Adaptive Hierarchical Clustering Using Ordinal Queries

Ehsan Emamjomeh-Zadeh* David Kempe†

November 2, 2017

Abstract

In many applications of clustering (for example, ontologies or clusterings of animal or plant species), hierarchical clusterings are more descriptive than a flat clustering. A hierarchical clustering over n elements is represented by a rooted binary tree with n leaves, each corresponding to one element. The subtrees rooted at interior nodes capture the clusters. In this paper, we study active learning of a hierarchical clustering using only ordinal queries. An ordinal query consists of a set of three elements, and the response to a query reveals the two elements (among the three elements in the query) which are "closer" to each other than to the third one. We say that elements x and x' are closer to each other than x'' if there exists a cluster containing x and x', but not x''.

When all the query responses are correct, there is a deterministic algorithm that learns the underlying hierarchical clustering using at most $n \log_2 n$ adaptive ordinal queries. We generalize this algorithm to be robust in a model in which each query response is correct independently with probability $p > \frac{1}{2}$, and adversarially incorrect with probability $1 - p$. We show that in the presence of noise, our algorithm outputs the correct hierarchical clustering with probability at least $1 - \delta$, using $O(n \log n + n \log(1/\delta))$ adaptive ordinal queries. For our results, adaptivity is crucial: we prove that even in the absence of noise, every non-adaptive algorithm requires $\Omega(n^3)$ ordinal queries in the worst case.

*Department of Computer Science, University of Southern California, emamjome@usc.edu
†Department of Computer Science, University of Southern California, dkempe@usc.edu

1 Introduction

One of the most useful and frequent computational tasks in interacting with a set of elements is to *cluster* them, inferring groups (*clusters*) of elements that are "similar to each other." Clustering is often an essential step in "learning" facts about the items, inferring generative models, deriving scientific or other insights, or letting users interact with large data sets meaningfully. The wide range of applications has led to an enormous amount of research spanning many disciplines, driven by different applications, different notions of "similarity," different constraints on allowable groups of elements, and different models of user interaction, to name just a few.

Two particularly common restrictions on the allowable groups are *flat clustering*, in which each item belongs to exactly (or at most) one cluster, and *hierarchical clustering*, in which the groups form a nested structure. Both restrictions have myriad natural applications. Among the prototypical applications of hierarchical clusterings are hierarchies of animal or plant species [19], ontologies of words or other entities, or social networks, viewed at different levels of granularity. Hierarchical clusterings are particularly well suited for human data exploration, in that they allow a person to zoom in/out to understand the elements and their relationships at different levels. In this paper, we are interested in inferring a hierarchical clustering on a set $\mathcal{X}$ of n elements.

A second key question is how the similarities between elements are defined and obtained. In much of the work on clustering, the similarities are derived from observable features of the elements (such as their position in an observable metric space), and finding a clustering is cast as an optimization problem for a suitably chosen objective function. Such an approach frequently identifies clusters quite similar — but not identical — to the ground truth. An alternative approach is to interact with human users, and obtain input to guide the search for the true clustering. These interactions are typically expensive; thus, minimizing the number of interactions with human users is often a primary goal in designing algorithms. When interacting with humans, it is also important to keep in mind that even when there is an objective meaning to numerical similarity values, humans are notoriously bad at estimating such numerical quantities. This has led to a search for suitable models of interaction, advocated recently in [8, 7, 6, 38], in which human users are only given queries they may reasonably be in a position to answer.

Inspired by the equivalence query model for learning a classifier [3], Balcan and Blum [8] and Awasthi et al. [7, 6] defined a model of interaction (specifically for flat clustering) in which an algorithm repeatedly proposes a clustering, and a user can request corrections, such as splitting or merging clusters. In this model, the user has a lot of leeway in determining what feedback the algorithm receives. The *ordinal query* model ([35, 23, 38, 28], for example) is more reminiscent of traditional "active learning" models. Here, a user is given queries comprising triplets of elements, and asked to identify the pair from each triplet that is closest. For example, a user may be asked which pair out of {lion, tiger, shark} is most similar. Topics such as social computing and a computational lens on political and other decision making processes have led to increasing interest in ordinal query responses. Not only have they been analyzed in the context of clustering or inferring distances [35, 1, 36, 23, 38], but also in social choice scenarios such as voting [5, 4] and political and group decision making [20, 21]. *Learning of a hierarchical clustering from ordinal queries is the topic of the present work.*

More formally, we assume that there is an unknown ground truth hierarchical clustering to be learned. It is represented by an (unknown) tree $\mathcal{T}^*$ whose n leaves are the elements. For each node v of the tree, the leaves in the subtree rooted at v form a cluster in the hierarchical clustering. For most of the paper (except Section 6), we assume that $\mathcal{T}^*$ is binary. This implies that for each triplet $\{x, x', x''\}$ of leaves/elements, there is a unique pair that shares a closer common ancestor

than any other pair:[1] this pair is the correct *response* to the query $\{x, x', x''\}$. The goal is to learn $\mathcal{T}^*$ (or an isomorphic tree) using few ordinal queries.

For some of this paper, we assume that the answers to all ordinal queries are correct. However, in particular in light of the motivation of interaction with humans, it is natural to assume that responses may be noisy. In the *independent noise mode* (which we use in Section 5), each response is correct independently with probability $p > \frac{1}{2}$, and adversarially incorrect otherwise.

It is not difficult to show (see Section 2.2) that even without incorrect responses, when the ordinal queries are non-adaptive (i.e., have to be chosen ahead of time), $\Theta(n^3)$ queries are necessary and sufficient. We therefore focus on adaptive queries.

For adaptive queries, when $\mathcal{T}^*$ has height $O(\log n)$, as pointed out by Tamuz et al. [36], there is a simple top-down algorithm to learn $\mathcal{T}^*$ using $O(n \log n)$ adaptive ordinal queries (with no incorrect responses). This algorithm can be considered as a special case of an algorithm of Pearl and Tarsi [34]. Their algorithm was designed for a different context, namely, discovering underlying causal trees over visible random variables. However, as a preliminary step, they studied a model essentially identical to the ordinal query model. Their algorithm learns the underlying tree (of any height) using $O(n \log n)$ queries. In Section 4, we will review their algorithm as a point of departure for our noise-tolerant algorithm in Section 5. A matching lower bound of $\Omega(n \log n)$ follows easily from a counting argument like the one for sorting (see Section 3).

Our Results and Techniques

The analogy with sorting is also useful in the design of algorithms for inferring a hierarchy. We begin (in Sections 3 and 4) by studying the model without incorrect responses.

Our first (and simplest) algorithm (in Section 3) is modeled after randomized QuickSort. It picks *two* random pivot elements x_A, x_B, and partitions the elements into those closest to x_A, those closest to x_B, and those which are further from both pivots than the two pivots are from each other. We prove that with constant probability, each of the three sets constitutes only a constant fraction of the original set. Then, we present an algorithm that, once the hierarchies for all three sets have been (recursively) computed, can merge them using a linear number of queries. As a result, the algorithm uses $O(n \log n)$ ordinal queries.

In preparation for the noise-tolerant algorithm in Section 5, in Section 4, we review[2] the algorithm of Pearl and Tarsi [34], which can be considered as a variant of InsertionSort. The main observation guiding the algorithm in Section 4 is that, given the true hierarchy over i elements, even when the tree is arbitrarily unbalanced, a new element can be inserted using at most $1 + \log_2 i$ ordinal queries, giving an overall query complexity of $n \log_2 n$. The algorithm to insert an element is a close variant of the generalization of Binary Search to trees or arbitrary graphs [33, 32, 16]. While the work on Binary Search in trees/graphs assumes a different query model called *vertex queries* (see Section 2.3), the algorithm is easily adapted.

Next, we turn our attention to the noisy model. It is easy to see that any algorithm that learns the true hierarchy in the absence of noise can be made robust to noise by repeating each ordinal query $\Theta(\log n)$ times and using a majority output. Then, Hoeffding's inequality and union bounds guarantee that the algorithm succeeds with high probability. However, this approach increases the number of queries by a factor of $\Theta(\log n)$. In Section 5, we show how to avoid the worse dependence on n for the InsertionSort-based algorithm, and only increase the number of queries by a constant factor depending solely on the correctness probability p.

[1] This does not hold in trees of higher degree.

[2] We would like to thank Daniel Hsu for pointing us to the article [34].

