Flattening a Hierarchical Clustering through Active Learning

Fabio Vitale
Department of Computer Science
INRIA Lille, France &
Sapienza University of Rome, Italy
fabio.vitale@inria.fr

Anand Rajagopalan
Google Research NY
New York, USA
anandbr@google.com

Claudio Gentile
Google Research NY
New York, USA
cgentile@google.com

Abstract

We investigate active learning by pairwise similarity over the leaves of trees originating from hierarchical clustering procedures. In the realizable setting, we provide a full characterization of the number of queries needed to achieve perfect reconstruction of the tree cut. In the non-realizable setting, we rely on known important-sampling procedures to obtain regret and query complexity bounds. Our algorithms come with theoretical guarantees on the statistical error and, more importantly, lend themselves to *linear-time* implementations in the relevant parameters of the problem. We discuss such implementations, prove running time guarantees for them, and present preliminary experiments on real-world datasets showing the compelling practical performance of our algorithms as compared to both passive learning and simple active learning baselines.

1 Introduction

Active learning is a learning scenario where labeled data are scarse and/or expensive to gather, as they require careful assessment by human labelers. This is often the case in several practical settings where machine learning is routinely deployed, from image annotation to document classification, from speech recognition to spam detection, and beyond. In all such cases, an active learning algorithm tries to limit human intervention by seeking as little supervision as possible, still obtaining accurate prediction on unseen samples. This is an attractive learning framework offering substantial practical benefits, but also presenting statistical and algorithmic challenges.

A main argument that makes active learning effective is when combined with methods that exploit the *cluster structure* of data (e.g., [11, 21, 10], and references therein), where a cluster typically encodes some notion of semantic similarity across the involved data points. An ubiquitous solution to clustering is to organize data into a *hierarchy*, delivering clustering solutions at different levels of resolution. An (agglomerative) Hierarchical Clustering (HC) procedure is an unsupervised learning method parametrized by a similarity function over the items to be clustered and a linkage function that lifts similarity from items to clusters of items. Finding the "right" level of resolution amounts to turning a given HC into a *flat* clustering by cutting the resulting tree appropriately. We would like to do so by resorting to human feedback in the form of *pairwise similarity* queries, that is, yes/no questions of the form "are these two products similar to one another ?" or "are these two news items covering similar events ?". It is well known that such queries are relatively easy to respond to, but are also intrinsically prone to subjectiveness and/or noise. More importantly, the hierarchy at hand need not be aligned with the similarity feedback we actually receive.

In this paper, we investigate the problem of cutting a tree originating from a pre-specified HC procedure through pairwise similarity queries generated by active learning algorithms. Since the tree is typically not consistent with the similarity feedback, that is to say, the feedback is *noisy*, we are lead to tackle this problem under a variety of assumptions about the nature of this noise (from noiseless to random but persistent to general agnostic). Moreover, because different linkage functions

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

applied to the very same set of items may give rise to widely different tree topologies, our study also focuses on characterizing active learning performance as a function of the structure of the tree at hand. Finally, because these hierarchies may in practice be sizeable (in the order of billion nodes), scalability will be a major concern in our investigation.

On motivation. It is often the case in big organizations that data processing pipelines are split into *services*, making Machine Learning solution providers be constrained by the existing hardware/software infrastructure. In the Active Learning applications that motivate this work, the hierarchy over the items to be clustered is provided by a third party, i.e., by an exogenous data processing tool that relies on side information on the items (e.g., word2vec mappings and associated distance functions) which are possibly generated by yet another service, etc. In this modular environment, it is reasonable to assume that the tree is *given to us as part of the input* of our Active Learning problem. The human feedback our algorithms rely upon may or may not be consistent with the tree at hand both because human feedback is generally noisy and because this feeback may originate from yet another source of data, e.g., another production team in the organization that was not in charge of building the original tree. In fact, the same tree over the data items may serve the clustering needs of different groups within the organization, having different goals and views on the same data. This also motivates why, when studying this problem, we are led to consider different noise scenarios, that is, the presence of noisy feedback and the possibility that the given tree is not consistent with the clustering corresponding to the received feedbacks.

Our contribution. In the realizable setting (both noiseless and persistent noisy, Section 3), we introduce algorithms whose expected number of queries scale with the *average complexity* of tree cuts, a notion which is introduced in this paper. A distinctive feature of these algorithms is that they are rather ad hoc in the way they deal with the structure of our problem. In particular, they cannot be seen as finding the query that splits the version space as evenly as possible, a common approach in many active learning papers (e.g., [12, 25, 15, 16, 27, 24], and references therein). We then show that, at least in the noiseless case, this average complexity measure characterizes the expected query complexity of the problem. Our ad hoc analyses are beneficial in that they deliver sharper guarantees than those readily available from the above papers. In addition, and perhaps more importantly for practical usage, our algorithms admit *linear-time* implementations in the relevant parameters of the problem (like the number of items to be clustered). In the non-realizable setting (Section 4), we build on known results in importance-weighted active learning (e.g., [5, 6]) to devise a selective sampling algorithm working under more general conditions. While our statistical analysis follows by adaptating available results, our goal here is to rather come up with fast implementations, so as to put the resulting algorithms on the same computational footing as those operating under (noisy) realizability assumptions. By leveraging the specific structure of our hypothesis space, we design a fast incremental algorithm for selective sampling whose running time per round is linear in the height of the tree. In turn, this effort paves the way for our experimental investigation (Section 5), where we compare the effectiveness of the two above-mentioned approaches (realizable with persistent noise vs non-realizable) on real data originating from various linkage functions. Though preliminary in nature, these experiments seem to suggest that in practice the algorithms originating from the persistent noise assumption exhibit more attractive learning curves than those working in the more general non-realizable setting.

Related work. The literature on active learning is vast, and we can hardly do it justice here. In what follows we confine ourselves to the references which we believe are closest to our paper. Since our sample space is discrete (the set of all possible pairs of items from a finite set of size n), our realizable setting is essentially a *pool-based* active learning setting. Several papers have considered greedy algorithms which generalize binary search [1, 20, 12, 25, 15, 24]. The query complexity can be measured either in the worst case or averaged over a prior distribution over all possible labeling functions in a given set. The query complexity of these algorithms can be analyzed by comparing it to the best possible query complexity achieved for that set of items. In [12] it is shown that if the probability mass of the version space is split as evenly as possible then the approximation factor for its average query complexity is $\mathcal{O}(\log(1/p_m))$, where p_m is the minimal prior probability of any considered labeling function. [15] extended this result through a more general approach to approximate greedy rules, but with the worse factor $\mathcal{O}(\log^2(1/p_m))$. [20] observed that modifying the prior distribution always allows one to replace $\mathcal{O}(\log(1/p_m))$ by the smaller factor $\mathcal{O}(\log N)$, where N is the size of the set of labeling functions. Results of a similar flavor are contained in [25, 24]. In our case, N can be exponential in n (see Section 2), making these landmark results too broad to be tight for our specific setting. Furthermore, some of these papers (e.g., [12, 25, 24]) have

only theoretical interest because of their difficult algorithmic implementation. Interesting advances on this front are contained in the more recent paper [27], though when adapted to our specific setting, their results give rise to worse query bounds than ours. In the same vein are the papers by [7, 8], dealing with persistent noise. Finally, in the non-realizable setting, our work fully relies on [6], which in turns builds on standard references like [9, 5, 17] – see, e.g., the comprehensive survey by [18]. Further references, specifically related to clustering with queries, are mentioned in Appendix A.

2 Preliminaries and learning models

We consider the problem of finding cuts of a given binary tree through pairwise similarity queries over its leaves. We are given in input a binary[1] tree T originating from, say, an agglomerative (i.e., bottom-up) HC procedure (single linkage, complete linkage, etc.) applied to a set of items $L = \{x_1, \ldots, x_n\}$. Since T is the result of successive (binary) merging operations from bottom to top, T turns out to be a *strongly binary tree*[2] and the items in L are the leaves of T. We will denote by V the set of nodes in T, including its leaves L, and by r the root of T. The height of T will be denoted by h. When referring to a subtree T' of T, we will use the notation $V(T')$, $L(T')$, $r(T')$, and $h(T')$, respectively. We also denote by $T(i)$ the subtree of T rooted at node i, and by $L(i)$ the set of leaves of $T(i)$, so that $L(i) = L(T(i))$, and $r(T(i)) = i$. Moreover, $\mathrm{par}(i)$ will denote the parent of node i (in tree T), $\mathrm{left}(i)$ will be the left-child of i, and $\mathrm{right}(i)$ its right child.

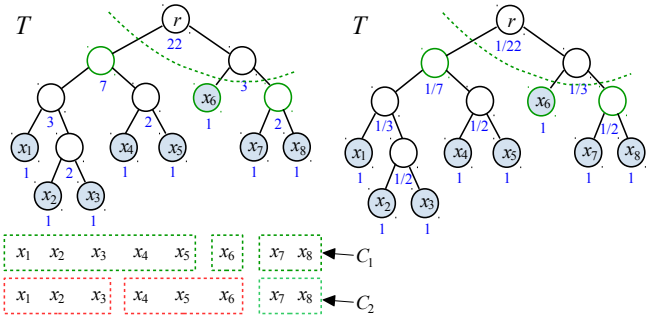

Figure 1: Left: A binary tree corresponding to a hierarchical clustering of the set of items $L = \{x_1, \ldots, x_8\}$. The cut depicted in dashed green has two nodes above and the rest below. This cut induces over L the flat clustering $C_1 = \{\{x_1, x_2, x_3, x_4, x_5\}, \{x_6\}, \{x_7, x_8\}\}$ corresponding to the leaves of the subtrees rooted at the 3 green-bordered nodes just below the cut (the lower boundary of the cut). Clustering C_1 is therefore realized by T. On the contrary, clustering $C_2 = \{\{x_1, x_2, x_3\}, \{x_4, x_5, x_6\}, \{x_7, x_8\}\}$ is not. Close to each node i is also displayed the number $N(i)$ of realized cuts by the subtree rooted at i. For instance, in this figure, $7 = 1 + 3 \cdot 2$, and $22 = 1 + 7 \cdot 3$, so that T admits overall $N(T) = 22$ cuts. **Right:** The same figure, where below each node i are the probabilities $\boldsymbol{p}(i)$ encoding a uniform prior distribution over cuts. Notice that $\boldsymbol{p}(i) = 1/N(i)$ so that, like all other cuts, the depicted green cut has probability $(1 - 1/22) \cdot (1 - 1/3) \cdot (1/7) \cdot 1 \cdot (1/2) = 1/22$.

A *flat* clustering $\mathcal{C}$ of L is a partition of L into disjoint (and non-empty) subsets. A *cut* c of T of size K is a set of K edges of T that partitions V into two disjoint subsets; we call them the nodes *above* c and the nodes *below* c. Cut c also univocally induces a clustering over L, made up of the clusters $L(i_1), L(i_2), \ldots, L(i_K)$, where $i_1, i_2, \ldots, i_K$ are the nodes below c that the edges of c are incident to. We denote this clustering by $\mathcal{C}(c)$, and call the nodes $i_1, i_2, \ldots, i_K$ the *lower boundary* of c. We say that clustering $\mathcal{C}_0$ is *realized* by T if there exists a cut c of T such that $\mathcal{C}(c) = \mathcal{C}_0$. See Figure 1 (left) for a pictorial illustration.

Clearly enough, for a given L, and a given tree T with set of leaves L, not all possible clusterings over L are realized by T, as the number and shape of the clusterings realized by T are strongly influenced by T's structure. Let $N(T)$ be the number of clusterings realized by T (notice that this is also equal to the number of distinct cuts admitted by T). Then $N(T)$ can be computed through a simple recursive formula. If we let $N(i)$ be the number of cuts realized by $T(i)$, one can easily verify that $N(i) = 1 + N(\mathrm{left}(i)) \cdot N(\mathrm{right}(i))$, with $N(x_i) = 1$ for all $x_i \in L$. With this notation, we then have $N(T) = N(r(T))$. If T has n leaves, $N(T)$ ranges from n, when T is a degenerate line tree, to the exponential $\lfloor \alpha^n \rfloor$, when T is the full binary tree, where $\alpha \simeq 1.502$ (e.g., `http://oeis.org/A003095`). See again Figure 1 (left) for a simple example.

[1] In fact, the trees we can handle are more general than binary: we are making the binary assumption throughout for presentational convenience only.

[2] A strongly binary tree is a rooted binary tree for which the root is adjacent to either zero or two nodes, and all non-root nodes are adjacent to either one or three nodes.

