of high conductance, which are weakly interconnected. As these graphs form some of the most meaningful objects for clustering in practice, one would naturally ask whether our $O(1)$-approximation for expanders can be extended to well-clustered graphs. In this section, we will give an affirmative answer to this question.

To formalise the well-clustered property, we introduce the notion of $(\phi_{\mathrm{in}}, \phi_{\mathrm{out}})$-*decomposition* : for a graph $G = (V, E, w)$ and $k \in \mathbb{Z}^+$, we say that G has k well-defined clusters if $V(G)$ can be partitioned into disjoint subsets $\{S_i\}_{i=1}^k$, such that the following hold: (i) there's a sparse cut between S_i and $V \setminus S_i$ for any $1 \leq i \leq k$, which is formulated as $\Phi_G(S_i) \leq \phi_{\mathrm{out}}$, [3] and (ii) each induced subgraph $G[S_i]$ has high conductance $\Phi_{G[S_i]} \geq \phi_{\mathrm{in}}$. We underline that, through the celebrated higher-order Cheeger inequality [18], this condition of $(\phi_{\mathrm{in}}, \phi_{\mathrm{out}})$-decomposition can be approximately reduced to other formulations of a well-clustered graph found in the literature e.g., [23, 26, 27].

The starting point of our second result is the following polynomial-time algorithm presented by Gharan and Trevisan [14] for $(\phi_{\mathrm{in}}, \phi_{\mathrm{out}})$-decomposition of graph G. Specifically, given any well-clustered graph as input, their algorithm returns disjoint sets of vertices $\{P_i\}_{i=1}^{\ell}$ with bounded $\Phi(P_i)$ and $\Phi_{G[P_i]}$ for each P_i, and the algorithm's performance is as follows:

Lemma 4.1 ([14], Theorem 1.6)**.** *Let $G = (V, E, w)$ be a graph such that $\lambda_{k+1} > 0$, for some $k \geq 1$. Then, there is a polynomial-time algorithm that finds an ℓ-partitioning $\{P_i\}_{i=1}^{\ell}$ of V, for some $\ell \leq k$, such that the following hold for every $1 \leq i \leq \ell$:*

$(A1)$ $\Phi(P_i) = O\left(k^6\sqrt{\lambda_k}\right)$;

$(A2)$ $\Phi_{G[P_i]} = \Omega\left(\lambda_{k+1}^2/k^4\right)$.

Informally, this result states that, when the underlying input graph G presents a clear structure of clusters, one can find in polynomial-time a partition $\{P_i\}_{i=1}^{\ell}$ such that both the outer and inner conductance of every P_i can be bounded. One natural question raising from this partition $\{P_i\}_{i=1}^{\ell}$ is whether we can directly use $\{P_i\}_{i=1}^{\ell}$ to construct an HC tree. As an obvious approach, one could consider to (i) construct trees $\mathcal{T}_i = \mathcal{T}_{\deg}(G[P_i])$ for every $1 \leq i \leq \ell$, and (ii) merge the tres $\{\mathcal{T}_i\}$ in the best way to construct the final tree $\mathcal{T}_G$. Unfortunately, we will show in Appendix an example where this approach fails to achieve an $O(1)$-approximation[4]. To address this, we follow our intuition

[3] Without loss of generality, throughout the paper we assume that $\phi_{\mathrm{out}} < 1/3$.

[4] The example consists of two copies of the graph in Figure 1(a), connected by a sparse cut.

gained from Figure 1, and further decompose every P_i into smaller subsets. Similar with analysing *dense branches*, we introduce the *critical nodes* associated with each $\mathcal{T}_i$.

Definition 4.2 (Critical nodes)**.** *Let $\mathcal{T}_i = \mathcal{T}_{\deg}(G[P_i])$ be the tree obtained by applying Algorithm 1 to the induced graph $G[P_i]$. Suppose $(A_0, \ldots, A_{r_i})$ is the dense branch of $\mathcal{T}_i$, for some $r_i \in \mathbb{Z}_+$, B_j is the sibling of A_j and let A_{r_i+1}, B_{r_i+1} be the two children of A_{r_i}. We define $\mathcal{S}_i \triangleq \{B_1, \ldots, B_{r_i+1}, A_{r_i+1}\}$ to be the set of* critical nodes *of P_i. Each node $N \in \mathcal{S}_i$ is a* critical node.