The robustness is achieved by replacing the Binary Search subroutine with a robust version. While robust Binary Search algorithms in a vertex query model in the presence of noise had been given in work of Feige et al. [18] and Emamjomeh-Zadeh et al. [16], neither algorithm itself has strong enough guarantees for our purposes: the algorithm of [18] requires a number of queries linear in the diameter of the tree, while the algorithm of [16] has a dependency on the desired error guarantee that would lead to a query complexity of $\Theta(\log^2 n)$ in our context. Our main contribution in Section 5 is a hybrid algorithm combining the first stage of the algorithm from [16] with the algorithm of [18] to obtain a number of queries that is logarithmic in n. Specifically, we show how, given the tree over a subset of elements, to correctly insert a new element into the tree with probability $1-\delta$, using $O(\log n + \log(1/\delta))$ queries. This leads to an algorithm that learns the correct hierarchy $\mathcal{T}^*$ with probability at least $1-\delta$ using $O(n(\log n + \log(1/\delta)))$ ordinal queries.

Related Work

In order to infer phylogenies, similarities and dissimilarities of different species can be interpreted as ordinal constraints. The algorithmic question of discovering the "best" hierarchical clustering over species (where "best" is with respect to a fixed objective function) has been surveyed in [19]. In particular, Aho et al. [2] presented an algorithm that, given a set of ordinal constraints ahead of time, finds a tree consistent with the constraints (or determines that none exists).

Vikram and Dasgupta [38] addressed the problem of incorporating hard ordinal constraints into algorithms for hierarchical clusterings. However, contrary to our model, they assume that the algorithm is provided information regarding the geometry of the data, and the hard constraints given by the user are additional information to improve the accuracy of the outcome.

A hierarchical clustering over n elements can be seen as a metric (or in fact, ultra-metric) over the elements.[3] In this sense, learning a hierarchical clustering is related to a recent line of work (mostly in the machine learning community) [1, 25, 24, 31, 36, 37, 29, 23] on constructing a geometric interpretation of data based on ordinal information. In particular, the "non-metric multidimensional scaling" problem [1, 25, 29, 23] is formulated as finding an embedding of the elements into a (low-dimensional) Euclidean space, such that the distances between the points satisfy a given set of ordinal constraints. Many of the algorithms on the non-metric multidimensional scaling problem are in the non-adaptive model (e.g., [1, 23]). Jamieson and Nowak [25] point out that an adaptive algorithm may be able to find an embedding of the elements into the low-dimensional space using only a few ordinal queries, in a way that the embedding satisfies all the $\Theta(n^3)$ possible queries. They proved a lower bound of $\Omega(dn \log n)$ where d is the dimension, and they conjectured that this lower bound is tight.

In the context of the *multinomial logit model*, a finite set of choices $\mathcal{C}$ satisfies the "Independence of Irrelevant Alternatives" (IIA) property [30] if for every two choices $c_1, c_2 \in \mathcal{C}$, the ratio of the probabilities of being chosen is a constant, regardless of which other choices (from $\mathcal{C}$) are available or unavailable. That is, for every two subsets of choices $\mathcal{C}', \mathcal{C}'' \subseteq \mathcal{C}$ with $\{c_1, c_2\} \subseteq \mathcal{C}', \mathcal{C}''$,

$$\frac{\Pr[c_1 \text{ is chosen} \mid \mathcal{C}']}{\Pr[c_2 \text{ is chosen} \mid \mathcal{C}']} = \frac{\Pr[c_1 \text{ is chosen} \mid \mathcal{C}'']}{\Pr[c_2 \text{ is chosen} \mid \mathcal{C}'']}.$$

In many applications where this strong assumption is not met, the *nested logistic regression* model [12] is used. In the nested logistic regression model, the choices are the leaves of an underlying (usually unknown) rooted tree (which need not be binary). A pair of choices c_1, c_2 is

[3] For instance, let the distance between every pair of elements be the size of the smallest cluster in the hierarchy containing both of them.

independent of a third choice c_3 if there is a (rooted) subtree containing c_1 and c_2, but not c_3. Using the terminology of this paper, the pair c_1, c_2 is independent of c_3 if c_1 and c_2 are closer to each other than to c_3.

In a recent paper, Benson et al. [12] considered the problem of algorithmically recovering the underlying tree structure from an observed set of choices. In particular, they showed that using $O(n^2)$ adaptive ordinal queries, one can recover the tree. In fact, their algorithm is a deterministic version of our QuickSort-based algorithm from Section 3. As we discuss in Section 6, for trees of very high degree, the performance of the generalization of our algorithm deteriorates to $\Theta(n^2)$, matching the guarantee of [12]. However, for low-degree trees, and in particular, for binary trees, our algorithm is provably more efficient.

Yet another application of the algorithmic problem is *inferring underlying causal structures*. Pearl and Tarsi [34] introduced a tree-like model where the leaves are observable random variables and the internal nodes are hidden causes. They considered a statistical test over the observable variables (i.e., leaves) which is equivalent to the notation of ordinal query we discuss in this article. They showed that using $O(n \log n)$ statistical tests, one can discover the underlying causal tree if the degrees are bounded by a constant. We discuss their algorithm in Section 4 as the foundation of our main result in Section 5.

Feige et al. [18] studied several problems under the independent noise model. In particular, they showed that for the classical Binary Search problem, if comparisons are noisy (i.e., each comparison is correct independently with probability $p > \frac{1}{2}$ and flipped otherwise), there exists an algorithm that uses $O(\log n)$ queries and succeeds with high probability. Ben-Or and Hassidim [11] improved this result and obtained the information-theoretically optimal query complexity. Braverman and Mossel [13] studied the sorting problem under an independent noise model in a non-adaptive setting. They proposed an algorithm that, given the noisy comparisons for every pair of elements, finds the maximum-likelihood permutation. They also showed that the maximum-likelihood permutation is "close" to the ground truth.

2 Definitions and Preliminaries

2.1 Hierarchical Clusterings

Let $\mathcal{X}$ be a set of $n \geq 2$ elements. A *hierarchical clustering* of $\mathcal{X}$ can be equivalently characterized in two ways:

1. A laminar family $\mathcal{H}$ of subsets[4] (called *clusters*) of $\mathcal{X}$, with the properties that $\emptyset \notin \mathcal{H}$, $\mathcal{X} \in \mathcal{H}$ and $\{x\} \in \mathcal{H}$ for all $x \in \mathcal{X}$.

2. A rooted tree $\mathcal{T}$ whose leaves are the elements $x \in \mathcal{X}$.

The equivalence is obvious: the clusters in $\mathcal{H}$ exactly correspond to vertices v in $\mathcal{T}$, or more specifically, to the set of leaves in the subtree rooted at v. We call the set of all clusters corresponding to rooted subtrees of $\mathcal{T}$ the *hierarchical clustering corresponding to* $\mathcal{T}$, and denote it by $\mathcal{H}[\mathcal{T}]$. When there is no risk of confusion, we also refer to $\mathcal{T}$ itself as a hierarchical clustering.

We primarily use the representation in terms of the tree $\mathcal{T}$. In keeping with much of the prior literature, ([15, 14, 19], for example) we focus (except in Section 6) on the case when $\mathcal{T}$ is binary. In terms of the representation $\mathcal{H}$, this means that for each $X \in \mathcal{H}$ with $|X| \geq 2$, there exists a set $A \subsetneq X$ with $A, X \setminus A \in \mathcal{H}$. We assume without loss of generality that $\mathcal{T}$ has no internal node with one child, since such a node could be removed without changing the corresponding family of sets.

[4]Recall that a family of sets is *laminar* if $A, B \in \mathcal{H}$ and $A \cap B \neq \emptyset$ implies that $A \subseteq B$ or $B \subseteq A$.

There is an unknown ground truth hierarchical clustering $\mathcal{T}^*$, which an algorithm is to recover using ordinal queries. The algorithm succeeds if it finds a rooted binary tree $\mathcal{T}$ with $\mathcal{H}[\mathcal{T}] = \mathcal{H}[\mathcal{T}^*]$; this corresponds to the rooted binary trees $\mathcal{T}$ and $\mathcal{T}^*$ being isomorphic (when the ordering of left vs. right subtrees is immaterial). In this case, we also write $\mathcal{T} \equiv \mathcal{T}^*$.