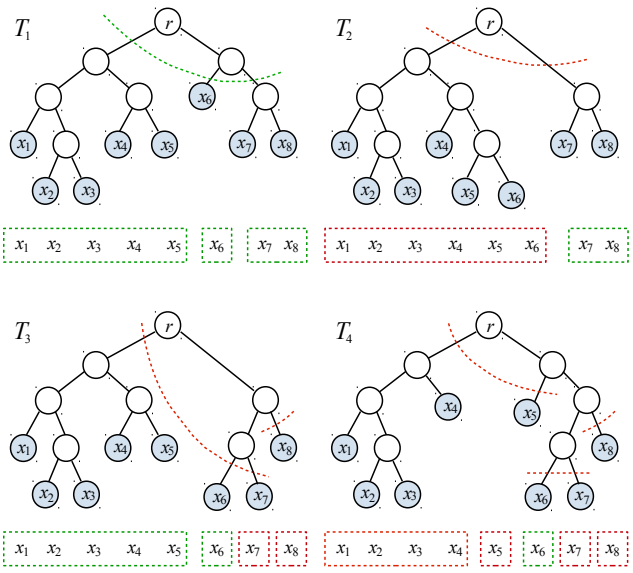

Figure 2: Cuts and corresponding values of $d_H(\Sigma, \widehat{\mathcal{C}})$ in four input trees for the same ground-truth Σ determined by the clustering $\{\{x_1, x_2, x_3, x_4, x_5\}, \{x_6\}, \{x_7, x_8\}\}$. Underneath each tree is the clustering $\widehat{\mathcal{C}}$ induced by the depicted cut. The color of a cluster is green if it corresponds to a cluster in the clustering of Σ, and is red otherwise. In T_1 we have $d_H(\Sigma, \widehat{\mathcal{C}}) = 0$ (realizable case). The other 3 trees illustrate the non-realizable case with the cut minimizing $d_H(\Sigma, \widehat{\mathcal{C}})$ on the corresponding tree. Recalling that $d_H(\Sigma, \widehat{\mathcal{C}})$ counts *ordered* pairs, in T_2 we have $d_H(\Sigma, \widehat{\mathcal{C}}) = 10$, because x_6 now belongs to the first cluster according to clustering $\widehat{\mathcal{C}}$. In T_3 we have $d_H(\Sigma, \widehat{\mathcal{C}}) = 2$. Finally, in T_4 it is easy to verify that $d_H(\Sigma, \widehat{\mathcal{C}}) = 10$.

A *ground-truth* matrix Σ is an $n \times n$ and ± 1-valued symmetric matrix $\Sigma = [\sigma(x_i, x_j)]_{i,j=1}^{n \times n}$ encoding a pairwise similarity relation over L. Specifically, if $\sigma(x_i, x_j) = 1$ we say that x_i and x_j are similar, while if $\sigma(x_i, x_j) = -1$ we say they are dissimilar. Moreover, we always have $\sigma(x_i, x_i) = 1$ for all $x_i \in L$. Notice that Σ need not be consistent with a given clustering over L, i.e., the binary relation defined by Σ over L need not be transitive.

Given T and its leaves L, an active learning algorithm A proceeds in a sequence of rounds. In a purely active setting, at round t, the algorithm queries a pair of items (x_{i_t}, x_{j_t}), and observes the associated label $\sigma(x_{i_t}, x_{j_t})$. In a *selective sampling* setting, at round t, the algorithm is presented with (x_{i_t}, x_{j_t}) drawn from some distribution over $L \times L$, and has to decide whether or not to query the associated label $\sigma(x_{i_t}, x_{j_t})$. In both cases, the algorithm is stopped at some point, and is compelled to commit to a specific cut of T (inducing a flat clustering over L). Coarsely speaking, the goal of A is to come up with a good cut of T, by making as few queries as possible on the entries of Σ.

Noise Models. The simplest possible setting, called *noiseless realizable* setting, is when Σ itself is consistent with a given clustering realized by T, i.e., when there exists a cut c^* of T such that $\mathcal{C}(c^*) = \{L(i_1), \ldots, L(i_K)\}$, for some nodes $i_1, \ldots, i_K \in V$, that satisfies the following: For all $r = 1, \ldots, K$, and for all pairs $(x_i, x_j) \in L(i_r) \times L(i_r)$ we have $\sigma(x_i, x_j) = 1$, while for all other pairs we have $\sigma(x_i, x_j) = -1$. We call (persistent) *noisy realizable* setting one where Σ is generated as follows. Start off from the noiseless ground-truth matrix, and call it Σ^*. Then, in order to obtain Σ from Σ^*, consider the set of all $\binom{n}{2}$ pairs (x_i, x_j) with $i < j$, and pick uniformly at random a subset of size $\lfloor \lambda \binom{n}{2} \rfloor$, for some $\lambda \in [0, 1/2)$. Each such pair has flipped label in Σ: $\sigma(x_i, x_j) = 1 - \sigma^*(x_i, x_j)$. This is then combined with the symmetric $\sigma(x_i, x_j) = \sigma(x_j, x_i)$, and the reflexive $\sigma(x_i, x_i) = 1$ conditions. We call λ the noise level. Notice that this kind of noise is random but *persistent*, in that if we query the same pair (x_i, x_j) twice we do obtain the same answer $\sigma(x_i, x_j)$. Clearly, the special case $\lambda = 0$ corresponds to the noiseless setting. Finally, in the general *non-realizable* (or agnostic) setting, Σ is an arbitrary matrix that need not be consistent with any clustering over L, in particular, with any clustering over L realized by T.

Error Measure. If Σ is some ground-truth matrix over L, and $\widehat{c}$ is the cut output by A, with induced clustering $\widehat{\mathcal{C}} = \mathcal{C}(\widehat{c})$, we let $\Sigma_{\widehat{\mathcal{C}}} = [\sigma_{\widehat{\mathcal{C}}}(x_i, x_j)]_{i,j=1}^{n \times n}$ be the similarity matrix associated with $\widehat{\mathcal{C}}$, i.e., $\sigma_{\widehat{\mathcal{C}}}(x_i, x_j) = 1$ if x_i and x_j belong to the same cluster, and -1 otherwise. Then the *Hamming* distance $d_H(\Sigma, \widehat{\mathcal{C}})$ simply counts the number of (ordered) pairs (x_i, x_j) having inconsistent sign:

$$d_H(\Sigma, \widehat{\mathcal{C}}) = \left| \{(x_i, x_j) \in L^2 \, : \, \sigma(x_i, x_j) \neq \sigma_{\widehat{\mathcal{C}}}(x_i, x_j)\} \right| .$$

The same definition applies in particular to the case when Σ itself represents a clustering over L. The quantity d_H, sometimes called correlation clustering distance, is closely related to the Rand index [26] – see, e.g., [23]. Figure 2 contains illustrative examples.

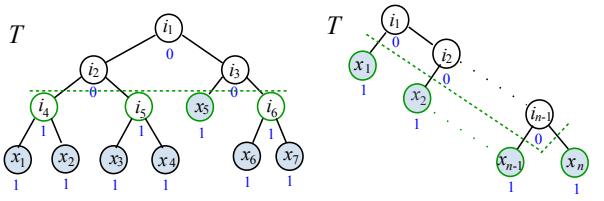

Figure 3: Left: The dotted green cut c^* can be described by the set of values of $\{y(i), i \in V\}$, below each node. In this tree, in order to query, say, node i_2, it suffices to query any of the four pairs (x_1, x_3), (x_1, x_4), (x_2, x_3), or (x_2, x_4). The baseline queries i_1 through i_6 in a breadth-first manner, and then stops having identified c^*. **Right:** This graph has $N(T) = n$. On the depicted cut, the baseline has to query all $n - 1$ internal nodes.

Prior distribution. Recall cut c^* defined in the noiseless realizable setting and its associated Σ^*. Depending on the specific learning model we consider (see below), the algorithm may have access to a *prior* distribution $\mathbb{P}(\cdot)$ over c^*, parametrized as follows. For $i \in V$, let $\boldsymbol{p}(i)$ be the conditional probability that i is below c^* given that all i's ancestors are above. If we denote by $\mathcal{AB}(c^*) \subseteq V$ the nodes of T which are above c^*, and by $\mathcal{LB}(c^*) \subseteq V$ those on the lower boundary of c^*, we can write

$$\mathbb{P}(c^*) = \left(\prod_{i \in \mathcal{AB}(c^*)} (1 - \boldsymbol{p}(i)) \right) \cdot \left(\prod_{j \in \mathcal{LB}(c^*)} \boldsymbol{p}(j) \right) , \tag{1}$$

where $\boldsymbol{p}(i) = 1$ if $i \in L$. In particular, setting $\boldsymbol{p}(i) = 1/N(i)\ \forall i$ yields the *uniform* prior $\mathbb{P}(c^*) = 1/N(T)$ for all c^* realized by T. See Figure 1 (right) for an illustration. A canonical example of a non-uniform prior is one that favors cuts close to the root, which are thereby inducing clusterings having few clusters. These can be obtained, e.g., by setting $\boldsymbol{p}(i) = \alpha$, for some constant $\alpha \in (0, 1)$.

Learning models. We consider two learning settings. The first setting (Section 3) is an active learning setting under a noisy realizability assumption with prior information. Let $\mathcal{C}^* = \mathcal{C}(c^*)$ be the ground truth clustering induced by cut c^* before noise is added. Here, for a given prior $\mathbb{P}(c^*)$, the goal of learning is to identify $\mathcal{C}^*$ either exactly (when $\lambda = 0$) or approximately (when $\lambda > 0$), while bounding the expected number of queries (x_{i_t}, x_{j_t}) made to the ground-truth matrix Σ, the expectation being over the noise, and possibly over $\mathbb{P}(c^*)$. In particular, if $\widehat{\mathcal{C}}$ is the clustering produced by the algorithm after it stops, we would like to prove upper bounds on $\mathbb{E}[d_H(\Sigma^*, \widehat{\mathcal{C}})]$, as related to the number of active learning rounds, as well as to the properties of the prior distribution. The second setting (Section 4) is a selective sampling setting where the pairs (x_{i_t}, x_{j_t}) are drawn i.i.d. according to an arbitrary and unknown distribution $\mathcal{D}$ over the n^2 entries of Σ, and the algorithm at every round can choose whether or not to query the label. After a given number of rounds the algorithm is stopped, and the goal is the typical goal of agnostic learning: no prior distribution over cuts is available anymore, and we would like to bound with high probability over the sample $(x_{i_1}, x_{j_1}), (x_{i_2}, x_{j_2}), \ldots$ the so-called *excess risk* of the clustering $\widehat{\mathcal{C}}$ produced by A, i.e., the difference

$$\mathbb{P}_{(x_i, x_j) \sim \mathcal{D}} \left(\sigma(x_i, x_j) \neq \sigma_{\widehat{\mathcal{C}}}(x_i, x_j) \right) - \min_c \mathbb{P}_{(x_i, x_j) \sim \mathcal{D}} \left(\sigma(x_i, x_j) \neq \sigma_{\mathcal{C}(c)}(x_i, x_j) \right) , \tag{2}$$

the minimum being over all possible cuts c realized by T. Notice that when $\mathcal{D}$ is uniform the excess risk reduces to $\frac{1}{n^2} \left(d_H(\Sigma, \widehat{\mathcal{C}}) - \min_c d_H(\Sigma, \mathcal{C}(c)) \right)$. At the same time, we would like to bound with high probability the total number of labels the algorithm has queried.

3 Active learning in the realizable case

As a warm up, we start by considering the case where $\lambda = 0$ (no noise). The underlying cut c^* can be conveniently described by assigning to each node i of T a binary value $y(i) = 0$ if i is above c^*, and $y(i) = 1$ if i is below. Then we can think of an active learning algorithm as querying *nodes*, instead of querying pairs of leaves. A query to node $i \in V$ can be implemented by querying *any* pair $(x_{i_\ell}, x_{i_r}) \in L(\text{left}(i)) \times L(\text{right}(i))$. When doing so, we actually receive $y(i)$, since for any such (x_{i_ℓ}, x_{i_r}), we clearly have $y(i) = \sigma^*(x_{i_\ell}, x_{i_r})$. An obvious baseline is then to perform a kind of breadth-first search in the tree: We start by querying the root r, and observe $y(r)$; if $y(r) = 1$ we stop and output clustering $\widehat{\mathcal{C}} = \{L\}$; otherwise, we go down by querying both $\text{left}(r)$ and $\text{right}(r)$, and then proceed recursively. It is not hard to show that this simple algorithm will make at most $2K - 1$ queries, with an overall running time of $O(K)$, where K is the number of clusters of $\mathcal{C}(c^*)$. See Figure 3 for an illustration. If we know beforehand that K is very small, then this baseline is a tough competitor. Yet, this is not the best we can do in general. Consider, for instance, the line graph in Figure 3 (right), where c^* has $K = n$.