We remark that each critical node $N \in \mathcal{S}_i$ $(1 \leq i \leq \ell)$ is a maximal internal node of $\mathcal{T}_i$ that is not on the dense branch. Moreover, each $\mathcal{S}_i$ forms a partition of P_i. Based on critical nodes, we present an improved decomposition algorithm, which is similar to Lemma 4.1, and prove that the output quality of our algorithm can be significantly strengthened for hierarchical clustering. Specifically, in addition to satisfying $(A1)$ and $(A2)$, we prove that the total weight between each critical node $N \in \mathcal{S}_i$ and all the other clusters P_j for all $i \neq j$ can be upper bounded. We highlight that this is one of the key properties that allows us to obtain our main result, and also suggests that the original decomposition algorithm in [14] might not suffice for our purpose.

Lemma 4.3 (Strong Decomposition Lemma)**.** *Let $G = (V, E, w)$ be a graph such that $\lambda_{k+1} > 0$. Then, there is a polynomial-time algorithm that finds an ℓ-partitioning of V into sets $\{P_i\}_{i=1}^{\ell}$ for some $\ell \leq k$, such that properties $(A1)$ and $(A2)$ hold for every $1 \leq i \leq \ell$. Additionally, for every cluster P_i and every critical node $N \in \mathcal{S}_i$, it holds that*

$$(A3)\ \ w(N, V \setminus P_i) \leq 6(k+1) \cdot \mathrm{vol}_{G[P_i]}(N).$$

To underline the importance of $(A3)$, recall that, in general, each subtree $\mathcal{T}_i$ cannot be directly used to construct an $O(1)$-approximate HC tree of G because of the potential high cost of the crossing edges $E(P_i, V \setminus P_i)$. However, if the internal cost of $\mathcal{T}_i$ is high enough to compensate for the cost introduced for the crossing edges $E(P_i, V \setminus P_i)$, then one can safely use this $\mathcal{T}_i$ as a building block. This is one of the most crucial insights that leads us to design our final algorithm `PruneMerge`.

Now we are ready to describe the algorithm `PruneMerge`, and we refer the reader to Algorithm 2 for the formal presentation. At a high-level, our algorithm consists of three phases: `Partition`, `Prune` and `Merge`. In the `Partition` phase (Lines 1–2), the algorithm invokes Lemma 4.3 to partition $V(G)$ into sets $\{P_i\}_{i=1}^{\ell}$, and applies Algorithm 1 to obtain the corresponding trees $\{\mathcal{T}_i\}_{i=1}^{\ell}$. The `Prune` phase (Lines 4–12) consists of a repeated pruning process: for every such tree $\mathcal{T}_i$, the algorithm checks in Line 7 if the maximal possible cost of the edges $E(P_i, V \setminus P_i)$ (i.e., the LHS of the inequality in the `if`-condition) can be bounded by the internal cost of the critical nodes $N \in S_i$, up to a factor of $O(k)$.

- If so, the algorithm uses $\mathcal{T}_i$ as a building block and adds it to a global set of trees $\mathbb{T}$.
- Otherwise, the algorithm prunes the subtree $\mathcal{T}_i[N]$, where $N \in \mathcal{S}_i$ is the critical node closest to the root in $\mathcal{T}_i$, and adds $\mathcal{T}_i[N]$ to $\mathbb{T}$ (cf. Line 11).

The process is repeated with the pruned $\mathcal{T}_i$ until either the condition in Line 7 is satisfied, or $\mathcal{T}_i$ is completely pruned. Finally, in the `Merge` phase (Lines 13–15) the algorithm combines the trees in $\mathbb{T}$ in a "caterpillar style" according to an increasing order of their sizes. The performance of this algorithm is summarised in Theorem 2.