For a set $S \subseteq \mathcal{X}$ with $|S| \geq 2$, the *induced hierarchy* $\mathcal{H}[S]$ is defined in the obvious way: $\mathcal{H}[S] = \{X \cap S \mid X \in \mathcal{H}\}$. We define the *induced tree* $\mathcal{T}[S]$ as follows: remove from $\mathcal{T}$ all leaves not in S, then repeatedly remove all internal nodes without children, and contract all nodes with a single child.[5]

Proposition 2.1 *$\mathcal{H}[S]$ is the hierarchy corresponding to the tree induced by S, i.e., $\mathcal{H}[S] = \mathcal{H}[\mathcal{T}[S]]$.*

In reasoning about trees, we use the following notation. For an *internal* vertex v, let $\mathcal{T}_L^v, \mathcal{T}_R^v$ be the subtrees rooted at v's left and right children, and let $\overline{\mathcal{T}}^v = \mathcal{T} \setminus (\mathcal{T}_L^v \cup \mathcal{T}_R^v)$ be the set of all other vertices (so $v \in \overline{\mathcal{T}}^v$). As is standard for *rooted* trees, we consider the *degree* of a node to be its number of children, i.e., we do not count its parent.

2.2 Ordinal Queries

An *ordinal* query is a set of three distinct elements $\{x, x', x''\} \subseteq \mathcal{X}$. The response to the query reveals which two elements are "closest" to each other. We say that x and x' are closer to each other than to x'' (with respect to $\mathcal{T}$) if there exists a cluster (subtree rooted at an internal node) in $\mathcal{T}$ which contains x and x', but not x''. In other words, for every set of three elements, the two elements which have the lowest common ancestor are closest. Because $\mathcal{T}$ is a binary tree, such a cluster — and hence a valid response — always exists and is unique.

Proposition 2.2 *Two hierarchical clusterings $\mathcal{T}$, $\mathcal{T}'$ of the same elements $\mathcal{X}$ satisfy $\mathcal{T} \equiv \mathcal{T}'$ if and only if for every triplet $\{x, x', x''\}$, the query responses with respect to $\mathcal{T}$ and $\mathcal{T}'$ are the same.*

Thus, using at most $\binom{n}{3} = \Theta(n^3)$ ordinal queries, an algorithm can uniquely recover the underlying hierarchy $\mathcal{T}^*$. (The actual algorithm is a simple bottom-up algorithm.) For non-adaptive algorithms, the following easy theorem (proved in Appendix A) gives a matching lower bound.

Theorem 2.3 *Every non-adaptive algorithm requires $\Omega(n^3)$ ordinal queries to learn the hierarchical clustering over n elements (even in the absence of noise).*

Our goal is to reduce the number of queries to $O(n \log n)$ using adaptive ordinal queries. The algorithm has access to an oracle (for example, a human user) that will answer ordinal queries based on the (otherwise unknown) ground truth hierarchy $\mathcal{T}^*$. For the next two sections, we assume that these answers are correct; in Section 5, we turn our attention to the *independent noisy* model. In this model, there is a known constant $p > \frac{1}{2}$ such that each query response is correct *independently* with probability p, and adversarially incorrect[6] with probability $1 - p$. Because the query responses are independent, if an algorithm queries the same triplet multiple times, it may get different responses.

In several of our algorithms, we will try to locate where within an existing (partial) hierarchy $\mathcal{T}$ another element x should be inserted. In doing so, it is particularly useful to "query" an internal vertex v, and learn whether x should be in the left subtree of v, the right subtree of v, or neither.

[5] If the root has one child, it is removed, and its child becomes the root of the tree.

[6] Of course, the adversary could give a correct answer if an algorithm were to rely on it always answering incorrectly.

This can be accomplished as follows. Let $x_L \in \mathcal{T}_L^v, x_R \in \mathcal{T}_R^v$ be arbitrary elements in the left and right subtrees of v. (x_L, x_R exist because v is an internal vertex, and thus has exactly two children.) We call the query $\{x_L, x_R, x\}$ an *ordinal query for x with pivot v*. Its possible outcomes can be interpreted as follows.

Proposition 2.4 *If the response to $\{x_L, x_R, x\}$ is that x_L and x are closest, then x's correct location is in $\mathcal{T}_L^v$ (the left subtree of v). If the response to $\{x_L, x_R, x\}$ is that x_R and x are closest, then x's correct location is in $\mathcal{T}_R^v$. If x_R and x_L are closest, then x's correct location is not in the subtree rooted at v, i.e., x belongs in $\overline{\mathcal{T}}^v$.*

Ordinal queries with pivot v are only defined when v is an internal node, and we will only use them in that case. When notationally convenient, we will consider the left/right child of v itself as the response to an ordinal query with pivot v.

2.3 Vertex Queries

Ordinal queries with pivot v provide "directional" information regarding the location of x with respect to v. As such, they are similar to (though subtly different from) *vertex queries* defined in past work [33, 16]. In the vertex query model, the goal is to identify an a priori unknown *target node* t in a known (undirected, not necessarily rooted) tree $\mathcal{T}$, by repeatedly querying nodes v of $\mathcal{T}$. In response to a vertex query, the algorithm is told that $t = v$, or otherwise is given the unique neighbor that is closer to the target.

Vertex queries are subtly more powerful than ordinal queries with a pivot, because they can identify v as the target. We will show later how to simulate (for our purposes) vertex queries with a constant number of ordinal queries with pivots. In fact, our analysis of the algorithm inspired by InsertionSort is based on ideas from prior work in the vertex query model, and the analysis for the independent noisy model explicitly uses a reduction from the ordinal query model to the vertex query model.

3 A QuickSort-like Randomized Algorithm

In this section, we propose a simple randomized algorithm (Algorithm 1), reminiscent of QuickSort. The algorithm uses $O(n \log n)$ ordinal queries in expectation to learn the hierarchy over n elements. Algorithm 1 randomly partitions the elements into three groups A, B, and C (lines 4–10), until all of the partitions are small enough. Once they are, the algorithm recursively determines the hierarchy for each partition, and finally merges them.

The partitioning is akin to the partitioning stage in QuickSort: it draws a pair of distinct *pivot elements* $x_A, x_B \in X$, uniformly at random. It then partitions the elements of X into three sets. These sets can be characterized as follows: Let $\overline{v}$ be the lowest common ancestor of x_A, x_B in the ground truth hierarchy $\mathcal{T}^*$, with children v_A, v_B. Then, A (B) consists of all elements in the subtree of $\mathcal{T}^*$ rooted at v_A (v_B), while C consists of all remaining elements: exactly the elements outside of the subtree of $\mathcal{T}^*$ rooted at $\overline{v}$. Therefore, after the recursive calls, $\mathcal{T}_A$ and $\mathcal{T}_B$ are the subtrees of $\mathcal{T}^*$ rooted at v_A and v_B, while $\mathcal{T}_C$ is the result of removing $\overline{v}$ and its subtree from $\mathcal{T}^*$ (and contracting its parent).

The role of the Merge function (Algorithm 2) is then to insert $\overline{v}$ and its subtree in the correct place in $\mathcal{T}_C$; the primary task here is to correctly identify the sibling v of $\overline{v}$. The algorithm does this by starting at the root and repeatedly using ordinal queries for a representative element x of

Algorithm 1 QUICKCLUSTERING (X)

1: **if** $|X| \leq 2$ **then**
2: **return** the obvious hierarchy.
3: **repeat**
4: Let (x_A, x_B) be a pair of distinct elements chosen uniformly at random.
5: **for** each $x \in X \setminus \{x_A, x_B\}$ **do**
6: Make an ordinal query of $\{x_A, x_B, x\}$.
7: Let $A \leftarrow \{x_A\} \cup \{x \in X \mid x$ and x_A are closer to each other than to $x_B\}$.
8: Let $B \leftarrow \{x_B\} \cup \{x \in X \mid x$ and x_B are closer to each other than to $x_A\}$.
9: Let $C \leftarrow \{x \in X \mid x_A$ and x_B are closer to each other than to $x\}$.
10: **until** $\max(|A|, |B|, |C|) \leq \frac{15}{16} \cdot |X|$.
11: $\mathcal{T}_A \leftarrow$ QUICKCLUSTERING(A).
12: $\mathcal{T}_B \leftarrow$ QUICKCLUSTERING(B).
13: $\mathcal{T}_C \leftarrow$ QUICKCLUSTERING(C).
14: **return** MERGE $(\mathcal{T}_A, \mathcal{T}_B, \mathcal{T}_C)$.

Algorithm 2 MERGE $(\mathcal{T}_A, \mathcal{T}_B, \mathcal{T}_C)$

1: Add a vertex $\overline{v}$ whose children are the roots of $\mathcal{T}_A$ and $\mathcal{T}_B$.
2: Let x be an arbitrary element (i.e., leaf) in $\mathcal{T}_A$ or $\mathcal{T}_B$.
3: Let v be the root of $\mathcal{T}_C$.
4: **while** v is not a leaf, and the ordinal query for x with pivot v does not return $\overline{\mathcal{T}}^v$ **do**
5: Update v to the child of v returned by that ordinal query for x
6: Insert a new vertex v' as parent of v, and make $\overline{v}$ its other child.