Ideally, for a given prior $\mathbb{P}(\cdot)$, we would like to obtain a query complexity of the form $\log(1/\mathbb{P}(c^*))$, holding in the worst-case for all underlying c^*. As we shall see momentarily, this is easily obtained when $\mathbb{P}(\cdot)$ is uniform. We first describe a version space algorithm (One Third Splitting, OTS) that admits a fast implementation, and whose number of queries in the worst-case is $\mathcal{O}(\log N(T))$. This will in turn pave the way for our second algorithm, Weighted Dichotomic Path (WDP). WDP leverages $\mathbb{P}(\cdot)$, but its theoretical guarantees only hold *in expectation* over $\mathbb{P}(c^*)$. WDP will then be extended to the persistent noisy setting through its variant Noisy Weighted Dichotomic Path (N-WDP).

We need a few ancillary definitions. First of all note that, in the noiseless setting, we have a clear hierarchical structure on the labels $y(i)$ of the internal nodes of T: Whenever a query reveals a label $y(i) = 0$, we know that all i's ancestors will have label 0. On the other hand, if we observe $y(i) = 1$ we know that all internal nodes of subtree $T(i)$ have label 1. Hence, disclosing the label of some node indirectly entails disclosing the labels of either its ancestors or its descendants. Given T, a bottom-up path is any path connecting a node with one of its ancestors in T. In particular, we call a *backbone* path any bottom up path having maximal length. Given $i \in V$, we denote by $S_t(i)$ the *version space* at time t associated with $T(i)$, i.e., the set of all cuts of $T(i)$ that are consistent with the labels revealed so far. For any node $j \neq i$, $S_t(i)$ splits into $S_t^{y(j)=0}(i)$ and $S_t^{y(j)=1}(i)$, the subsets of $S_t(i)$ obtained by imposing a further constraint on $y(j)$.

OTS **(One Third Splitting):** For all $i \in V$, OTS maintains over time the value $|S_t(i)|$, i.e., the size of $S_t(i)$, along with the forest F made up of all maximal subtrees T' of T such that $|V(T')| > 1$ and for which none of their node labels have been revealed so far. OTS initializes F to contain T only, and maintains F updated over time, by picking any backbone of any subtree $T' \in F$, and visiting it in a bottom-up manner. See the details in Appendix B.1. The following theorem (proof in Appendix B.1) crucially relies on the fact that π is a backbone path of T', rather than an arbitrary path.

Theorem 1 *On a tree T with n leaves, height h, and number of cuts N,* OTS *finds c^* by making $\mathcal{O}(\log N)$ queries. Moreover, an ad hoc data-structure exists that makes the overall running time $\mathcal{O}(n + h \log N)$ and the space complexity $\mathcal{O}(n)$.*

Hence, Theorem 1 ensures that, for all c^*, a time-efficient active learning algorithm exists whose number of queries is of the form $\log(1/\mathbb{P}(c^*))$, provided $\mathbb{P}(c^*) = 1/N(T)$ for all c^*. This query bound is fully in line with well-known results on splittable version spaces [12, 25, 24], so we cannot make claims of originality. Yet, what is relevant here is that this splitting can be done *very efficiently*.

We complement the above result with a *lower* bound holding in expectation over prior distributions on c^*. This lower bound depends in a detailed way on the structure of T. Given tree T, with set of leaves L, and cut c^*, recall the definitions of $\mathcal{AB}(c^*)$ and $\mathcal{LB}(c^*)$ we gave in Section 2. Let T'_{c^*} be the subtree of T whose nodes are $(\mathcal{AB}(c^*) \cup \mathcal{LB}(c^*)) \setminus L$, and then let $\widetilde{K}(T, c^*) = |L(T'_{c^*})|$ be the number of its leaves. For instance, in Figure 3 (left), T'_{c^*} is made up of the six nodes $i_1, \ldots, i_6$, so that $\widetilde{K}(T, c^*) = 3$, while in Figure 3 (right), T'_{c^*} has nodes $i_1, \ldots, i_{n-1}$, hence $\widetilde{K}(T, c^*) = 1$. Notice that we always have $\widetilde{K}(T, c^*) \leq K$, but for many trees T, $\widetilde{K}(T, c^*)$ may be much smaller than K. A striking example is again provided by the cut in Figure 3 (right), where $\widetilde{K}(T, c^*) = 1$, but $K = n$. It is also helpful to introduce $L_{\mathrm{s}}(T)$, the set of all pairs of *sibling leaves* in T. For instance, in the tree of Figure 3, we have $|L_{\mathrm{s}}(T)| = 3$. One can easily verify that, for all T we have

$$\max_{c^*} \widetilde{K}(T, c^*) = |L_{\mathrm{s}}(T)| \leq \log_2 N(T) \, .$$

We now show that there always exist families of prior distributions $\mathbb{P}(\cdot)$ such that the expected number of queries needed to find c^* is $\Omega(\mathbb{E}[\widetilde{K}(T, c^*)])$. The quantity $\mathbb{E}[\widetilde{K}(T, c^*)]$ is our notion of *average (query) complexity*. Since the lower bound holds in expectation, it also holds in the worst case. The proof can be found in Appendix B.2.

Theorem 2 *In the noiseless realizable setting, for any tree T, any positive integer $B \leq |L_{\mathrm{s}}(T)|$, and any (possibly randomized) active learning algorithm A, there exists a prior distribution $\mathbb{P}(\cdot)$ over c^* such that the* expected *(over $\mathbb{P}(\cdot)$ and A's internal randomization) number of queries A has to make in order to recover c^* is lower bounded by $B/2$, while $B \leq \mathbb{E}[\widetilde{K}(T, c^*)] \leq 2B$, the latter expectation being over $\mathbb{P}(\cdot)$.*

Next, we describe an algorithm that, unlike OTS, is indeed able to take advantage of the prior distribution, but it does so at the price of bounding the number of queries only *in expectation*.

WDP **(Weighted Dichotomic Path)**: Recall prior distibution (1), collectively encoded through the values $\{\boldsymbol{p}(i), i \in V\}$. As for OTS, we denote by F the forest made up of all maximal subtrees T'

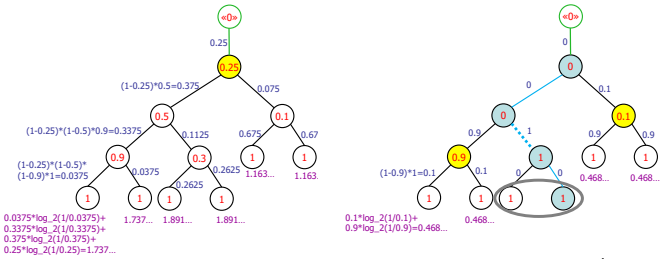

Figure 4: An example of input tree T before (left) and after (right) the first binary search of WDP. The green node is a dummy super-root. The nodes in yellow are the roots of the subtrees currently included in forest F. The numbers in red within each node i indicate the probabilities $\boldsymbol{p}(i)$, while the $q(i)$ values are in blue, and viewed here as associated with edges $(\mathrm{par}(i), i)$. The magenta numbers at each leaf ℓ give the entropy $H(\pi(\ell, r(T')))$, where $r(T')$ is the root of the subtree in F that contains both ℓ and $r(T')$. **Left:** The input tree T at time $t = 0$. No labels are revealed, and no clusters of $\mathcal{C}(c^*)$ are found. **Right:** Tree T after a full binary search has been performed on the depicted light blue path. Before this binary search, that path connected a leaf of a subtree in F to its root (in this case, F contains only T). The selected path is the one maximazing entropy within the forest/tree on the left. The dashed line indicates the edge of c^* found by the binary search. The red, blue and magenta numbers are updated accordingly to the result of the binary search. The leaves enclosed in the grey ellipse are now known to form a cluster of $\mathcal{C}(c^*)$.

of T such that $|V(T')| > 1$ and for which none of their node labels have so far been revealed. F is updated over time, and initially contains only T. We denote by $\pi(u, v)$ a bottom-up path in T having as terminal nodes u and v (hence v is an ancestor of u in T). For a given cut c^*, and associated labels $\{y(i), i \in V\}$, any tree $T' \in F$, and any node $i \in V(T')$, we define[3]

$$q(i) = \mathbb{P}(y(i) = 1 \wedge y(\mathrm{par}(i)) = 0) = \boldsymbol{p}(i) \cdot \prod_{j \in \pi(\mathrm{par}(i), r(T'))} (1 - \boldsymbol{p}(j)) \,. \tag{3}$$

We then associate with any backbone path of the form $\pi(\ell, r(T'))$, where $\ell \in L(T')$, an entropy $H(\pi(\ell, r(T'))) = -\sum_{i \in \pi(\ell, r(T'))} q(i) \log_2 q(i)$. Notice that at the beginning we have $\sum_{i \in \pi(\ell, r(T))} q(i) = 1$ for all $\ell \in L$. This invariant will be maintained on all subtrees T'. The prior probabilities $\boldsymbol{p}(i)$ will evolve during the algorithm's functioning into posterior probabilities based on the information revealed by the labels. Accordingly, also the related values $q(i)$ w.r.t. which the entropy $H(\cdot)$ is calculated will change over time.

Due to space limitations, WDP's pseudocode is given in Appendix B.3, but we have included an example of its execution in Figure 4. At each round, WDP finds the path whose entropy is maximized over all bottom-up paths $\pi(\ell, r')$, with $\ell \in L$ and $r' = r(T')$, where T' is the subtree in F containing ℓ. WDP performs a binary search on such $\pi(\ell, r')$ to find the edge of T' which is cut by c^*, taking into account the current values of $q(i)$ over that path. Once a binary search terminates, WDP updates F and the probabilities $\boldsymbol{p}(i)$ at all nodes i in the subtrees of F. See Figure 4 for an example. Notice that the $\boldsymbol{p}(i)$ on the selected path become either 0 (if above the edge cut by c^*) or 1 (if below). In turn, this causes updates on all probabilities $q(i)$. WDP continues with the next binary search on the next path with maximum entropy at the current stage, discovering another edge cut by c^*, and so on, until F becomes empty. Denote by $\mathcal{P}_{>0}$ the set of all priors $\mathbb{P}(\cdot)$ such that for all cuts c of T we have $\mathbb{P}(c) > 0$. The proof of the following theorem is given in Appendix B.3.

Theorem 3 *In the noiseless realizable setting, for any tree T of height h, any prior distribution $\mathbb{P}(\cdot)$ over c^*, such that $\mathbb{P}(\cdot) \in \mathcal{P}_{>0}$, the expected number of queries made by* WDP *to find c^* is $\mathcal{O}\left(\mathbb{E}\left[\widetilde{K}(T, c^*)\right] \log h\right)$, the expectations being over $\mathbb{P}(\cdot)$.*

For instance, in the line graph of Figure 3 (right), the expected number of queries is $\mathcal{O}(\log n)$ for any prior $\mathbb{P}(\cdot)$, while if T is a complete binary tree with n leaves, and we know that $\mathcal{C}(c^*)$ has $\mathcal{O}(K)$ clusters, we can set $\boldsymbol{p}(i)$ in (1) as $\boldsymbol{p}(i) = 1/\log K$, which would guarantee $\mathbb{E}[\widetilde{K}(T, c^*)] = \mathcal{O}(K)$, and a bound on the expected number of queries of the form $\mathcal{O}(K \log \log n)$. By comparison, observe that the results in [12, 15, 27] would give a query complexity which is at best $\mathcal{O}(K \log^2 n)$, while those in [25, 24] yield at best $\mathcal{O}(K \log n)$. In addition, we show below (Remark 1) that our algorithm has very compelling running time guarantees.

It is often the case that a linkage function generating T also tags each internal node i with a *coherence level* α_i of $T(i)$, which is typically increasing as we move downwards from root to leaves. A common

[3] For definiteness, we set $y(\mathrm{par}(r)) = 0$, that is, we are treating the parent of $r(T)$ as a "dummy super–root" with labeled 0 since time $t = 0$. Thus, according to this definition, $q(r) = \boldsymbol{p}(r)$.

situation in hierarchical clustering is then to figure out the "right" level of granularity of the flat clustering we search for by defining parallel *bands* of nodes of similar coherence where c^* is possibly located. For such cases, a slightly more involved guarantee for WDP is contained in Theorem 6 in Appendix B.3, where the query complexity depends in a more detailed way on the interplay between T and the prior $\mathbb{P}(\cdot)$. In the above example, if we have b-many edge-disjoint bands, Theorem 6 replaces factor $\log h$ of Theorem 3 by $\log b$.