Theorem 2. *Let $G = (V, E, w)$ be graph, and $k > 1$ such that $\lambda_{k+1} > 0$. Then, the algorithm* `PruneMerge` *runs in polynomial-time and constructs an HC tree $\mathcal{T}_{\mathrm{PM}}$ of G satisfying $\mathrm{cost}_G(\mathcal{T}_{\mathrm{PM}}) = O\left(k^2/\phi_{\mathrm{in}}^{10}\right) \cdot \mathsf{OPT}_G$. In particular, when $\lambda_{k+1} = \Omega(1)$ and $\lambda_{k+1}/\lambda_k = \Omega(\mathrm{poly}(k))$ for some constant k, the algorithm's constructed tree $\mathcal{T}_{\mathrm{PM}}$ satisfies that $\mathrm{cost}_G(\mathcal{T}_{\mathrm{PM}}) = O(1) \cdot \mathsf{OPT}_G$.*

We remark that, although Algorithm 2 requires a parameter k as input, we can apply the standard trick of running Algorithm 2 for different values of k and return the tree of lowest cost. By introducing a factor of $O(k)$ to the algorithm's runtime, this ensures that one of the resulting trees constructed by Algorithm 2 would always satisfy our promised approximation ratio.

5 Experiments

We experimentally evaluate the performance of our proposed algorithm, and compare it against the previous state-of-the-art and well-known linkage heuristics on both synthetic and real-world datasets. Specifically, our algorithm `PruneMerge` would be compared against the following 5 algorithms:

Algorithm 2: `PruneMerge`(G, k)

Input: A graph $G = (V, E, w)$, a parameter $k \in \mathbb{Z}_+$ such that $\lambda_{k+1} > 0$;
Output: An HC tree $\mathcal{T}_{\mathrm{PM}}$ of G;

Apply the partitioning algorithm (Lemma 4.3) on input (G, k) to obtain $\{P_i\}_{i=1}^{\ell}$ for some $\ell \leq k$;
Let $\mathcal{T}_i = \texttt{HCwithDegrees}(G[P_i])$;
Initialise $\mathbb{T} = \emptyset$;
for *All clusters* P_i **do**
  Let $\mathcal{S}_i$ be the set of critical nodes of $\mathcal{T}_i$;
  while $\mathcal{S}_i$ *is nonempty* **do**
    if $n \cdot \sum_{N \in \mathcal{S}_i} w(N, V \setminus P_i) \leq 6(k+1) \cdot \sum_{N \in \mathcal{S}_i} |\mathrm{parent}_{\mathcal{T}_i}(N)| \cdot \mathrm{vol}_{G[P_i]}(N)$ **then**
      Update $\mathbb{T} \leftarrow \mathbb{T} \cup \mathcal{T}_i$ and $\mathcal{S}_i = \emptyset$;
    else
      Let N, M be the two children of the root of $\mathcal{T}_i$ such that $N \in \mathcal{S}_i$;
      Update $\mathbb{T} \leftarrow \mathbb{T} \cup \mathcal{T}_i[N]$, $\mathcal{S}_i \leftarrow \mathcal{S}_i \setminus \{N\}$ and $\mathcal{T}_i \leftarrow \mathcal{T}_i[M]$;
Let $t = |\mathbb{T}|$ and $\mathbb{T} = \{\widetilde{\mathcal{T}}_1, \ldots, \widetilde{\mathcal{T}}_t\}$ be such that $|\widetilde{\mathcal{T}}_i| \leq |\widetilde{\mathcal{T}}_{i+1}|$, for all $1 \leq i < t$;
Initialise $\mathcal{T}_{\mathrm{PM}} = \widetilde{\mathcal{T}}_1$;
for $i = 2 \ldots t$ **do**
  Update $\mathcal{T}_{\mathrm{PM}}$ as the union of $\mathcal{T}_{\mathrm{PM}}$ and $\widetilde{\mathcal{T}}_i$
return $\mathcal{T}_{\mathrm{PM}}$

- `AvgLinkage`, `CompleteLinkage`, and `SingleLinkage`: the three well-known linkage algorithms studied in the literature. We refer the reader to [12] for complete description.
- `CAKMT_SL` and `CAKMT_AL`: two variations of the algorithm proposed in [11]. At a high level, these two algorithms project the input data points into a lower dimensional Euclidean subspace, and apply the single and average linkage algorithms to the set of projected points.