$\mathcal{T}_A \cup \mathcal{T}_B$ with v as a pivot, and following the direction returned by the ordinal query. That inserting $\overline{v}$ as a sibling of v (with a new parent v') is correct follows from Proposition 2.1.

We next analyze the expected number of ordinal queries required by the QUICKCLUSTERING algorithm. The number of queries required for MERGE is at most[7] $|X|$, and the number of queries for each iteration of the partitioning loop (lines 4–10 in Algorithm 1) is $|X| - 2$. The key observation is that in expectation, the loop is executed only a constant number of times before succeeding, captured by Lemma 3.1.

Lemma 3.1 *The probability that in line 10 of Algorithm 1, at least one A, B, C is larger than $\frac{15}{16} \cdot |X|$ is at most $\frac{125}{128}$.*

Proof. We first prove that there exist clusters C and C' with $C' \subseteq C$ in $\mathcal{H}[\mathcal{T}]$ such that $\frac{n}{16} < |C'| \leq \frac{n}{8}$ and $\frac{n}{4} < |C| \leq \frac{n}{2}$. This follows because $\mathcal{T}$ is binary: starting from $C = \mathcal{X}$, we can repeatedly consider a bipartition of C into two disjoint subclusters. While C has more than $\frac{n}{2}$ elements, we can replace it by the larger of the two subclusters, which must have at least half the size of C. We end up with a set C of the required size. Continuing from $C' = C$ in the same way until C' has size at most $\frac{n}{8}$ gives us the claimed sets.

[7] In fact, using the techniques from Section 4, the bound could be improved to $1 + \log_2 |X|$; but this improvement is drowned out by the number of queries for the partitioning.

The probability that one of x_A, x_B is in C' and the other in $C \setminus C'$ is at least

$$\frac{|C'| \cdot |C \setminus C'|}{\binom{n}{2}} > 2 \cdot \frac{1}{16} \cdot \frac{3}{16} = \frac{3}{128}.$$

When this event happens, we assume w.l.o.g. that $x_A \in C'$ and $x_B \in C \setminus C'$. Then, $C' \subseteq A \subseteq C$, and therefore $\frac{n}{16} \leq |A| \leq \frac{n}{2} \leq \frac{15n}{16}$. Because $|A| \geq \frac{n}{16}$, we also get that $|B|, |C| \leq \frac{15n}{16}$, completing the proof. ■

By Lemma 3.1 and the preceding discussion, the total number of ordinal queries for both partitioning and merging is $O(|X|)$. Letting $Q(n)$ denote the expected number of comparisons for QuickClustering on n elements, we obtain the recurrence $Q(n) \leq Q(|A|) + Q(|B|) + Q(|C|) + O(n)$. Because $|A| + |B| + |C| = n$ and $\max(|A|, |B|, |C|) \leq \frac{15}{16} \cdot n$, the solution of this recurrence is $Q(n) = O(n \log n)$. Hence, we have proved the following theorem.

Theorem 3.2 *Algorithm 1 learns the hierarchy using $O(n \log n)$ ordinal queries in expectation.*

The query complexity of Algorithm 1 is asymptotically optimal. Because the hierarchical clusterings include all permutations of $\mathcal{X}$, there are at least $2^{\Omega(n \log n)}$ different hierarchical clusterings of n elements (see also [38]). More precisely, as pointed out in [19], there are $(2n-3) \cdot (2n-5)!! = \frac{(2n-3)!}{(n-2)! \cdot 2^{n-2}}$ hierarchical clusterings. By Stirling's approximation, at least $n \log_2 n - O(n)$ bits of information are necessary to uniquely identify $\mathcal{T}^*$. Since each query response reveals at most $\log_2(3)$ bits of information, the number of queries required is at least $n \log_3 n - O(n)$.

4 An InsertionSort-like Algorithm without Noise

In this section, we present and analyze the algorithm of Pearl and Tarsi [34]. This algorithm is reminiscent of InsertionSort. It is also essentially the same as an algorithm proposed by Tamuz et al. [36], who only considered it when the ground-truth tree is balanced and thus has logarithmic height. We give a self-contained analysis of this algorithm as it is the foundation of our main result in Section 5, and the analysis allows us to introduce key concepts and abstractions.

Algorithm 3 considers the elements in an arbitrary order $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$. In each iteration i, element x_i is inserted into the hierarchy $\mathcal{T}_{i-1}$ for the preceding $i-1$ elements. Proposition 4.1 (whose proof follows directly from Proposition 2.1) shows that x_i will always be inserted as a leaf sibling of an existing (leaf or internal) node v_i, by inserting a new common parent of x_i and v_i.

Proposition 4.1 *Let $C \in \mathcal{H}[x_1, \ldots, x_{i-1}]$ be minimal with the property that $C \cup \{x_i\} \in \mathcal{H}[x_1, \ldots, x_i]$. (Informally speaking, $C \subseteq \{x_1, \ldots, x_{i-1}\}$ is the cluster corresponding to the sibling of x_i in $\mathcal{T}^*[x_1, \ldots, x_i]$.) In $\mathcal{T}^*[x_1, \ldots, x_{i-1}]$, let v_i be the root of the subtree corresponding to the cluster C. Then, $\mathcal{T}^*[x_1, \ldots, x_i]$ is obtained from $\mathcal{T}^*[x_1, \ldots, x_{i-1}]$ by inserting a new internal node v' as the parent of v_i, and making the leaf x_i its other child.*

When $\mathcal{T}^*$ has logarithmic height, FindSibling can be simply implemented as a top-down search similar to the Merge algorithm in Section 3. This approach can take linear time if the tree is not balanced, and our main focus in this section is on improving the time to $\log n$ for arbitrary trees.

Assuming that the function FindSibling (to be specified below) correctly identifies v_i, the algorithm maintains the following invariants:

1. Each internal node of $\mathcal{T}_i$ has two children; this holds because every time an internal node is added, it is explicitly given two children.

Algorithm 3 InsertionClustering $(x_1, \ldots, x_n)$

Let $\mathcal{T}_2$ be the unique (trivial) hierarchical clustering of elements x_1, x_2.
for $i = 3, \ldots, n$ **do**
  Let $v_i = \textsc{FindSibling}(\mathcal{T}_{i-1}, x_i)$.
  Let v_i' be the parent of v_i (or $\emptyset$ if v_i is the root)
  Let $\mathcal{T}_i$ be the tree obtained from $\mathcal{T}_{i-1}$ by adding a new vertex v' with children v_i and x_i and parent v_i'.
return $\mathcal{T}_n$

2. $\mathcal{T}_i$ is always the correct hierarchy for $\{x_1, x_2, \ldots, x_i\}$, in the sense that $\mathcal{T}_i \equiv \mathcal{T}^*[x_1, \ldots, x_i]$; this follows inductively from Proposition 4.1.

Thus, it remains to specify an efficient implementation of the function FindSibling and analyze the overall number of ordinal queries.

FindSibling is very similar to "Binary Search in Trees" [32, 33, 16] in the *vertex query* model. Recall that in the vertex query model, an unknown node t is to be located in a given and known tree $\mathcal{T}$. When an algorithm queries node v which is not the target, it learns the first edge of the unique v-t path (equivalently, the connected component of $\mathcal{T} \setminus \{v\}$ in which the target lies). The "Binary Search" algorithm is based on the following observation of Jordan [26]: each tree $\mathcal{T}$ has a separator node v with the property that each connected component of $\mathcal{T} \setminus \{v\}$ contains at most half the nodes of $\mathcal{T}$. Repeatedly querying such a separator node of the remaining subtree is easily shown to find the target t in a tree of n nodes using at most $\log_2 n$ vertex queries [33].

FindSibling treats the (unknown) sibling v_i as the target to discover, and uses ordinal queries with internal vertices as pivots (see Section 2.2) to essentially simulate vertex queries. It maintains a set S of candidate vertices v that have not been ruled out from being v_i.

Algorithm 4 FindSibling $(\mathcal{T}, x)$

Let $S = V(\mathcal{T})$ be the set of all vertices of $\mathcal{T}$.
while $|S| > 1$ **do**
  Let v be a *pivot vertex* minimizing $\max(|\mathcal{T}_L^v \cap S|, |\mathcal{T}_R^v \cap S|, |\overline{\mathcal{T}}^v \cap S|)$.
  Make an ordinal query for x with pivot v.
  if the query returns the left subtree of v **then**
    Let $S = S \cap \mathcal{T}_L^v$.
  else if the query returns the right subtree of v **then**
    Let $S = S \cap \mathcal{T}_R^v$.
  else
    Let $S = S \cap \overline{\mathcal{T}}^v$.
return the unique vertex v in S.