N-WDP (**Noisy Weighted Dichotomic Path**): This is a robust variant of WDP that copes with persistent noise. Whenever a label $y(i)$ is requested, N-WDP determines its value by a majority vote over randomly selected pairs from $L(\text{left}(i)) \times L(\text{right}(i))$. Due to space limitations, all details are contained in Appendix B.4. The next theorem quantifies N-WDP's performance in terms of a tradeoff between the expected number of queries and the distance to the noiseless ground-truth matrix Σ^*.

Theorem 4 *In the noisy realizable setting, given any input tree T of height h, any cut $c^* \sim \mathbb{P}(\cdot) \in \mathcal{P}_{>0}$, and any $\delta \in (0, 1/2)$,* N-WDP *outputs with probability $\geq 1-\delta$ (over the noise in the labels) a clustering $\widehat{\mathcal{C}}$ such that $\frac{1}{n^2}\, d_H(\Sigma^*, \widehat{\mathcal{C}}) = \mathcal{O}\left(\frac{1}{n}\,\frac{(\log(n/\delta))^{3/2}}{(1-2\lambda)^3}\right)$ by asking $\mathcal{O}\left(\frac{\log(n/\delta)}{(1-2\lambda)^2}\,\mathbb{E}\widetilde{K}(T, c^*)\log h\right)$ queries in expectation (over $\mathbb{P}(\cdot)$).*

Remark 1 *Compared to the query bound in Theorem 3, the one in Theorem 4 adds a factor due to noise. The very same extra factor is contained in the bound of [22]. Regarding the running time of* WDP*, the version we have described can be naively implemented to run in $\mathcal{O}(n\,\mathbb{E}\widetilde{K}(T, c^*))$ expected time overall. A more time-efficient variant of* WDP *exists for which Theorem 3 and Theorem 6 still hold, that requires $\mathcal{O}(n + h\,\mathbb{E}\widetilde{K}(T, c^*))$ expected time. Likewise, an efficient variant of* N-WDP *exists for which Theorem 4 holds, that takes $\mathcal{O}\left(n + \left(h + \frac{\log^2 n}{(1-2\lambda)^2}\right)\mathbb{E}\widetilde{K}(T, c^*)\right)$ expected time.*

4 Selective sampling in the non-realizable case

In the non-realizable case, we adapt to our clustering scenario the importance-weighted algorithm in [6]. The algorithm is a selective sampler that proceeds in a sequence of rounds $t = 1, 2, \ldots$. In round t a pair (x_{i_t}, x_{j_t}) is drawn at random from distribution $\mathcal{D}$ over the entries of a given ground truth matrix Σ, and the algorithm produces in response a probability value $p_t = p_t(x_{i_t}, x_{j_t})$. A Bernoulli variable $Q_t \in \{0, 1\}$ is then generated with $\mathbb{P}(Q_t = 1) = p_t$, and if $Q_t = 1$ the label $\sigma_t = \sigma(x_{i_t}, x_{j_t})$ is queried, and the algorithm updates its internal state; otherwise, we skip to the next round. The way p_t is generated is described as follows. Given tree T, the algorithm maintains at each round t an importance-weighted empirical risk minimizer cut $\hat{c}_t$, defined as $\hat{c}_t = \operatorname{argmin}_c \text{err}_{t-1}(\mathcal{C}(c))$, where the "argmin" is over all cuts c realized by T, and $\text{err}_{t-1}(\mathcal{C}) = \frac{1}{t-1}\sum_{s=1}^{t-1}\frac{Q_s}{p_s}\left\{\sigma_{\mathcal{C}}(x_{i_s}, x_{j_s}) \neq \sigma_s\right\}$, being $\{\cdot\}$ the indicator function of the predicate at argument. This is paired up with a *perturbed* empirical risk minimizer

$$\hat{c}'_t = \operatorname*{argmin}_{c\,:\,\sigma_{\mathcal{C}(c)}(x_{i_t}, x_{j_t}) \neq \sigma_{\mathcal{C}(\hat{c}_t)}(x_{i_t}, x_{j_t})} \text{err}_{t-1}(\mathcal{C}(c))\ ,$$

the "argmin" being over all cuts c realized by T that disagree with $\hat{c}_t$ on the current pair (x_{i_t}, x_{j_t}). The value of p_t is a function of $d_t = \text{err}_{t-1}(\mathcal{C}(\hat{c}'_t)) - \text{err}_{t-1}(\mathcal{C}(\hat{c}_t))$, of the form

$$p_t = \min\left\{1, \mathcal{O}\left(1/d_t^2 + 1/d_t\right)\log((N(T)/\delta)\log t)/t\right\}\ , \tag{4}$$

where $N(T)$ is the total number of cuts realized by T (i.e., the size of our comparison class), and δ is the desired confidence parameter. Once stopped, say in round t_0, the algorithm gives in output cut $\hat{c}_{t_0+1}$, and the associated clustering $\mathcal{C}(\hat{c}_{t_0+1})$. Let us call the resulting algorithm NR (Non-Realizable).

Despite $N(T)$ can be exponential in n, there are very efficient ways of computing $\hat{c}_t$, $\hat{c}'_t$, and hence p_t at each round. In particular, an ad hoc procedure exists that incrementally computes these quantities by leveraging the sequential nature of NR. For a given T, and constant $K \geq 1$, consider the class $\mathbb{C}(T, K)$ of cuts inducing clusterings with at most K clusters. Set $R^* = R^*(T, \mathcal{D}) = \min_{c\in\mathbb{C}(T,K)} \mathbb{P}_{(x_i,x_j)\sim\mathcal{D}}\left(\sigma(x_i, x_j) \neq \sigma_{\mathcal{C}(c)}(x_i, x_j)\right)$, and $B_\delta(K, n) = K\log n + \log(1/\delta)$. The following theorem is an adaptation of a result in [6]. See Appendix C.1 for a proof.

Theorem 5 *Let T have n leaves and height h. Given confidence parameter δ, for any $t \geq 1$, with probability at least $1-\delta$, the excess risk (2) achieved by the clustering $\mathcal{C}(\hat{c}_{t+1})$ computed by* NR *w.r.t. the best cut in class $\mathbb{C}(T, K)$ is bounded by $\mathcal{O}\left(\sqrt{\frac{B_\delta(K,n)\log t}{t}} + \frac{B_\delta(K,n)\log t}{t}\right)$, while the (expected)*

Tree	Avg depth	Std. dev	BEST's error	BEST's K
SING	2950	1413.6	8.26%	4679
MED	186.4	41.8	8,51%	1603
COMP	17.1	3.3	8.81%	557

Table 1: Statistics of the trees used in our experiments. These trees result from applying the linkage functions SING, COMP, and MED to the MNIST dataset (first 10000 samples). Each tree has the same set of $n = 10000$ leaves. "Avg depth" is the average depth of the leaves in the tree, "Std. dev" is its standard deviation. For reference, we report the performance of BEST (i.e., the minimizer of d_H over all possible cuts realized by the trees), along with the associated number of clusters K.

number of labels $\sum_{s=1}^{t} p_s$ *is bounded by* $\mathcal{O}\left(\theta\left(R^* t + \sqrt{t\, B_\delta(K,n)\log t} + B_\delta(K,n)\log^3 t\right)\right)$, *where* $\theta = \theta(\mathbb{C}(T,K),\mathcal{D})$ *is the disagreement coefficient of* $\mathbb{C}(T,K)$ *w.r.t. distribution* $\mathcal{D}$. *In particular, when* $\mathcal{D}$ *is uniform we have* $\theta \leq K$. *Moreover, there exists a fast implementation of* NR *whose expected running time per round is* $\mathbb{E}_{(x_i,x_j)\sim\mathcal{D}}[\mathrm{de}(\mathrm{lca}(x_i,x_j))] \leq h$, *where* $\mathrm{de}(\mathrm{lca}(x_i,x_j))$ *is the depth in* T *of the lowest common ancestor of* x_i *and* x_j.

5 Preliminary experiments

The goal of these experiments was to contrast active learning methods originating from the persistent noisy setting (specifically, N-WDP) to those originating from the non-realizable setting (specifically, NR). The comparison is carried out on the hierarchies produced by standard HC methods operating on the first $n = 10000$ datapoints in the well-known MNIST dataset from `http://yann.lecun.com/exdb/mnist/`, yielding a sample space of 10^8 pairs. We used Euclidean distance combined with the single linkage (SING), median linkage (MED), and complete linkage (COMP) functions. The $n \times n$ ground-truth matrix Σ is provided by the 10 class labels of MNIST.

We compared N-WDP with uniform prior and NR to two baselines: passive learning based on empirical risk minimization (ERM), and the active learning baseline performing breadth-first search from the root (BF, Section 3) made robust to noise as in N-WDP. For reference, we also computed for each of the three hierarchies the performance of the best cut in hindsight (BEST) on the *entire* matrix Σ. That is essentially the best one can hope for in each of the three cases. All algorithms except ERM are randomized and have a single parameter to tune. We let such parameters vary across suitable ranges and, for each algorithm, picked the best performing value on a validation set of 500 labeled pairs.

In Table 1, we have collected relevant statistics about the three hierarchies. In particular, the single linkage tree turned out to be very deep, while the complete linkage one is quite balanced. We evaluated test set accuracy vs. number of queries after parameter tuning, excluding these 500 pairs. For N-WDP, once a target number of queries was reached, we computed as current output the maximum-a-posteriori cut. In order to reduce variance, we repeated each experiment 10 times.

The details of our empirical comparison are contained in Appendix C.3. Though our experiments are quite preliminary, some trends can be readily spotted (see Table 2 in Appendix C.3): i. N-WDP significantly outperforms NR. E.g., in COMP at 250 queries, the test set accuracy of N-WDP is at 9.52%, while NR is at 10.1%. A similar performance gap at low number of queries one can observe in SING and MED. This trend was expected: NR is very conservative, as it has been designed to work under more general conditions than N-WDP. We conjecture that, whenever the specific task at hand allows one to make an aggressive noise-free algorithm (like WDP) robust to persistent noise (like N-WDP), this outcome is quite likely to occur. ii. BF is competitive only when BEST has few clusters. iii. N-WDP clearly outperforms ERM, while the comparison between NR and ERM yields mixed results.

Conclusions and ongoing activity. Beyond presenting new algorithms and analyses for pairwise similarity-based active learning, our goal was to put different approaches to active learning on the same footing for comparison on real data. The initial evidence emerging from our experiments is that Active Learning algorithms based on persistent noise can in practice be more effective than those making the more general non-realizable assumption. Notice that the similarities of the pairs of items have been generated by the MNIST class labels, hence they have virtually nothing to do with the trees we generated, which in turn do not rely on those labels at all. These initial trends suggested by our experiments clearly need a more thorough investigation. We are currently using other datasets, of different nature and size. Further HC methods are also under consideration, like those based on k-means.

Acknowledgments

Fabio Vitale acknowledges support from the Google Focused Award "ALL4AI" and the ERC Starting Grant "DMAP 680153", awarded to the Department of Computer Science of Sapienza University.

References

[1] E. Arkin, H. Meijer, J. Mitchell, D. Rappaport, and S. Skiena. Decision trees for geometric models. In *Proc. Symposium on Computational Geometry*, pages 369–378, 1993.

[2] H. Ashtiani, S. Kushagra, and S. Ben-David. Clustering with same-cluster queries. In *Proc. 30th NIPS*, 2016.

[3] P. Awasthi, M. F. Balcan, and K. Voevodski. Local algorithms for interactive clustering. *Journal of Machine Learning Research*, 18, 2017.

[4] M. F. Balcan and A. Blum. Clustering with interactive feedback. In *Proc. of the 19th International Conference on Algorithmic Learning Theory*, pages 316–328, 2008.

[5] Alina Beygelzimer, Sanjoy Dasgupta, and John Langford. Importance weighted active learning. In *Proc. ICML*, pages 49–56. ACM, 2009.

[6] Alina Beygelzimer, Daniel Hsu, John Langford, and Tong Zhang. Agnostic active learning without constraints. In *Proc. 23rd International Conference on Neural Information Processing Systems*, NIPS'10, pages 199–207, 2010.