All algorithms were implemented in Python 3.8 and the experiments were performed using an Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz processor, with 16 GB RAM. All of the reported costs below are averaged over 5 independent runs. For the parameter ϕ_{in} used in Lemma 4.3 and our implementation, we set $\phi_{\mathrm{in}} = \max\left\{\frac{\lambda_{k+1}}{140 \cdot (k+1)^2}, 2 \cdot \lambda_k\right\}$ throughout the experiments reported here.

Synthetic datasets.

Clusters of the same size. Now we study the graphs generated from the Stochastic Block Model (SBM). We first set the number of clusters as $k = 3$, and the number of vertices in each cluster $\{C_i\}_{i=1}^{3}$ as $1,000$. We assume that any pair of vertices within each cluster is connected by an edge with probability p, and any pair of vertices from different clusters is connected by an edge with probability q. We fix the value $q = 0,002$, and consider different values of $p \in [0.04, 0.2]$. As reported in Figure 3, the performance of our algorithm is marginally better than the previous state-of-the-art [11]. This is well expected, as the algorithm presented in [11] is specifically designed for the SBM, in which all the clusters have the same inner density characterised by parameter p, and their algorithm achieves a $(1 + o(1))$-approximation for those instances.

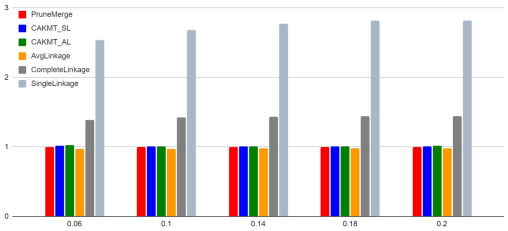
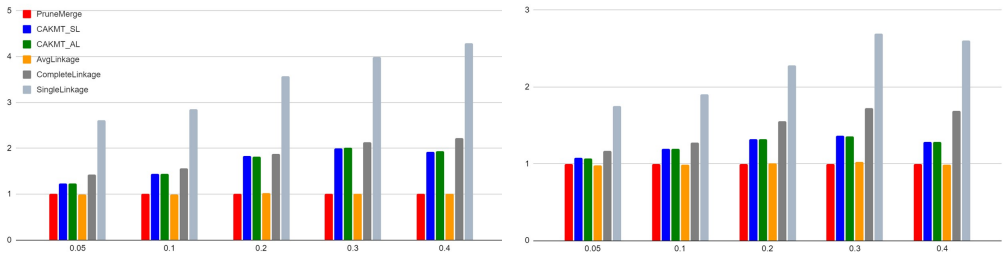


Figure 3: Results for clusters of the same size. The x-axis represents different values of p, while the y-axis represents the cost of the algorithms normalised by the cost of `PruneMerge`.

Clusters with non-uniform densities. Next we study graphs in which edges are present *non-uniformly* within each cluster (e.g., Figure 1(a) discussed earlier). Specifically, we set $k = 3, q = 0.002$, and $p = 0.06$; we choose a random set $S_i \subset C_i$ of size $|S_i| = c_p \cdot |C_i|$ from each cluster, and add edges to connect every pair of vertices in each S_i. That is, the vertices of each S_i form a clique. By setting

different values of $c_p \in [0.05, 0.4]$, the performance of our algorithm is about $20\% - 50\%$ better than the previous state-of-the-art [11] with respect to the cost value of the constructed tree, see Figure 4(a) for detailed results. To explain the outperformance of our algorithm, notice that, by adding a clique into some cluster, the cluster structure is usually preserved with respect to $(\phi_{\text{in}}, \phi_{\text{out}})$ or similar eigen-gap assumption on well-clustered graphs. However, the existence of such a clique within some cluster would make the vertices' degrees highly unbalanced; as such many clustering algorithms that involve the matrix perturbation theory in their analysis might not work well.