Lemma 4.2 *When all answers to ordinal queries are correct,* $\textsc{FindSibling}(\mathcal{T}, x)$ *finds* v_i *in* $\mathcal{T}$ *using at most* $\log |\mathcal{T}|$ *ordinal queries.*

Proof. Inductively, the induced subgraph $\mathcal{T}[S]$ always forms a binary tree in which each internal node has two children. In verifying the correctness of the algorithm, notice first that the pivot

vertex v is indeed never a leaf. That is because for any leaf v, we would have $\overline{\mathcal{T}}^v \cap S = S$, whereas for v's parent v', we would have that $\max(|\mathcal{T}_L^{v'} \cap S|, |\mathcal{T}_R^{v'} \cap S|, |\overline{\mathcal{T}}^{v'} \cap S|) < |S| = |\overline{\mathcal{T}}^v \cap S|$. The fact that the update of S is correct follows from Proposition 2.4.

Next, we analyze the number of ordinal queries made. By the result of Jordan [26], $\mathcal{T}[S]$ has a separator node v with the property that each of $\mathcal{T}_L^v \cap S, \mathcal{T}_R^v \cap S, (\overline{\mathcal{T}}^v \cap S) \setminus \{v\}$ contains at most $|S|/2$ nodes. Because in the third case (x_L and x_R are closest), the node v remains in S, the size of S might change to $1 + |S|/2$ instead of $|S|/2$. However, notice that this can only happen when $|S|$ is odd. For when $|S|$ is even, if $\overline{\mathcal{T}}^v$ contained $1 + |S|/2$ nodes from S, then choosing the parent v' of v instead would strictly decrease the size of $\overline{\mathcal{T}}^{v'}$, while the size of each of the subtrees of v' is bounded by $|S|/2$. Thus, we obtain that the new size of S is at most $\lceil |S|/2 \rceil$.

When $|S| = 3$, a single query suffices to identify v_i; the case $|S| = 4$ cannot arise, as there is no binary tree with 4 nodes in which all internal nodes have degree 2. Thus, the number $Q(k)$ of required ordinal queries satisfies the recurrence $Q(k) \leq 1$ for $k \leq 4$ and $Q(k) \leq 1 + Q(\lceil k/2 \rceil)$ for $k \geq 5$. An easy induction proof shows that $Q(k) \leq \log k$, so the number of ordinal queries required to find v_i in $\mathcal{T}$ is at most $\log |\mathcal{T}|$. ■

Theorem 4.3 *When the answers to all ordinal queries are correct, Algorithm 3 learns the correct hierarchical clustering of n elements using at most $n \log_2 n$ ordinal queries.*

Proof. The correctness of the algorithm follows by repeatedly applying Proposition 4.1, and using that by Lemma 4.2, each iteration uses the correct v_i.

By Lemma 4.2, because $\mathcal{T}_i$ has $2i - 1$ nodes (i element leaves and $i - 1$ internal nodes), inserting element x_i into $\mathcal{T}_{i-1}$ for $i = 3, \ldots, n$ requires at most $\log_2(2i - 3)$ ordinal queries. Thus, the total number of ordinal queries is at most

$$\sum_{i=3}^{n} \log_2(2i-3) < \sum_{i=2}^{n-1} (1 + \log_2 i) < n + \log_2((n-1)!)$$
$$\overset{(*)}{\leq} n + (n \log_2 n - n \log_2 e + O(\log n)) < n \log_2 n.$$

The step labeled (*) used Stirling's inequality. ■

5 Dealing with Noisy Feedback

In this section, we turn our attention to the noisy model: the response to any ordinal query is correct independently with probability $p > \frac{1}{2}$, and adversarially incorrect otherwise. Our main result is the following theorem:

Theorem 5.1 *When each query response is correct with constant probability $p > \frac{1}{2}$, and adversarially incorrect with probability $1 - p$, there is an algorithm that uses $O(n \log n + n \log(1/\delta))$ ordinal queries, and learns the correct hierarchical clustering with probability at least $1 - \delta$.*

Since ordinal queries are only used in the call to FindSibling (in Line 3) inside the algorithm InsertionClustering, it suffices to show how to find v_i in each iteration with high probability using $O(\log n + \log(1/\delta))$ ordinal queries. This is guaranteed by the following lemma, the focus of this section.

Lemma 5.2 *Assume that each response to an ordinal query is correct independently with probability $p > \frac{1}{2}$, and adversarially incorrect with probability $1 - p$. There exists an adaptive algorithm with the following property: Given a hierarchical clustering $\mathcal{T}$ with i leaves, an element x to insert, and an error guarantee $\delta > 0$, the algorithm finds the sibling $\overline{v}$ of x using at most[8] $O(\log i + \log(1/\delta))$ ordinal queries.*

Proof of Theorem 5.1. Using Lemma 5.2, the proof of Theorem 5.1 is straightforward: set $\delta' = \delta/n$ and apply Lemma 5.2 to each invocation of FindSibling. Then, by the union bound, all iterations will succeed with probability at least $1 - \delta$. The total number of ordinal queries will be at most

$$\sum_{i=3}^{n} O(\log(2i-3) + \log(1/\delta')) \leq n \cdot O(\log n + \log(n/\delta)) = O(n(\log n + \log(1/\delta))). \blacksquare$$

5.1 Simulating Vertex Queries with Ordinal Queries

To avoid essentially repeating a large amount of prior analysis, we begin by making the connection between ordinal queries and vertex queries more explicit: we show how to simulate one vertex query using a constant number of ordinal queries.

To illustrate the reduction most easily, consider the case $p = 1$ in which all queries are answered correctly. To simulate a vertex query to the root, simply make an ordinal query for x with the root as pivot. This one query suffices, since the response reveals the subtree that x belongs to, or otherwise that the root is the sibling of x (if x does not belong to either of the subtrees rooted at the root's children). To simulate a vertex query to a leaf node v, instead make an ordinal query with the parent v' of v as the pivot. If the response points to v, then return v as the sibling; otherwise, x is not the sibling of v.

Finally, to simulate a vertex query to an internal vertex v that is not the root, let v' be the parent of v in $\mathcal{T}$, and make two ordinal queries for x, one with pivot v and the other with pivot v'. If in response to the ordinal query with pivot v, a child of v is identified as containing x in its subtree, then this case directly corresponds to the corresponding vertex query response at v. If $\overline{\mathcal{T}}^v$ is known to contain x, then the response to the query with pivot v' clarifies whether the target is in fact v itself, or contained in $\overline{\mathcal{T}}^v \setminus \{v\}$. Thus, the responses to the two queries together contain (at least) all the information obtained from the vertex query. (Notice that applying this reduction, we could have obtained the result of Lemma 4.2 with a loss of a factor of 2 in the number of queries required.)

We now show how to generalize this idea to $p \in (\frac{1}{2}, 1)$. The cases of the root vertex and a leaf node are handled exactly as in the case $p = 1$; in particular, using only a single ordinal query. To simulate a vertex query to an internal vertex v, we choose a suitably large constant k_p (determined below) and make k_p ordinal queries for x with pivot v. If a strict majority of these queries places x in one of the subtrees below v, then that subtree is returned. If a strict majority of the queries with pivot v places x in $\overline{\mathcal{T}}^v$, then v's parent v' is queried k_p times to determine if the desired target is equal to v or contained in $\overline{\mathcal{T}}^v \setminus \{v\}$. The former is returned as the answer if a strict majority of the responses place x in the subtree of v' also containing v. The latter is returned when a strict majority places x in the other subtree below v', or a strict majority place it in $\overline{\mathcal{T}}^{v'}$. Finally, in all other cases, the responses are inconclusive, and the simulation algorithm returns an arbitrary response. Formally, the algorithm (which has a number of tedious case distinctions) is given as Algorithm 5.

[8] The big-O notation hides constants depending on p, but not on any other parameter.