[7] Yuxin Chen, S. Hamed Hassani, Amin Karbasi, and Andreas Krause. Sequential information maximization: When is greedy near-optimal? In *Proc. 28th Conference on Learning Theory, PMLR 40*, pages 338–363, 2015.

[8] Yuxin Chen, S. Hamed Hassani, and Andreas Krause. Near-optimal bayesian active learning with correlated and noisy tests. In *Proc. 20th International Conference on Artificial Intelligence and Statistics*, 2017.

[9] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15:201–221, 1994.

[10] C. Cortes, G. DeSalvo, C. Gentile, M. Mohri, and N. Zhang. Region-based active learning. In *Proc. 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

[11] S. Dasgupta and D. Hsu. Hierarchical sampling for active learning. In *Proc. of the 25th International Conference on Machine Learning*, 2008.

[12] Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. In *Advances in neural information processing systems*, pages 235–242, 2005.

[13] S. Davidson, S. Khanna, T. Milo, and S. Roy. Top-k and clustering with noisy comparisons. *ACM Trans. Database Syst.*, 39(4):35:1–35:39, 2014.

[14] Donatella Firmani, Barna Saha, and Divesh Srivastava. Online entity resolution using an oracle. *Proc. VLDB Endow.*, 9(5):384–395, 2016.

[15] Daniel Golovin and Andreas Krause. Adaptive submodularity: A new approach to active learning and stochastic optimization. In *arXiv:1003.3967*, 2017.

[16] Alon Gonen, Sivan Sabato, and Shai Shalev-Shwartz. Efficient active learning of halfspaces: An aggressive approach. *Journal of Machine Learning Research*, 14:2583–2615, 2013.

[17] S. Hanneke. A bound on the label complexity of agnostic active learning. In *Proc. 24th International Conference on Machine Learning*, pages 353–360, 2007.

[18] S. Hanneke. Theory of disagreement-based active learning. *Foundations and Trends in Machine Learning*, 7(2-3):131–309, 2014.

[19] Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[20] S. Kosaraju, T. Przytycka, and R. Borgstrom. On an optimal split tree problem. In *Proc. 6th International Workshop on Algorithms and Data Structures*, pages 157–168, 1999.

[21] S. Kpotufe, R. Urner, and S. Ben-David. Hierarchical label queries with data-dependent partitions. In *Proc. 28th Conference on Learning Theory*, pages 1176–1189, 2015.

[22] A. Mazumdar and B. Saha. Clustering with noisy queries. In *arXiv:1706.07510v1*, 2017b.

[23] M. Meila. Local equivalences of distances between clusterings—a geometric perspective. *Machine Learning*, 86(3):369–389, 2012.

[24] Stephen Mussmann and Percy Liang. Generalized binary search for split-neighborly problems. In *Proc. 21st International Conference on Artificial Intelligence and Statistics (AISTATS) 2018*, 2018.

[25] Robert D. Nowak. The geometry of generalized binary search. *IEEE Transactions on Information Theory*, 57(12):7893–7906, 2011.

[26] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.

[27] C. Tosh and S. Dasgupta. Diameter-based active learning. In *Thirty-fourth International Conference on Machine Learning (ICML)*, 2017.

[28] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. Crowdsourcing algorithms for entity resolution. *Proc. VLDB Endow.*, 7(12):1071–1082, 2014.

[29] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.

A Missing material from Section 1

A.1 Further related work

Further papers related to our work are those dealing with clustering with queries, e.g., [13, 2, 22, 4, 3]. In [13] the authors show that $\mathcal{O}(Kn)$ similarity queries are both necessary and sufficient to achieve exact reconstruction of an *arbitrary* clustering with K clusters on n items. This is generalized by [22] where persistent random noise is added. [2] assume the feedback is center-based with a margin condition on top. Because we are constrained to a clustering produced by cutting a given tree, the results in [13, 2, 22] are incomparable to ours, due to the different assumptions. In [4, 3]) the authors consider clusterings realized by a given comparison class (as we do here). Yet, the queries they are allowing are different from ours, hence their results are again incomparable to ours.

Further relevant papers include [29, 28, 14], though these papers are not readily comparable to ours, since their noise assumptions are slightly different, see, e.g., the uneven noise gap analysis in [14] (Theorem 1 and 2 and Propositions 3 and 5 therein). Moreover, being restricted to a clustering realized by T, we generally need less queries. For instance, combining our Theorems 2 and 3, our bound in the realizable case is between $\mathbb{E}[K]$ and $\mathbb{E}[K]\log h$, while in general it takes $n\mathbb{E}[K]$ queries to fully reconstruct the clustering [13]. Also recall the lower bound $n - K + \binom{K}{2}$ in [29] and [14].

B Missing material from Section 3

B.1 One Third Splitting (OTS)

For all $i \in V$, OTS maintain over time the value $|S_t(i)|$, i.e., the size of $S_t(i)$, along with the forest F made up of all maximal subtrees T' of T such that $|V(T')| > 1$ and for which none of their node labels have been revealed so far. Notice that we will not distinguish between labels $y(i)$ revealed directly by a query or indirectly by the hierarchical structure. By maximal here we mean that it is not possible to extend any such subtrees by adding a node of V whose label has not already been revealed. OTS initializes F (when no labels are revealed) to contain T only, and maintains F updated over time. Let subtree $T' \in F$ be arbitrarily chosen, and π be any backbone path of T'. At time t, OTS visits π in a bottom-up manner, and finds the *lowest* node i_t^* in this path satisfying $\left|S_t^{y(i_t^*)=1}(r)\right| \le 2\left|S_t^{y(i_t^*)=0}(r)\right|$, i.e., $\left|S_t^{y(i_t^*)=1}(r)\right| \le \frac{2}{3}|S_t(r)|$, then query node i_t^*. We repeat the above procedure until $|S_t(r)| = 1$, i.e., until we find c^*.

The next lemma is key to showing the logarithmic number of queries made by OTS.

Lemma 1 *With the notation introduced in Section 3, at each time t, the query $y(i_t^*)$ made by* OTS *splits the version space $S_t(r)$ in such a way that*[4]

$$\min\left\{\left|S_t^{y(i_t^*)=0}(r)\right|, \left|S_t^{y(i_t^*)=1}(r)\right|\right\} \ge \frac{|S_t(r)|}{3} .$$

Proof. At each time t, $|S_t(r)|$ is the product of the cardinality of $S_t(\widetilde{r})$ over all roots $\widetilde{r}$ of the trees currently contained in F. Let π be a backbone of one such tree, say tree T', with root r'. Since T' is arbitrary, in order to prove the statement, it is sufficient to show that

$$\min\left\{\left|S_t^{y(i_t^*)=0}(r')\right|, \left|S_t^{y(i_t^*)=1}(r')\right|\right\} \ge \frac{|S_t(r')|}{3} .$$

Let $h(\pi)$ be the length of π, i.e., the number of its edges, and $\langle j_0, j_1, \ldots, j_{h(\pi)}\rangle$ be the sequence of its nodes, from bottom to top. For any $k < h(\pi)$, we denote by j_k^{s} the sibling of j_k in T' (hence, by this definition j_k^{s} does *not* belong to π). Now, observe that the number of possible labelings of π is equal to $h(\pi)+1$, that is, each labeling of π corresponds to an integer $z \in \{0, 1, \ldots, h(\pi)\}$ such that $y(j_k) = 1$ for all $k \le z$ and $y(j_k) = 0$ for all $z < k \le h(\pi)$. Then, given any labeling of the nodes of π (represented by the above z), we have

$$|S_t(r')| = \begin{cases} \prod_{k=z}^{h(\pi)-1} |S_t(j_k^{\mathrm{s}})| & \text{if } z < h(\pi) , \\ 1 & \text{if } z = h(\pi) . \end{cases}$$

[4] This bound is indeed tight for this strategy when the input is a full binary tree of height 3.

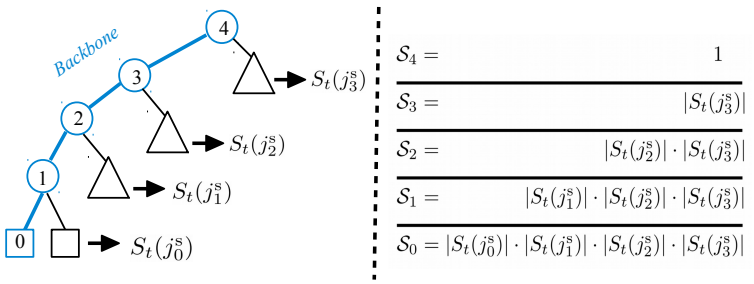

Figure 5: A backbone path π selected by OTS and the associated quantities needed to prove the main properties of the selected node i_t^*. Leaves are represented by squares, subtrees by triangles. For simplicity, in this picture π is starting from the leftmost leaf of T', but it can clearly be chosen to start from any of its deepest leaves. The sum of all terms on the right of each subtree equals $|S_t(r')|$. The cornerstone of the proof is that i_t^* (and hence z^*) corresponds to the lowest among the $h(\pi) = 4$ horizontal lines depicted in this figure for which the sum of all products below the chosen line is at least half the sum of all the products above the line. Furthermore, the fact that $|S_t(j_0^s)| = 1$ guarantees that $\mathcal{S}_0 = \mathcal{S}_1$. Combined with the fact that $\mathcal{S}_z \geq \mathcal{S}_{z+1}$ for all $z \in \{0, \ldots, h(\pi) - 1\}$, this ensures that the abovementioned horizontal line always exists, and splits the sum of all $h(\pi) + 1$ terms into two parts such that the smaller one is at least $\frac{1}{3}$ of the total.

In fact, the disclosure of all labels of the nodes in π when $z < h(\pi)$ would decompose T' into $(h(\pi) - z)$-many subtrees whose labelings are independent of one another. For all $z \in \{0, 1, \ldots, h(\pi) - 1\}$, let us denote for brevity $\prod_{k=z}^{h(\pi)-1} |S_t(j_k^s)|$ by $\mathcal{S}_z$, and also denote for convenience $|S_t^{y(r')=1}(r')|$ by $\mathcal{S}_{h(\pi)}$ Notice that, by definition, $|S_t^{y(r')=1}(r')| = 1$, and corresponds to the special case $z = h(\pi)$. With this notation, it is now important to note that i_t^* must be the parent of $j_{z^*}^s$, for some $z^* \in \{0, 1, \ldots, h(\pi) - 1\}$, and that $\left|S_t^{y(i_t^*)=0}(r')\right| = \mathcal{S}_{z^*}$. Thus, taking into account all possible $(h(\pi) + 1)$ labelings of π, the cardinality of $S_t(r')$ can be written as follows:

$$|S_t(r')| = \sum_{z=0}^{h(\pi)} \mathcal{S}_z \ .$$

At this point, by definition, we have:

(i) $\mathcal{S}_0 = \mathcal{S}_1$, as $|S_t(j_0^s)| = 1$, which in turn implies $\max_z \mathcal{S}_z \leq \frac{|S_t(r')|}{2}$, and
(ii) $\mathcal{S}_z \geq \mathcal{S}_{z+1}$ for all $z \in \{0, \ldots, h(\pi) - 1\}$.

See Figure 5 for a pictorial illustration.

The proof is now concluded by contradiction. If our statement is false, then there must exist a value z' such that $\mathcal{S}_{z'} > \frac{2}{3}|S_t(r')|$ and $\mathcal{S}_{z'+1} < \frac{1}{3}|S_t(r')|$. However, because the sequence $\langle \mathcal{S}_0, \mathcal{S}_1, \ldots \mathcal{S}_{h(\pi)} \rangle$ is monotonically decreasing and we have $\mathcal{S}_0 \leq \frac{|S_t(r')|}{2}$, implying $\mathcal{S}_0 \leq \sum_{z=1}^{h(\pi)} \mathcal{S}_z$, such value z' cannot exist. Thus, it must exist z such that

$$\frac{1}{3}|S_t(r')| \leq \mathcal{S}_z \leq \frac{2}{3}|S_t(r')| \ .$$

Let z^* be the smallest z satisfying the above inequalities. Note that i_t^* is the parent of $j_{z^*}^s$, because of the bottom-up search on π performed by OTS. Exploiting again the monotonicity of the sequence $\langle \mathcal{S}_0, \mathcal{S}_1, \ldots \mathcal{S}_{h(\pi)} \rangle$ and recalling that $\mathcal{S}_{z^*} = \prod_{k=z^*}^{h(\pi)-1} |S_t(j_k^s)|$, we conclude that

$$\frac{1}{3}|S_t(r')| \leq \left|S_t^{y(i_t^*)=0}(r')\right| \leq \frac{2}{3}|S_t(r')| \ .$$

Since $\left|S_t^{y(i_t^*)=1}(r')\right| + \left|S_t^{y(i_t^*)=0}(r')\right| = |S_t(r')|$, we must also have

$$\frac{1}{3}|S_t(r')| \leq \left|S_t^{y(i_t^*)=1}(r')\right| \leq \frac{2}{3}|S_t(r')|,$$

thereby concluding the proof. □

From the above proof, one can see that it is indeed necessary that π is a backbone path, since the proof hinges on the fact that $|S_t(j_0^{\mathrm{s}})| = 1$. In fact, if $|S_t(j_0^{\mathrm{s}})|$ is larger than $2\sum_{z=1}^{h(\pi)} |\mathcal{S}_z|$, that is larger than $\frac{2}{3}|S_t(r')|$ (which may happen if π is not a backbone path), we would not have $\max_z \mathcal{S}_z \leq \frac{|S_t(r')|}{2}$, hence $\mathcal{S}_z$ would not be guaranteed to be at least $\frac{2}{3}|S_t(r')|$ for all z.