(a) (b)

Figure 4: Results for graphs with non-uniform densities, or different sizes. The x-axis represents the c_p-values, while the y-axis represents the cost of the algorithms normalised by the cost of `PruneMerge`. Figure (a) corresponds to inputs where all clusters have the same size, while in figure (b) the clusters have different sizes.

Clusters of different sizes. To further highlight the significance of our algorithm on synthetic graphs of non-symmetric structures among the clusters, we study the graphs in which the clusters have different sizes. We choose the same set of k and q values as before ($k = 3, q = 0.002$), but set the sizes of the clusters to be $|C_1| = 1900, |C_2| = 900$ and $|C_3| = 200$. Every pair of vertices $u, v \in C_i$, for $i \in \{1, 2\}$ is connected by an edge with probability $p_1 = 0.06$, while pairs of vertices $u, v \in C_3$ are connected with probability[5] $p_2 = 5 \cdot p_1 = 0.3$. We further plant a clique $S_i \subset C_i$ of size $|S_i| = c_p \cdot C_i$ for each cluster C_i, as the previous set of experiments. By choose different values of c_p from $[0.05, 0.4]$, our results are reported in Figure 4(b), demonstrating that our algorithm performs better than the ones in [11].

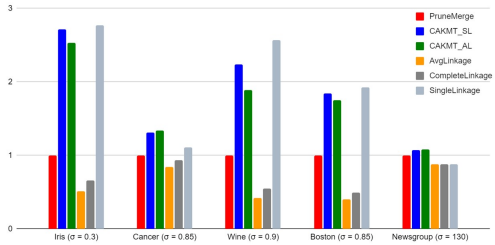


Figure 5: Results on real-world datasets. The x-axis represents the various datasets with the corresponding choice of σ for generating the similarity graphs. The y-axis corresponds to the cost of the algorithms normalised by the cost of `PruneMerge`.

Real-world datasets. To evaluate the performance of our algorithm on real-world datasets, we follow the sequence of recent work on hierarchical clustering [2, 11, 20, 24], all of which are based on the following 5 datasets from the Scikit-learn library [6] as well as the UCI ML repository [1]: Iris, Cancer, Wine, Boston and Newsgroup[6]. Similar with [24], for each dataset we construct the similarity graph based on the Gaussian kernel, in which the σ-value is chosen according to the standard heuristic [22]. As reported in Figure 5, compared with the previous state-of-the-art our algorithm produces much better results with respect to the cost values.

These experimental results demonstrate that our presented algorithms not only have excellent theoretical guarantees, but also produce output of lower cost than the previous state-of-the-art on various datasets. We believe that our work can lead to future practical applications.

[5] Such choice of p_2 is to compensate for the cluster C_3 of smaller size, and this ensures that the outer conductance of $\Phi_G(C_3)$ is low.

[6] Due to the very large size of this dataset, we consider only a subset consisting of "comp.graphics", "comp.os.ms-windows.misc", "comp.sys.ibm.pc.hardware", "comp.sys.mac.hardware", "rec.sport.baseball", and "rec.sport.hockey".

References

[1] UCI ML Repository. `https://archive.ics.uci.edu/ml/index.php`. Accessed: 2021-05-22.

[2] Amir Abboud, Vincent Cohen-Addad, and Hussein Houdrouge. Subquadratic high-dimensional hierarchical clustering. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *33rd Advances in Neural Information Processing Systems (NeurIPS'19)*, volume 32, pages 11576–11586. Curran Associates, Inc., 2019.

[3] Noga Alon, Yossi Azar, and Danny Vainstein. Hierarchical clustering: a 0.585 revenue approximation. In *33rd Annual Conference on Learning Theory (COLT'20)*, pages 153–162, 2020.