Algorithm 5 SimulateVertexQuery $(\mathcal{T}, v, x)$

if v is the root of $\mathcal{T}$ **then**
  Make one ordinal query for x with pivot v, and return its result.
  (If $\overline{\mathcal{T}}^v = \{v\}$, the response reveals the root as the proposed answer.)
else if v is a leaf **then**
  Let v' be the parent of v in $\mathcal{T}$.
  Make one ordinal query for x with pivot v'.
  if the response places x at v **then**
    return the vertex v as a proposed answer.
  else
    return the direction towards v'.
else
  Let v' be the parent of v and k_p be an odd constant such that $1 - e^{-k_p(2p-1)^2/2} \geq \sqrt{p}$.
  Make k_p ordinal queries for x with pivot v.
  if a strict majority of responses place x in $\mathcal{T}_L^v$ **then**
    return the direction towards the left child of v.
  else if a strict majority of responses place x in $\mathcal{T}_R^v$ **then**
    return the direction towards the right child of v.
  else if a strict majority of responses place x in $\overline{\mathcal{T}}^v$ **then**
    Make k_p ordinal queries for x with pivot v'.
    if a strict majority of responses place x in the subtree of v' containing v **then**
      return the vertex v as a proposed answer.
    else if a strict majority of responses place x in the other subtree of v' or $\overline{\mathcal{T}}^{v'}$ **then**
      return the direction towards v'.
    else
      return an arbitrary response.
  else
    return an arbitrary response.

Lemma 5.3 *If each ordinal query is answered correctly independently with probability p, then (for a suitable choice of k_p, specified below),* SimulateVertexQuery $(\mathcal{T}, v, x)$ *returns the correct response (either v or the direction in which to search) to the vertex query for v, with probability at least p.*

Proof. By the additive Hoeffding Bound [22], the probability that a majority out of k query responses is correct for every $p > \frac{1}{2}$ is at least $q(k) := 1 - e^{-k(2p-1)^2/2}$. For odd k, any majority is strict. For any $p > \frac{1}{2}$, we have that $e^{-(2p-1)^2/2} < 1$, so $q(k) \to 1$ as $k \to \infty$. In particular, $q(k) \geq \sqrt{p}$ for k sufficiently large. Fixing such a k_p, the probability that a strict majority of the queries both with pivots v, v' are correct is at least $(\sqrt{p})^2 = p$. Whenever strict majorities of both sets of ordinal queries are correct, the output of SimulateVertexQuery is correct, completing the proof. ■

5.2 Reanalyzing Binary Search with Noisy Vertex Queries

In light of Lemma 5.3, for the purpose of analysis in this section, we can ignore ordinal queries, and instead focus on finding x's sibling $\overline{v}$ using vertex queries: in response to querying v, the algorithm is told either that $v = \overline{v}$ is the correct sibling of x, or is given a subtree (rooted at one of the children of v, or obtained by removing v and its subtrees) that must contain x. This information is correct with probability (at least) p.

[16] gave an algorithm for Binary Search on undirected graphs with noisy responses that solves this problem with probability at least $1 - \delta$, using at most $O(\log n + \log^2(1/\delta))$ vertex queries [16, Theorem 8]. Since we need $\delta = O(1/n)$ to be able to take a Union Bound, applying this result directly would only give us an overall guarantee of $O(n \log^2 n)$. Hence, in this section, we show how to combine algorithms of [16, Algorithm 2] and of Feige et al. [18] (see also related algorithms for noisy Binary Search on the line [11, 27]) to obtain an algorithm finding $\overline{v}$ in $O(\log n + \log(1/\delta))$ queries. In contrast to [16], the dependence of this new algorithm's number of queries on p is not information-theoretically optimal, but it obtains an improved dependence on δ in return.

Algorithm 2 of [16] is a variant of a multiplicative weights algorithm which starts with a candidate set S of vertices, and within essentially $O(\log |S|)$ rounds reduces it to only $O(\log |S|)$ "likely" remaining candidates. The overall algorithm in [16] works by first reducing the candidate set of *all* nodes to just $O(\log n)$ nodes, then reducing that further to $O(\log \log n)$ nodes with a second invocation of the multiplicative weights algorithm, and finally querying each remaining candidate frequently enough to identify the correct node with sufficiently high probability. (Dependencies on δ are omitted in our high-level overview.) As part of our algorithm here, we just use a single invocation of the Multiplicative Weights algorithm (Algorithm 2 of [16]), with parameters modified to obtain a better dependence on δ at the cost of a worse dependence on p. The performance of of this algorithm (used as a Black Box here) is summarized by Lemma 5.4, which is obtained from Lemma 5 of [16] by setting $\lambda = \frac{1+p\log p+(1-p)\log(1-p)}{2\log(p/(1-p))}$.

Lemma 5.4 *There exists an algorithm which uses at most $O(\log n + \log(1/\delta))$ vertex queries and returns a set S of $O(\log n + \log(1/\delta))$ nodes such that the unknown target is contained in S with probability at least $1 - \delta$.*

Once the number of candidate targets has been reduced to $O(\log n)$, due to the $\log(1/\delta)$ dependence, a further reduction using the techniques of [16] seems difficult. However, at this point, we can apply a modification of a beautiful algorithm of Feige et al. [18]. Translated into our nomenclature, Lemma 3.1 of [18] about noisy Binary Search can be phrased as follows:

Lemma 5.5 (Lemma 3.1 of [18], restated) *Let $\mathcal{T}$ be a binary tree, and assume that one of the leaves of $\mathcal{T}$ is the unknown target. In response to a vertex query, assume that the algorithm is given the correct answer with probability p, and an adversarially incorrect one otherwise. Then, there is an algorithm which finds the correct target with probability at least $1 - \delta$, using $O(D + \log(1/\delta))$ queries where D is the diameter of $\mathcal{T}$.*

Notice two minor inconveniences in applying Lemma 5.5 for our purposes:

1. In the specific application of [18], the binary tree is complete and thus, D is logarithmic in size of the tree. In our application, on the other hand, after invocation of the Multiplicative Weights algorithm, the number of candidates (and hence diameter of the tree) is bounded by $O(\log n)$.

2. [18] assume that the target is always a leaf node, whereas in our application, the unknown sibling $\overline{v}$ may be an internal node. It is very straightforward to modify the algorithm of Feige et al. to handle non-leaf targets.

For completeness, we present the modified algorithm and corresponding analysis in Appendix B. Its guarantee is summarized by the following lemma:

Lemma 5.6 *Assume that each query response is correct independently with probability $p > \frac{1}{2}$, and adversarially incorrect with probability $1-p$. There exists an adaptive algorithm with the following property: Given a tree $\mathcal{T}$ of diameter D, and error parameter $\delta > 0$, the algorithm finds the target with probability at least $1-\delta$, using at most $O(D + \log(1/\delta))$ vertex queries.*

Using Lemmas 5.4 and 5.6, it is easy to see how to find a target in $O(\log n + \log(1/\delta))$ queries. Set $\delta' = \delta/2$ and apply Lemma 5.4. Within $O(\log n + \log(1/\delta')) = O(\log n + \log(1/\delta))$ iterations, we obtain a set S of at most $O(\log n + \log(1/\delta))$ nodes that contains the target with probability at least $1-\delta'$. Then run the algorithm of Lemma 5.6 with parameter δ' on the tree $\mathcal{T}[S]$. If the target was in S, the algorithm will find it with probability at least $1-\delta'$. Since the height of $\mathcal{T}[S]$ is at most $|S| = O(\log n + \log(1/\delta))$, the number of queries required in this phase is at most $O(\log n + \log(1/\delta))$. Thus, the total number of queries is $O(\log n + \log(1/\delta))$, and the failure probability is at most $2\delta' = \delta$.

6 Conclusion and Open Problems

We presented adaptive algorithms for learning a hierarchical clustering from $O(n \log n)$ ordinal queries when query responses are noisy. In the absence of noise, there is an upper bound of $n \log_2 n$ (Section 4), and we gave a lower bound of $n \log_3 n - O(n)$. An immediate question is whether the constant in $n \log_2 n$ is correct in the noiseless case, or could be improved.

For all our results, we assumed that the tree $\mathcal{T}$ capturing the hierarchical clustering is *binary*. When nodes can have larger degrees, our analysis does not continue to hold. One problem is that for an ordinal query, another response of $\perp$ may be required: when x, x', x'' are leaves in three distinct subtrees of the same internal node v, no pair of them is closer to each other than the third. This issue also manifests itself in a higher query complexity, as pointed out by [12]: when $\mathcal{T}$ is a star in which one leaf has been replaced with an internal node with two new leaves, and the assignment of elements to leaves is uniformly random, it is not difficult to show that any non-adaptive or adaptive algorithm (even randomized) requires $\Omega(n^2)$ ordinal queries to learn $\mathcal{T}$.