Proof of Theorem 1

Proof. By Lemma 1, we immediately see that OTS finds c^* through $\mathcal{O}(\log N)$ queries. This is because $|S_{t+1}(r)| \leq \frac{2}{3}|S_t(r)|$ for all time steps t, implying by induction that the total number of queries is upper bounded by $\log_{3/2} N = \mathcal{O}(\log N)$.

We now sketch an implementation of OTS which requires $\mathcal{O}(n + h \log N)$ time and $\mathcal{O}(n)$ space.

In a preliminary phase, we compute in a bottom-up fashion the values $|S_0(i)|$ for all nodes $i \in V(T)$. This requires $\mathcal{O}(n)$. Thereafter, we perform a breath-first search on T, and each time we visit a leaf of T, we insert a pointer to it in a an array A in a sequential way. Thus, the j-th record of A will contain a reference to the j-th leaf found during this visit, which entails that the leaves referred by the pointers of A are sorted in ascending order of depth.

We recall that in the noiseless setting, each time the label of a node is revealed and is equal to 1 (to 0), also the labels of its descendants (ancestors) are indirectly revealed, because they are known to be equal to 1 (to 0). The total time OTS takes for assigning all indirectly revealed labels is clearly $\mathcal{O}(n)$. Each time OTS needs to find a backbone of a tree in the current forest F, we look for the largest index j for which the record $A[j]$ does not point to a leaf whose parent label has not been revealed yet. Observe that, at any time t, the deepest leaf $\ell \in L(T)$ satisfying this property must be the terminal node of a backbone path of a tree in F. Furthermore, the highest node of such backbone is either $r(T)$ or the lowest ancestor of ℓ whose label has not been revealed yet, and can therefore be found in $\mathcal{O}(h)$ time.

In order to accomplish this leaf search operation, we simply maintain over time an index that scans A from $A[n]$ to $A[1]$, looking for a leaf satisfying the above property. The total time OTS uses for scanning A is again linear in n. Finally, for each query, OTS traverses bottom-up a backbone π, exploiting the information previously stored to find i_t^*, and updates it after $y(i_t^*)$ is revealed. Note that only the information of the nodes in π has to be updated. In fact, the disclosure of the label of any node $i \in V(T)$ cannot affect the values of $S_t(j)$ for all nodes $j \in V(T)$ that are *not* ancestors of i. Besides, we are free to disregard the descendants of i since they will simply be indirectly labeled (by 1).

Overall, the total time required by this implementation of OTS is the sum of $\mathcal{O}(n)$ and $\mathcal{O}(h)$ times the total number of queries the algorithm makes, which results in the claimed $\mathcal{O}(n + h \log N)$ upper bound. The claim on the memory requirement immediately follows from the above description. □

B.2 Proof of the lower bound in Theorem 2

Proof. Let T' be the subtree of T constructed by visiting T from its root (for instance by a breadth-first or a depth-first visit), and such that $|L(T')| = B$. Note that the construction of T' satisfying this constraint is always possible because the maximum cardinality of $L(T')$ is equal to $|L_{\mathrm{s}}(T)|$ (which is also equal to $\max_{c^*} \widetilde{K}(T, c^*)$). For each leaf $\ell \in L(T')$, consider all cuts c^* that can be generated by cutting either the edge connecting ℓ with its parent or the two edges connecting ℓ with its children. The total number of such cuts is $2^{|L(T')|} = 2^B$. We set the prior $\mathbb{P}(\cdot)$ to be uniform over these 2^B-many cuts. Hence, for each leaf $\ell \in L(T')$, the probability (w.r.t. $\mathbb{P}(\cdot)$) that c^* cuts the edge connecting ℓ with its parent is 1/2, and so is the probability that c^* cuts the two edges connecting ℓ with its children.

Now, observe that, by construction, we have $B \leq \widetilde{K}(T, c^*) \leq 2B$ for *all* such cuts c^* and, as a consequence, $B \leq \mathbb{E}[\widetilde{K}(T, c^*)] \leq 2B$, the expectation being over $\mathbb{P}(\cdot)$. Since for each leaf of T' any (possibly randomized) active learning algorithm A has to make $\frac{1}{2}$ mistake in expectation (over $\mathbb{P}(\cdot)$ and its internal randomization), we conclude that $B/2$ queries are always necessary to find c^*, as claimed. □

Algorithm 1: WDP (Weighted Dichotomic Path)

▷ **INPUT** : T, $\{\boldsymbol{p}(i), i \in V\}$.

▷ **OUTPUT**: $\widehat{\mathcal{C}} = \mathcal{C}^*$.

Init:
- $\widehat{\mathcal{C}} \leftarrow \emptyset$; /* $\widehat{\mathcal{C}}$ contains all the clusters of $\mathcal{C}(c^*)$ found so far */
- $F \leftarrow \{T\}$; /* Forest of maximal subtrees T' of T */
- $y(\text{par}(r)) \leftarrow 0$; /* Dummy node $\text{par}(r)$ */
- **for** $i \in V$ **do** $q(i) \leftarrow \boldsymbol{p}(i) \cdot \prod_{j \in \pi(\text{par}(i), r(T))}(1 - \boldsymbol{p}(j))$;

/* -- Path with maximum entropy -- */

while $F \neq \emptyset$ **do**

Let $L(F)$ be the set of all leaves of T belonging to the subtrees in F.

Let $R(F)$ be the set of all roots of the subtrees in F.

$\pi(\ell, r') \leftarrow \arg\max_{\ell \in L(F), r' \in R(F)} H(\pi(\ell, r'))$;

$\mathcal{T} = \{T' \in F : \ell, r' \in V(T')\}$;

/* -- Binary search on path $\pi(\ell, r')$ -- */

$u \leftarrow \ell$; $v \leftarrow r'$;

while $u \neq v$ **do**

Let $\langle i_0 = u, i_1, \ldots, i_{h-1}, i_h = v \rangle$ be the sequence of nodes lying on $\pi(u, v)$ in descending order of depth.

$Q \leftarrow \sum_{k \in \{0,1,\ldots,h-1\}} q(i_k)$;

$k^* = \arg\min_{k \in \{0,1,\ldots,h-1\}} \left| \frac{Q}{2} - \sum_{j \in \{i_0, \ldots, i_k\}} q(j) \right|$;

$i^* \leftarrow \text{par}(i_{k^*})$;

Query $y(i^*)$;

if $y(i^*) = 0$ **then**

$v \leftarrow i_{k^*}$;

else

$u \leftarrow i^*$;

Set $\boldsymbol{p}(i) = y(i) = 1$ for all descendants i of u;

Set $\boldsymbol{p}(i) = y(i) = 0$ for all ancestors $i \neq u$ of u;

$q(u) \leftarrow 1$; Set $q(i) = 0$ for all descendants and ancestors $i \neq u$ of u;

Update $\boldsymbol{p}(i)$ and $q(i)$ for all descendants of all $i \in V(\pi(\text{par}(u), r'))$ such that $i \neq r'$;

$\widehat{\mathcal{C}} \leftarrow \widehat{\mathcal{C}} \cup \{L(u)\}$; /* $L(u)$ is a cluster of $\mathcal{C}^*$ */

/* -- Update F -- */

$F \leftarrow F \setminus \mathcal{T}$; /* Remove from F the subtree containing $\pi(\ell, r')$ */

$j \leftarrow \text{par}(u)$; /* Lowest node in $\pi(\ell, \text{par}(r'))$ with label known to be 0 */

while $j \in V(\pi(\ell, r'))$ **do**

Let j_c be the child of j that is not in $\pi(\ell, r')$.

if $j_c \in L$ **then**

$\widehat{\mathcal{C}} \leftarrow \widehat{\mathcal{C}} \cup \{j_c\}$; /* Add to $\widehat{\mathcal{C}}$ a singleton cluster */

else

$F \leftarrow F \cup \{T(j_c)\}$; /* j_c is the root of a subtree that will be processed later */

Update $q(i)$ for all $i \in V(T(j_c))$;

$j \leftarrow \text{par}(j)$; /* j is a node whose label is known to be 0 */

return $\widehat{\mathcal{C}}$.

B.3 Weighted Dichotomic Path (WDP)

In Algorithm 1 we give the pseudocode of WDP. At each round, WDP finds the path whose entropy is maximized over all bottom-up paths $\pi(\ell, r')$, with $\ell \in L$ and $r' = r(T')$, where T' is the subtree in F containing ℓ. Ties are broken arbitrarily. WDP performs a binary search on such $\pi(\ell, r')$ to find the edge of T' which is cut by c^*, taking into account the current values of $q(i)$ over that path.

Specifically, let $\langle i_0 = \ell, i_1, \ldots, i_{h-1}, i_h = r' \rangle$ be the sequence of nodes in $\pi(\ell, r')$ in descending order of depth. WDP finds an index k^* that corresponds to the middle point in $\pi(\ell, r')$, taking into account the current values of $q(i)$ over that path. Let $i^* = \mathrm{par}(i_{k^*})$. WDP queries the label of i^*: If $y(i^*) = 0$, WDP continues the binary search on $\pi(\ell, i_{k^*})$; if instead $y(i^*) = 1$, the binary search continues on $\pi(i^*, r')$, and so on. During the binary search, whenever WDP finds a node $u \in V(\pi(\ell, r'))$ with queried labels $y(u) = 0$ and $y(\mathrm{par}(u)) = 1$, then the edge of $\pi(\ell, r')$ cut by c^* has been found, and the binary search on this backbone path terminates. In the special case where $y(r') = 1$, the binary search also ends, and we know that all nodes in $L(r')$ form a cluster of $\mathcal{C}(c^*)$. Once a binary search terminates, WDP updates F and the probabilities $\boldsymbol{p}(i)$ at all nodes i in the subtrees of F, so as to reflect the new knowledge gathered by the queried labels.

Below, we prove WDP's query complexity. The proofs are split into a series of lemmas.

Lemma 2 *Given tree T with set of leaves L, any prior $\mathbb{P}(\cdot) \in \mathcal{P}_{>0}$ over c^*, and any $c^* \sim \mathbb{P}(\cdot)$, let j_0 be a node of $\mathcal{AB}(c^*)$, having as children a leaf $\ell \in L$ and an internal node j' of T (see Figure 6, left). Then, during its execution,* WDP *will never select the bottom-up path starting from ℓ.*

Proof. Let T_0 be the tree made up of all nodes of $\mathcal{AB}(c^*)$, and consider any given round with $q(i)$ in (3) defined by the current *posterior* distribution maintained by the algorithm. We first show that, for all ancestors a of j_0, path $\pi(\ell, a)$ cannot be selected by WDP, because its entropy[5] $H(\pi(\ell, a))$ will always be strictly smaller than $H(\pi(\ell', a))$ for *all* leaves $\ell' \in L(j')$. To this effect, we can write

$$\begin{aligned} H(\pi(\ell, a)) - H(\pi(\ell', a)) &= \left(-q(\ell)\log_2 q(\ell) - \sum_{u \in \pi(j_0, a)} q(u)\log_2 q(u) \right) \\ &\quad - \left(H(\pi(\ell', j') - \sum_{u \in \pi(j_0, a)} q(u)\log_2 q(u) \right) \\ &= -q(\ell)\log_2 q(\ell) + \sum_{v \in \pi(\ell', j')} q(v)\log_2 q(v)\ . \end{aligned} \tag{5}$$

Now, since

$$\sum_{u \in \pi(\ell', j')} q(u) + \sum_{v \in \pi(j_0, a)} q(v) = \sum_{u \in \pi(\ell', a)} q(u) = \sum_{u \in \pi(\ell, a)} q(u) = q(\ell) + \sum_{u \in \pi(j_0, a)} q(u)\ ,$$

we have $q(\ell) = \sum_{v \in \pi(\ell', j')} q(v)$.