[4] Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the ACM*, 62(5), November 2015.

[5] Sanjeev Arora, Subhash Khot, Alexandra Kolla, David Steurer, Madhur Tulsiani, and Nisheeth K. Vishnoi. Unique games on expanding constraint graphs are easy: extended abstract. In *40th Annual ACM Symposium on Theory of Computing (STOC'08)*, pages 21–28, 2008.

[6] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv:1309.0238*, 2013.

[7] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 841–854, 2017.

[8] Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. Hierarchical clustering better than average-linkage. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 2291–2304, 2019.

[9] Moses Charikar, Vaggos Chatziafratis, Rad Niazadeh, and Grigory Yaroslavtsev. Hierarchical clustering for euclidean data. In *22nd International Conference on Artificial Intelligence and Statistics (AISTATS'19)*, pages 2721–2730, 2019.

[10] Vaggos Chatziafratis, Grigory Yaroslavtsev, Euiwoong Lee, Konstantin Makarychev, Sara Ahmadian, Alessandro Epasto, and Mohammad Mahdian. Bisect and conquer: Hierarchical clustering via max-uncut bisection. In *23rd International Conference on Artificial Intelligence and Statistics (AISTATS'20)*, pages 3121–3132, 2020.

[11] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. Hierarchical clustering beyond the worst-case. In *31st Advances in Neural Information Processing Systems (NeurIPS'17)*, pages 6201–6209, 2017.

[12] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 378–397, 2018.

[13] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 118–127, 2016.

[14] Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1256–1266, 2014.

[15] Hoory, Linial, and Wigderson. Expander graphs and their applications. *BAMS: Bulletin of the American Mathematical Society*, 43, 2006.

[16] Ravi Kannan, Santosh S. Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, 2004.

[17] Alexandra Kolla. Spectral algorithms for unique games. In *25th Conference on Computational Complexity (CCC'10)*, pages 122–130, 2010.

[18] James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM*, 61(6):37:1–37:30, 2014.

[19] Huan Li, He Sun, and Luca Zanetti. Hermitian laplacians and a cheeger inequality for the max-2-lin problem. In *27th Annual European Symposium on Algorithms (ESA'19)*, volume 144 of *LIPIcs*, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[20] Aditya Krishna Menon, Anand Rajagopalan, Baris Sumengen, Gui Citovsky, Qin Cao, and Sanjiv Kumar. Online hierarchical clustering approximations. *arXiv:1909.09667*, 2019.

[21] Benjamin Moseley and Joshua Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *31st Advances in Neural Information Processing Systems (NeurIPS'17)*, pages 3094–3103, 2017.

[22] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *15th Advances in Neural Information Processing Systems (NeurIPS'01)*, 14:849–856, 2001.

[23] Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! *SIAM Journal Comput.*, 46(2):710–743, 2017.

[24] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. *The Journal of Machine Learning Research*, 18(1):3077–3111, 2017.

[25] Danny Vainstein, Vaggos Chatziafratis, Gui Citovsky, Anand Rajagopalan, Mohammad Mahdian, and Yossi Azar. Hierarchical clustering via sketches and hierarchical correlation clustering. In *24th International Conference on Artificial Intelligence and Statistics (AISTATS'21)*, pages 559–567, 2021.

[26] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[27] Zeyuan Allen Zhu, Silvio Lattanzi, and Vahab S. Mirrokni. A local algorithm for finding well-connected clusters. In *30th International Conference on Machine Learning (ICML'13)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 396–404, 2013.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 3 and Section 4.
   (b) Did you describe the limitations of your work? [Yes] In the Related Work paragraph we compare our results against previous work placing our contribution in the appropriate context.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Sections 2, 3 and 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] Omitted proofs of the main claims will be present in the Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code will be included in the supplementary material. It is sufficient for reproducing the experimental results claimed in our submission.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A] Our presented algorithms are unsupervised.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] Our reported experimental results do not rely on large amount of computing resources. The configuration of the hardware that we used is listed in Section 5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] The datasets used in Section 5 include citations.

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]