When the maximum degree d of $\mathcal{T}$ in not a constant, Pearl and Tarsi [34] pointed out that their InsertionSort-based algorithm can be generalized to obtain an upper bound of $O(d \cdot n \log n)$ on the number of queries. However, this generalized algorithm is not optimal: an algorithm of [16] for the *edge query model* can be adapted (using techniques similar to the ones in Section 4) to insert a new element x into $\mathcal{T}$ using $O(\frac{d}{\log d} \cdot \log n)$ ordinal queries. Hence, a hierarchical clustering of maximum degree d over n elements can be learned using $O(\frac{d}{\log d} \cdot n \log n)$ ordinal queries. A corollary of this result is that every hierarchy can be learned using $O(n^2)$ ordinal queries; this generalizes the algorithmic result of [12]. An interesting open question is whether there is a matching lower bound of $\Omega(\frac{d}{\log d} \cdot n \log n)$ for every d. For $d = 2$ (and more generally $d = O(1)$), our counting argument from Section 3 gives a matching lower bound up to a constant factor, and for $d = n-1$ (and more generally $d = \Omega(n)$), the example mentioned in the preceding paragraph does. We do not have a matching lower bound for intermediate d. Another natural question is whether the upper bound of $O(\frac{d}{\log d} \cdot n \log n)$ ordinal queries can be achieved in the noisy model.

A different interpretation of our result is that we show how to learn a (very restricted) ultrametric on the elements in $\mathcal{X}$ using ordinal queries. This raises the natural question of how well more general metrics can be learned using ordinal queries. Indeed, as mentioned earlier, some past work has considered this question, with some assumption on the geometry of the metric (e.g., [23]). Naturally, even for very simple metrics (such as points on a line), we cannot hope to learn all distances accurately: for example, if one element is further from all other elements than any other pair is from each other, no algorithm can learn its distance to other elements to within any approximation guarantee. Is there a natural ordinal approximation to a (not necessarily Euclidean) metric that can be obtained efficiently? Could techniques such as [9, 10, 17] of probabilistic tree embeddings be useful in this context?

Acknowledgments

Work support by NSF grant 1619458. We would like to thank Daniel Hsu for pointing us to [34]. We also thank Sanjoy Dasgupta, Shanghua Teng, and anonymous reviewers for useful feedback and suggestions.

References

[1] S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D. Kriegman, and S. Belongie. Generalized non-metric multidimensional scaling. In *Proc. 11th Intl. Conf. on Artificial Intelligence and Statistics*, pages 11–18, 2007.

[2] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

[3] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[4] E. Anshelevich. Ordinal approximation in matching and social choice. *ACM SIGecom Exchanges*, 15(1):60–64, July 2016.

[5] E. Anshelevich, O. Bhardwaj, and J. Postl. Approximating optimal social choice under metric preferences. In *Proc. 29th AAAI Conf. on Artificial Intelligence*, pages 777–783, 2015.

[6] P. Awasthi, M.-F. Balcan, and K. Voevodski. Local algorithms for interactive clustering. *Journal of Machine Learning Research*, 18:1–35, 2017.

[7] P. Awasthi and R. B. Zadeh. Supervised clustering. In *Proc. 24th Advances in Neural Information Processing Systems*, pages 91–99, 2010.

[8] M.-F. Balcan and A. Blum. Clustering with interactive feedback. In *Proc. 19th Intl. Conf. on Algorithmic Learning Theory*, pages 316–328, 2008.

[9] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proc. 37th IEEE Symp. on Foundations of Computer Science*, pages 184–193, 1996.

[10] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 161–168, 1998.

[11] M. Ben-Or and A. Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *Proc. 49th IEEE Symp. on Foundations of Computer Science*, pages 221–230. IEEE, 2008.

[12] A. R. Benson, R. Kumar, and A. Tomkins. On the relevance of irrelevant alternatives. In *25th Intl. World Wide Web Conference*, pages 963–973, 2016.

[13] M. Braverman and E. Mossel. Noisy sorting without resampling. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, pages 268–276, 2008.

[14] A. Clauset, C. Moore, and M. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, 2008.

[15] S. Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proc. 48th ACM Symp. on Theory of Computing*, pages 118–127, 2016.

[16] E. Emamjomeh-Zadeh, D. Kempe, and V. Singhal. Deterministic and probabilistic binary search in graphs. In *Proc. 48th ACM Symp. on Theory of Computing*, pages 519–532, 2016.

[17] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[18] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.

[19] J. Felsenstein. *Inferring Phylogenies*. Sinauer, Oxford University Pres, 2004.

[20] A. Goel and D. T. Lee. Triadic consensus. In *Proc. 8th Workshop on Internet and Network Economics (WINE)*, pages 434–447, 2012.

[21] A. Goel and D. T. Lee. Large-scale decision-making via small group interactions: the importance of triads. In *Workshop on Computational Social Choice (COMSOC)*, 2014.

[22] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[23] L. Jain, K. Jamieson, and R. Nowak. Finite sample prediction and recovery bounds for ordinal embedding. In *Proc. 30th Advances in Neural Information Processing Systems*, 2016.

[24] K. G. Jamieson and R. D. Nowak. Active ranking using pairwise comparisons. In *Proc. 25th Advances in Neural Information Processing Systems*, pages 2240–2248, 2011.

[25] K. G. Jamieson and R. D. Nowak. Low-dimensional embedding using adaptively selected ordinal data. In *49th Annual Allerton Conference on Communication, Control, and Computing (Allerton 2011)*, pages 1077–1084, 2011.

[26] C. Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

[27] R. M. Karp and R. Kleinberg. Noisy binary search and its applications. In *Proc. 18th ACM-SIAM Symp. on Discrete Algorithms*, pages 881–890, 2007.

[28] M. Kendall and J. D. Gibbons. *Rank Correlation Methods*. Oxford University Press, 1990.

[29] M. Kleindessner and U. v. Luxburg. Uniqueness of ordinal embedding. In *Proc. 27th Conference on Learning Theory*, pages 40–67, 2014.

[30] D. McFadden, W. B. Tye, and K. Train. *An Application of Diagnostic Tests for the Independence from Irrelevant Alternatives Property of the Multinomial Logit Model.* Institute of Transportation Studies, University of California, 1977.

[31] B. McFee and G. Lanckriet. Learning multi-modal similarity. *Journal of Machine Learning Research*, 12:491–523, 2011.

[32] S. Mozes, K. Onak, and O. Weimann. Finding an optimal tree searching strategy in linear time. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, pages 1096–1105, 2008.

[33] K. Onak and P. Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *Proc. 47th IEEE Symp. on Foundations of Computer Science*, pages 379–388, 2006.

[34] J. Pearl and M. Tarsi. Structuring causal trees. *Journal of Complexity*, 2(1):60–77, 1986.

[35] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Proc. 17th Advances in Neural Information Processing Systems*, pages 41–48, 2003.

[36] O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. T. Kalai. Adaptively learning the crowd kernel. In *Proc. 28th Intl. Conf. on Machine Learning*, pages 673–680, 2011.

[37] L. Van Der Maaten and K. Weinberger. Stochastic triplet embedding. In *IEEE Intl. Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2012.

[38] S. Vikram and S. Dasgupta. Interactive bayesian hierarchical clustering. In *Proc. 33rd Intl. Conf. on Machine Learning*, pages 2081–2090, 2016.

A Proof of Theorem 2.3

Here, we prove Theorem 2.3, restated for convenience.

Theorem 2.3 *Any non-adaptive algorithm requires $\Omega(n^3)$ ordinal queries to learn the hierarchical clustering over n elements (even in the absence of noise).*

Proof. A (possibly randomized) non-adaptive algorithm poses k queries and receives their responses all at once. Let n be a power of 2 and generate a ground truth hierarchical clustering $\mathcal{T}^*$ as a full binary tree in which the leaves form a uniformly random permutation of the elements. For every constant $\delta > 0$, we prove that if the algorithm uses fewer than $n(n-1)(n-2)(1-\delta)/24 = \Omega(n^3)$ ordinal queries, then it fails to uniquely identify the clustering with probability at least $1 - \delta$.

Because $\mathcal{T}^*$ is a full binary tree, there are exactly $n/4$ disjoint clusters of size 4 each (Figure 1). Let C be one of them. The 4 elements in C are partitioned into two clusters of size 2 each. The algorithm can learn this internal structure of C only if one of its queries consists of 3 (out of the 4) elements in C. Because the elements are uniformly randomly shuffled, the probability that the 3 elements of any one query all lie in C is $\frac{24}{n(n-1)(n-2)}$.