Consider the function $f(x) = -x\log_2 x$, for $x \in [0, 1]$. For all $x, x_1, x_2 \in (0, 1)$ such that $x_1 + x_2 = x$, it is easy to verify that we have $f(x) < f(x_1) + f(x_2)$. More generally, for all $x, x_1, x_2, \ldots, x_m \in (0, 1)$ with $\sum_{i=1}^m x_i = x$, one can show that $f(x) < \sum_{i=1}^m f(x_i)$. Since $|V(\pi(\ell', j'))| \geq 2$ (holding because $j' \notin L$ implies $\ell' \neq j'$), the above inequality on $f(\cdot)$ allows us to write

$$-q(\ell)\log_2 q(\ell) < -\sum_{v \in \pi(\ell', j')} q(v)\log_2 q(v)\ ,$$

i.e., (5) < 0. Notice that the assumption $\mathbb{P}(\cdot) \in \mathcal{P}_{>0}$ implies $q(v) > 0$ at any stage of the execution of WDP where node v has an unrevealed label. This is because, after any binary search on a path selected by WDP, for all v belonging to any tree in F, in the update phase each value $q(v)$ is multiplied by a strictly positive value. This ensures that we can use the above inequality about $f(\cdot)$, as its argument will always lie in the open interval $(0, 1)$.

The inequality in (5) implies that there always exists a leaf ℓ' of $T(j')$ such that WDP selects the path connecting ℓ' with the root of the tree containing ℓ in the current forest F. This selection entails the disclosure of either cut edge (j', j_0) (if $j_0 \in L(T_0)$), or a cut edge in $T(j')$ (if $j_0 \notin L(T_0)$), which in turn implies that the labels of i_0 and all its ancestors will be disclosed to the algorithm to be equal to 0, thereby indirectly revealing also cut edge (ℓ, i_0). Since F contains only trees whose height is larger than 1, after this cut edge disclosure the tree made up of leaf ℓ alone cannot be part of F, thus preventing WDP's selection of a path starting from ℓ. □

[5] Here, we are defining the entropy of a path π as $-\sum_{v \in V(\pi)} q(v)\log_2 q(v)$, even for paths π for which $\sum_{v \in V(\pi)} q(v) < 1$.

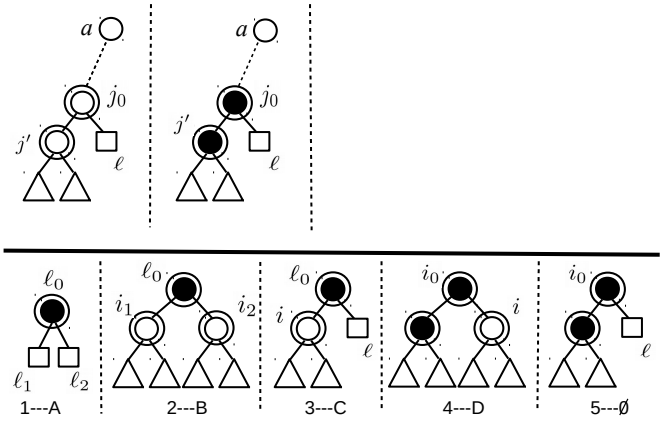

Figure 6: Illustration of all possible cases of Lemma 2 and Lemma 3. Nodes belonging to T_0 (see main text) are black, all remaining nodes are white. Leaves and subtrees of T are represented by squares and triangles, respectively. Each node of T'_{c^*} is enclosed in a circle. **Above:** The two possible cases illustrating Lemma 2, that is, $j' \notin V(T_0)$ on the left, and $j' \in V(T_0)$ on the right. **Below:** The five cases described in Lemma 3.

Lemma 3 *For any input tree T and any cut c^* with $\mathbb{P}(\cdot) \in \mathcal{P}_{>0}$, the number of paths selected by* WDP *before stopping is $\widetilde{K}(T, c^*)$.*

Proof. If c^* has only one cluster the statement is clearly true, since the binary search performed by WDP on the first selected path reveals that $y(r) = 1$ (hence $y(v) = 1$ for all $v \in V$). We then continute by assuming $y(r) = 0$, so that c^* has least two clusters.

Let Π be the set of all paths selected by WDP during the course of its execution. The binary search perfomed by WDP on *each* of such paths, discloses exactly *one* edge of c^*. Let c^*_{wdp} be the set containing all these cut edges, and c^*_0 be the set of the remaining cut edges of c^*. We show that, for any T and any cut c^* of T, $|c^*_{wdp}| = \widetilde{K}(T, c^*)$, while all edges in c^*_0 are *indirectly* disclosed by WDP, although none of them belongs to paths in Π.

Let T_0 be the subtree of T made up of all nodes in $\mathcal{AB}(c^*)$. The edges of c^* can be partitioned into the five disjoint sets $S_1, \ldots, S_5$ (see Figure 6 for reference):

- S_1: The set of all *pairs* of edges connecting a leaf ℓ_0 of T_0 to two sibling leaves ℓ_1 and ℓ_2 of T (Figure 6, below, 1);
- S_2: The set of all *pairs* of edges connecting a leaf ℓ_0 of T_0 to two sibling internal nodes i_1 and i_2 of T (Figure 6, below, 2);
- S_3: The set of all *pairs* of edges connecting a leaf ℓ_0 of T_0 to a leaf $\ell \in L$ and an internal node i of T (Figure 6, below, 3);
- S_4: The set of all edges connecting an internal node i_0 of T_0 to an internal node i of T, so that the sibling node of i belongs to $V(T_0)$ (Figure 6, below, 4);
- S_5: The set of all edges connecting an internal node i_0 of T_0 to a leaf ℓ of T, so that the sibling node of ℓ belongs to $V(T_0)$ (Figure 6, below, 5).

Recall that T'_{c^*} is the subtree of T whose nodes are $(\mathcal{AB}(c^*) \cup \mathcal{LB}(c^*)) \setminus L$, and that $\widetilde{K}(T, c^*)$ is the number of its leaves. The leaves of T'_{c^*} can be partitioned into the following four sets A, B, C, and D (see again Figure 6 for reference):

- A: The set of all leaves of T'_{c^*} that are also leaves of T_0, i.e., that belong to $\mathcal{AB}(c^*)$;
- B: The set of all *sibling* leaves of T'_{c^*} that are also (sibling) internal nodes of T;
- C: The set of all leaves of T'_{c^*} that are also internal nodes of T such that their sibling node is a leaf of T;
- D: The set of all leaves of T'_{c^*} that are also internal nodes of T such that their sibling node belongs to T_0.

We will not show a one-to-one mapping between $L(T'_{c^*})$ and the cut edges of c^*_{wdp} covering all possible cases.

$S_1 \leftrightarrow A$: For each pairs of cut edges in S_1), WDP clearly selects a path starting from either ℓ_1 or ℓ_2, which will indirectly disclose the cut edge incident to the sibling leaf (ℓ_2 or ℓ_1, respectively). S_1 is therefore about all leaves of set A.

$S_2 \leftrightarrow B$: For each pairs of cut edges in S_2, WDP selects two paths, one per cut edge. Each of these two paths clearly contains one of these two cut edges, and corresponds to all leaves of T'_{c^*} that are also leaves of T. Hence we are covering all leaves of set B.

$S_3 \leftrightarrow C$: For the edges in S_3, WDP selects only one path, starting from a leaf of $T(i)$. This path clearly contains edge (i, ℓ_0), and covers all leaves of set C. Observe that, by Lemma 2, edge (ℓ_0, ℓ) is always indirectly revealed and never contained in a path selected by WDP.

$S_4 \leftrightarrow D$: For the edges in S_4, whenever WDP selects a path starting from a leaf of $T(i)$, all the nodes in $V(T(i))$ are indirectly labeled 1, and from that point on, they will not be included in a tree in F. This path clearly contains edge (i_0, i), hence we are covering all leaves of set D.

$S_5 \leftrightarrow \emptyset$: Finally, Lemma 2 ensures that all cut edges in S_5 are indirectly disclosed whenever WDP selects a path starting from a leaf belonging to $T(i'_0)$, where i'_0 is the sibling node of ℓ. Hence this case is ruled out by Lemma 2, and does not correspond to any leaf.

From the above, we conclude that the number of paths selected by wdp is always equal to $\widetilde{K}(T, c^*)$, as claimed. □

The next lemma provides an entropic bound on the (condionally) expected number of queries WDP makes on a given path. Notice that the posterior distribution maintained by WDP never changes *during* each binary search, but only between a binary search and the next. Consider then $q(i)$ defined in (3) at the beginning of a given binary search in terms of the current posterior distribution, and let π be the selected path, after having observed the labels that generated the current posterior.

Lemma 4 *Let π be any path selected by* WDP *during the course of its execution, and $\{q(i)\}$ be the current distribution (3) at the time π is selected. Then the expected number of queries* WDP *makes on π, conditioned on past revealed labels, is $\mathcal{O}\left(\lceil H(\pi) \rceil\right) = \mathcal{O}\left(\log(|V(\pi)|)\right)$. Here, both the conditional expectation and $H(\pi)$ are defined i.t.o. $\{q(i)\}$.*

Proof. Let π be the currently selected path, and denote by $E_{\text{par}}(\pi)$ the set made up of the edges in π along with the edge connecting the top node of π to its parent (recall that in the special case where r is a terminal node of π, we can view r as the child of a dummy "super-root"). The binary search performed on π guarantees that the number of queries $Q(\pi, (u, v))$ made by WDP to find a cut edge (u, v) lying on π can be quantified as follows:

$$\left\lceil \log_2 \left(\frac{\sum_{(u',v') \in E_{\text{par}}(\pi)} \mathbb{P}((u', v') \in c^*)}{\mathbb{P}((u, v) \in c^*)} \right) \right\rceil = \left\lceil \log_2 \left(\frac{1}{\mathbb{P}((u, v) \in c^*)} \right) \right\rceil ,$$

where the probabilities above are defined w.r.t. the posterior distribution at the beginning of the binary search. The expected number of queries made on π, conditioned on past labels can thus be bounded as

$$\begin{aligned} \sum_{(u',v') \in E_{\text{par}}(\pi)} \mathbb{P}((u', v') \in c^*) \left\lceil \log_2 \left(\frac{1}{\mathbb{P}((u', v') \in c^*)} \right) \right\rceil &= \sum_{u \in V(\pi)} q(i) \left\lceil \log_2 \left(\frac{1}{q(i)} \right) \right\rceil \\ &= \mathcal{O}\left(\lceil H(\pi) \rceil\right) \\ &= \mathcal{O}\left(\log(|V(\pi)|)\right) , \end{aligned}$$

as claimed. □

We are now ready to prove Theorem 3 and Theorem 6.

Proof.[Theorem 3] For given c^*, let $\Pi = \Pi(c^*) = \langle \pi_1, \ldots, \pi_{|\Pi|} \rangle$ be the sequence of paths selected by WDP, sorted in the temporal order of selection during WDP's run. Also, denote by $Q(\pi_j)$ the number of queries made by WDP on $\pi_j \in \Pi$. Notice that the sequence Π is fully determined by c^*. Moreover, the paths in Π are orderer in such a way to guarantee that π_j contains a unique edge

$(\mathrm{par}(u_j), u_j)$ which c^* cuts across. Then, if we denote by $\{q_j(\cdot)\}$ the value of $q(\cdot)$ at the time path π_j is selected, it is easy to see that cut c^* can be alternatively generated by sequentially generating edge $(\mathrm{par}(u_1), u_1)$ according to distribution $\{q_1(\cdot)\}$ over π_1, then $(\mathrm{par}(u_2), u_2)$ according to (posterior) distrubution $\{q_2(\cdot)\}$ over π_2, then $(\mathrm{par}(u_3), u_3)$ according to (posterior) distrubution $\{q_3(\cdot)\}$ over π_3, and so on until $|\Pi|$ cuts have been generated. From Lemma 3, we have $|\Pi| = \widetilde{K}(T, c^*)$.