Because there are $n/4$ such clusters C, if the algorithm queries k triplets, in expectation, it learns the internal structure of at most $k \cdot \frac{n}{4} \cdot \frac{24}{n(n-1)(n-2)} = \frac{6k}{(n-1)(n-2)}$ clusters of size 4. If the algorithm succeeds with probability at least $1-\delta$, then it learns the internal structure of at least $(1-\delta)\frac{n}{4}$ size-4 clusters in expectation. Thus, $\frac{6k}{(n-1)(n-2)} \geq (1-\delta)\frac{n}{4}$ which implies $k \geq (1-\delta)n(n-1)(n-2)/24$. ■

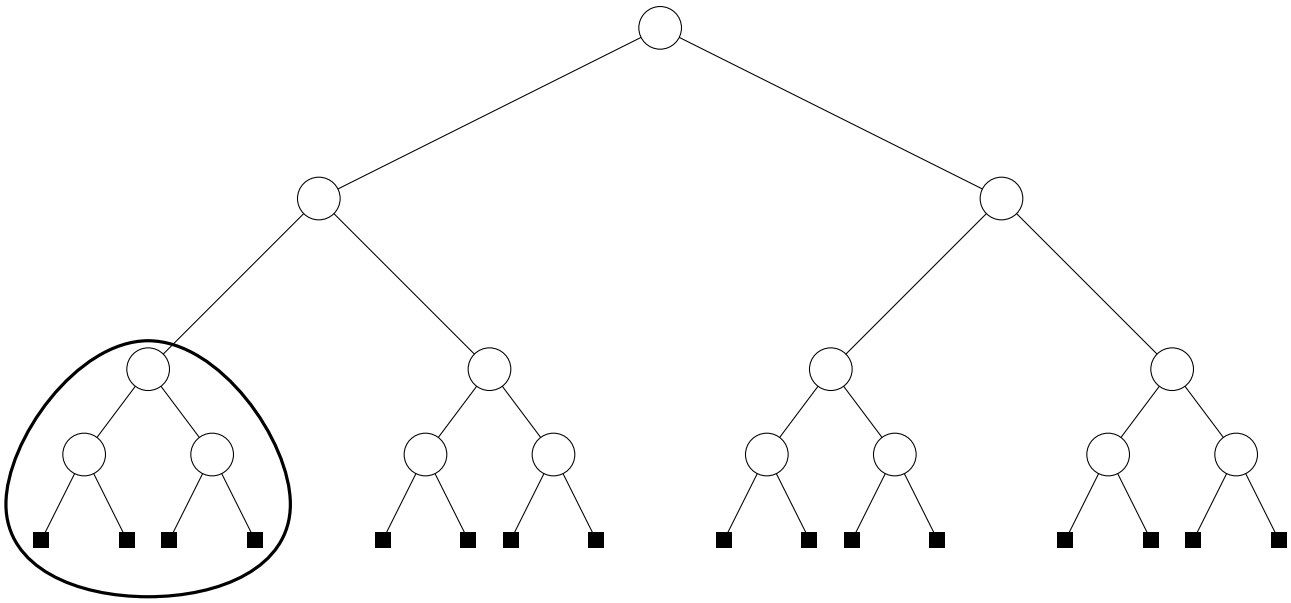

Figure 1: A full binary tree and one of its clusters of size 4.

B Proof of Lemma 5.6

In this section, we provide the proof of Lemma 5.6. We restate the lemma here for convenience.

Lemma 5.6 *Assume that each query response is correct independently with probability $p > \frac{1}{2}$, and adversarially incorrect with probability $1-p$. There exists an adaptive algorithm with the following property: Given a tree $\mathcal{T}$ of diameter D, and error parameter $\delta > 0$, the algorithm finds the target with probability at least $1-\delta$, using at most $O(D + \log(1/\delta))$ vertex queries.*

Our algorithm is a very minor modification of an algorithm of Feige et al. [18]. The algorithm performs a walk on the tree $\mathcal{T}$: in each iteration, a vertex v is queried, and if the response points to a neighbor v', the algorithm next "moves to" v'. However, with every response confirming v as the target, the algorithm moves "deeper" into v (by incrementing a counter): subsequently, responses pointing to neighbors of v will only move the walk one step "shallower" in v, and the walk will only leave v once the counter has been reduced down to 0.

More formally, each vertex v in $\mathcal{T}$ has a counter $c(v)$. All the counters are initially 0, and at any point of the algorithm, at most one counter will have a non-zero (positive) value.

In the i^{th} iteration, the algorithm queries a vertex q_i. The response to the query is either the node q_i itself or one of its neighbors in $\mathcal{T}$. If the query response is q_i, i.e., the noisy response proposes q_i as the target, then the algorithm increments $c(q_i)$ and will next query the same node $q_{i+1} = q_i$. If the query response is a node other than q_i, the algorithm's action depends on $c(q_i)$. If $c(q_i) > 0$, the algorithm decrements $c(q_i)$ and will next query the same node $q_{i+1} = q_i$. Finally, if $c(q_i) = 0$, then q_{i+1} is simply the vertex given in response to the query. The algorithm is given formally as Algorithm 6.

We use $c_i(v)$ to denote the value of the counter $c(v)$ after the i^{th} iteration. Let $d(v, v')$ denote the distance between v and v' in $\mathcal{T}$, and define the *potential value* of v in iteration i as

$$\Phi_i(v) = d(v, q_i) - c_i(v) + \sum_{v' \neq v} c_i(v').$$

Algorithm 6 TreeWalk $(\mathcal{T}, \delta)$

$c(v) \leftarrow 0$ for every vertex v in $\mathcal{T}$.
Let q be an arbitrary vertex in $\mathcal{T}$.
for $i = 1, \ldots \max(\frac{2(D+1)}{2p-1}, \frac{8\ln(1/\delta)}{(2p-1)^2})$ **do**
    Query the node q, and let r be the noisy response.
    if $r = q$ **then**
        $c(q) = c(q) + 1$.
    else if $c(q) > 0$ **then**
        $c(q) = c(q) - 1$.
    else
        $q \leftarrow r$.
return q

The following lemma states that the potential of the (unknown) target node must decrease with each correct response to a query (and may increase with incorrect responses).

Lemma B.1 *Let t be the (unknown) target. For every i, if the response to the i^{th} query is correct, then $\Phi_{i+1}(t) = \Phi_i(t) - 1$; if the response is incorrect, then $\Phi_{i+1}(t) \leq \Phi_i(t) + 1$.*

Proof. We distinguish two cases, based on the node queried in the i^{th} iteration.

- If $q_i = t$, then a correct response increments t's counter, and thus decreases t's potential by 1. An incorrect response either decrements t's counter or — if the counter was 0 — moves the query node to one of t's neighbors. In either case, t's potential is only increased by 1.
- If $q_i \neq t$, then a correct response points towards t. If $c_i(q_i) > 0$, then $c(q_i)$ is decremented; if $c_i(q_i) = 0$, then the new query $q_{i+1} = r_i$ is made to a node one step closer to t. Therefore, t's potential is decreased by 1.

  An incorrect response could increment the counter of q_i (if it points to q_i), decrement the counter of q_i (if it points to an incorrect neighbor of q_i and $c_i(q_i) > 0$), or move the next query $q_{i+1} = r_i$ to an incorrect neighbor of q_i that is one step further away from t. In all three cases, the potential of t increases or decreases by 1; in particular, it at most increases by 1. ■

Proof of Lemma 5.6. Algorithm 6 runs for k iterations. In expectation, a p fraction of the query responses are correct. Let $\epsilon = \frac{2p-1}{4} < p$. Using Hoeffding's inequality [22], with probability at least $1 - e^{-2k\epsilon^2}$, at least $(p-\epsilon)k$ query responses are correct. In that case, by Lemma B.1,

$$\begin{aligned}\Phi_k(t) &\leq \Phi_0(t) - (p-\epsilon)k + (1-p+\epsilon)k \\ &= d(t, q_0) - (2p-1-2\epsilon)k \leq D - \frac{2p-1}{2}k. \end{aligned} \tag{1}$$

Because $k \geq \frac{\ln(1/\delta)}{2\epsilon^2}$, the success probability is at least $1 - e^{-2\epsilon^2 k} \geq 1 - \delta$. And because $k \geq \frac{2(D+1)}{2p-1}$, Inequality 1 implies that $\Phi_k(t) \leq D - (D+1) < 0$. Therefore, by definition of the potential value, $c_k(t) > 0$. The only node that could have a positive counter in any iteration i is q_i; therefore, the returned node must be t in the high-probability case. ■