Let us then denote by $\mathbb{E}[\cdot]$ the expectation w.r.t. the *prior* distribution, and by $\mathbb{E}_j[\cdot]$ be the conditional expectation $\mathbb{E}[\cdot \,|\, (\mathrm{par}(u_1), u_1), (\mathrm{par}(u_2), u_2), \ldots, (\mathrm{par}(u_{j-1}), u_{j-1})]$. Notice that the sequence of random variables $\mathrm{par}(u_1), u_1), (\mathrm{par}(u_2), u_2), \ldots, (\mathrm{par}(u_{j-1}), u_{j-1})$ fully determines the posterior distribution $\{q_j(\cdot)\}$ before the selection of the j-th path π_j, and so, π_j itself. This way of viewing c^* makes $\widetilde{K} = \widetilde{K}(T, c^*)$ a (finite) stopping time w.r.t. the sequence of random variables $\mathrm{par}(u_1), u_1), (\mathrm{par}(u_2), u_2), \ldots,$, in that $\{\widetilde{K} \geq j\}$ is determined by $(\mathrm{par}(u_1), u_1), (\mathrm{par}(u_2), u_2), \ldots, (\mathrm{par}(u_{j-1}), u_{j-1})$. This allows us to write

$$
\begin{aligned}
\mathbb{E}\left[\sum_{j=1}^{\widetilde{K}} Q(\pi_j)\right] &= \sum_{i=1}^{n}\sum_{j=1}^{i} \mathbb{E}\left[Q(\pi_j)\{\widetilde{K} = i\}\right] \\
&= \sum_{j=1}^{n} \mathbb{E}\left[Q(\pi_j)\{\widetilde{K} \geq j\}\right] \\
&= \sum_{j=1}^{n} \mathbb{E}\left[\{\widetilde{K} \geq j\}\mathbb{E}_j[Q(\pi_j)]\right] \qquad \text{(since } \widetilde{K} \text{ is a stopping time)} \\
&= \sum_{j=1}^{n}\sum_{i=j}^{n} \mathbb{E}\left[\{\widetilde{K} = i\}\mathbb{E}_j[Q(\pi_j)]\right] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{i} \mathbb{E}\left[\{\widetilde{K} = i\}\mathbb{E}_j[Q(\pi_j)]\right] \\
&= \sum_{i=1}^{n} \mathbb{E}\left[\{\widetilde{K} = i\}\sum_{j=1}^{\widetilde{K}} \mathbb{E}_j[Q(\pi_j)]\right] \\
&= \mathbb{E}\left[\sum_{j=1}^{\widetilde{K}} \mathbb{E}_j[Q(\pi_j)]\right] \\
&= \mathcal{O}\left(\mathbb{E}\left[\sum_{j=1}^{\widetilde{K}} \lceil H_j(\pi_j)\rceil\right]\right) \qquad \text{(by Lemma 4, where } H_j(\cdot) \text{ is w.r.t. } \{q_j(\cdot)\}) \\
&= \mathcal{O}\left(\mathbb{E}[\widetilde{K}]\log h\right) \,,
\end{aligned}
\tag{6}
$$

as claimed □

A slightly more involved guarantee for WDP is given by the following theorem, where the query complexity depends in a more detailed way on interplay between T and the prior $\mathbb{P}(\cdot)$. Given any bottom-up path π in T, we denote by $\widetilde{H}(\pi)$ the *normalized* entropy of π, defined as $\widetilde{H}(\pi) = -\sum_{i \in V(\pi)} \widehat{q}(i)\log(\widehat{q}(i))$, where $\widehat{q}(i) = q(i)/\sum_{i \in V(\pi)} q(i)$, and $q(i)$ is defined according to the prior distribution $\mathbb{P}(\cdot)$, as in (3). Notice that we may have $\sum_{i \in V(\pi)} q(i) < 1$. Further, denote by $\mathbb{D}$ the family of all sets Π of all vertex-disjoint bottom-up paths starting from T's leaves such that the top terminal node of each path $\pi \in \Pi$ is either the root r of T or a node of another path of Π. The upper bound in the following theorem is *never* worse than the upper bound in Theorem 3.

Theorem 6 *In the noiseless realizable setting, for any tree T, any prior distribution $\mathbb{P}(\cdot)$ over c^* such that $\mathbb{P}(\cdot) \in \mathcal{P}_{>0}$, the expected number of queries made by* WDP *to find c^* is $\mathcal{O}\left(\max_{\Pi \in \mathbb{D}} \sum_{j=1}^{m(\Pi)} \lceil \widetilde{H}(\pi_{i_j})\rceil\right)$, where $m(\Pi) = \min\left\{\left\lceil \mathbb{E}\widetilde{K}(T, c^*)\right\rceil, |\Pi|\right\}$, and paths $\pi_{i_1}, \pi_{i_2}, \ldots$*

in $\Pi \in \mathbb{D}$ *are sorted in non-increasing value of normalized entropy* $\widetilde{H}(\cdot)$*. In the above, the expectations is again over* $\mathbb{P}(\cdot)$*.*

As an application of the above result, consider that oftentimes a linkage function generating T also tags each internal node i with a coherence level α_i of $T(i)$, which is typically increasing as we move downwards from root to leaves. A common situation in hierarchical clustering is then to figure out the "right" level of granularity of the flat clustering we are looking for through the definition of bands of nodes (i.e., bands of clusters) of similar coherence. This may be encoded through a prior $\mathbb{P}(\cdot)$ that uniformly spreads $(1-\epsilon)/b$ probability mass over b-many edge-disjoint cuts of T, for $b \ll h$, and an arbitrarily small $\epsilon > 0$, and the remaining mass ϵ over all remaining cuts (this is needed to comply with the condition $\mathbb{P}(\cdot) \in \mathcal{P}_{>0}$). As we said in the main body of the paper, Theorem 6 gives a bound of the form $\mathbb{E}\widetilde{K}(T, c^*) \log b$ as opposed to the bound $\mathbb{E}\widetilde{K}(T, c^*) \log h$ provided by Theorem 3.

Proof of Theorem 6

Proof. Given T and prior $\mathbb{P}(\cdot)$, let $\mathbb{D}_{wdp}$ be the set made up of all sets Π of bottom-up paths in T that WDP can potentially select during the course of its executions. Each set Π is uniquely determined by $c^* \sim \mathbb{P}(\cdot)$. The family of sets $\mathbb{D}$ is clearly a superset of $\mathbb{D}_{wdp}$. We prove the theorem by showing that the expected number of queries made by WDP is upper bounded by

$$\mathcal{O}\left(\max_{\Pi \in \mathbb{D}_{wdp}} \sum_{j=1}^{m(\Pi)} \left\lceil \widetilde{H}(\pi_{i_j}) \right\rceil\right) , \tag{7}$$

where, for any given $\Pi \in \mathbb{D}_{wdp}$, $\pi_1, \pi_2, \ldots$ is the sequence of paths of Π in the order they are selected by WDP, while $\pi_{i_1}, \pi_{i_2}, \ldots$ is the same sequence rearranged in non-increasing order of $\widetilde{H}(\cdot)$. Using the same notation as in the proof of Theorem 3, we observe that at the time when π_j gets selected by WDP the distribution $\{q_j(\cdot)\}$ sitting along path π_j is precisely the normalized distribution $\{\widehat{q}(\cdot)\}$ such that $\sum_{i=1}^{|V(\pi_j)|} \widehat{q}(i) = 1$, so that $H_j(\pi_j) = \widetilde{H}(\pi_j)$. Then, Eq. (6) combined with Lemma 4 allows us to write

$$\mathbb{E}\left[\sum_{j=1}^{\widetilde{K}} Q(\pi_j)\right] = \mathbb{E}\left[\mathcal{O}\left(\sum_{j=1}^{\widetilde{K}} \lceil \widetilde{H}_j(\pi_j) \rceil\right)\right] .$$

In the sequel, we show how to upper bound the right-hand side of the last (in)equality by (7). Set for brevity $\mathbb{E}[\widetilde{K}] = \lceil \mu \rceil$. We have

$$\begin{aligned}
\sum_{j=1}^{\widetilde{K}} \lceil \widetilde{H}_j(\pi_j) \rceil &= \sum_{j=1}^{\widetilde{K}} \{\widetilde{K} < \mu\} \lceil \widetilde{H}_j(\pi_j) \rceil + \sum_{j=1}^{\widetilde{K}} \{\widetilde{K} \geq \mu\} \lceil \widetilde{H}_j(\pi_j) \rceil \\
&\leq \sum_{j=1}^{\mu} \lceil \widetilde{H}_j(\pi_{i_j}) \rceil + \frac{\widetilde{K}}{\mu} \sum_{j=1}^{\widetilde{K}} \lceil \widetilde{H}_j(\pi_j) \rceil \\
&\leq \max_{\Pi \in \mathbb{D}_{wdp}} \sum_{j=1}^{m(\Pi)} \lceil \widetilde{H}_j(\pi_{i_j}) \rceil + \frac{\widetilde{K}}{\mu} \max_{\Pi \in \mathbb{D}_{wdp}} \sum_{j=1}^{m(\Pi)} \lceil \widetilde{H}_j(\pi_{i_j}) \rceil \\
&= \left(1 + \frac{\widetilde{K}}{\mu}\right) \max_{\Pi \in \mathbb{D}_{wdp}} \sum_{j=1}^{m(\Pi)} \lceil \widetilde{H}_j(\pi_{i_j}) \rceil ,
\end{aligned}$$

so that, taking the expectation of both sides,

$$\mathbb{E}\left[\sum_{j=1}^{\widetilde{K}} \lceil \widetilde{H}_j(\pi_j) \rceil\right] \leq 2 \max_{\Pi \in \mathbb{D}_{wdp}} \sum_{j=1}^{m(\Pi)} \lceil \widetilde{H}_j(\pi_{i_j}) \rceil .$$

This concludes the proof. □

B.4 N-WDP (Noisy Weighted Dichotomic Path)

N-WDP is a robust variant of WDP that copes with persistent noise. Given an internal node $i \in V \setminus L$, let $\mathcal{L}(i)$ be the set of all possible queries that can be made to determine $y(i)$, i.e., the set $(\ell, \ell') \in L(\text{left}(i)) \times L(\text{right}(i))$. Then, given confidence $\delta \in (0, 1]$, and noise level $\lambda \in [0, 1/2)$, N-WDP:

1. Preprocesses T and prior $\mathbb{P}(\cdot)$ by setting $y(i) = 1$ for all nodes $i \in V \setminus L$ such that $|\mathcal{L}(i)| < \frac{\alpha \log(n/\delta)}{(1-2\lambda)^2}$, for a suitable constant $\alpha > 0$. $\mathbb{P}(\cdot)$ is also updated (all $j \in T(i)$ have $\boldsymbol{p}(j) = 1$). At the end of this phase, each node in V is either unlabeled or labeled with 1.
2. Let T_λ be the subtree of T made up of all unlabeled nodes of T, together with all nodes whose label has been set to 1 that are children of unlabeled nodes. N-WDP operates on T_λ as WDP, with the following difference: Whenever a label $y(i)$ is requested, N-WDP determines its value by a majority vote over $\Theta\left(\frac{\log(n/\delta)}{(1-2\lambda)^2}\right)$-many queries selected uniformly at random from $\mathcal{L}(i)$.

Proof sketch of Theorem 4

Proof. Let Λ be the set of pairs of leaves whose label has been corrupted by noise. A standard Chernoff bound implies that for any fixed subset of $L \times L$ containing at least $\alpha \frac{\log(1/\delta)}{(1-2\lambda)^2}$ pairs (for a suitable constant $\alpha > 0$), the probability that the majority of them belongs to Λ is at most δ. Let us set for brevity $f(n, \lambda, \delta) = \alpha \frac{\log(n/\delta)}{(1-2\lambda)^2}$. A union bound over the at most $n - 1$ internal nodes of V guarantees that for all queries $y(i)$ made by N-WDP operating on T_λ the majority vote over $f(n, \lambda, \delta)$-many queries on pairs of leaves of $\mathcal{L}(i)$ will produce the correct label (i.e., before noise) of that node with probability at least $1 - \delta$.

Moreover, since the cut $\widehat{c}$ found by N-WDP on T_λ can be obtained with probability at least $1 - \delta$ from c^* by merging zero or more clusters on T, it is immediate to see that $\widetilde{K}(T_\lambda, \widehat{c}) \leq \widetilde{K}(T, c^*)$. It is also easy to verify that this inequality holds even in expectation over the prior distributions of cut c^* on T and $\widehat{c}$ on T_λ, that is, $\mathbb{E}_{\mathbb{P}_\lambda} \widetilde{K}(T_\lambda, \widehat{c}) \leq \mathbb{E}_{\mathbb{P}} \widetilde{K}(T, c^*)$, where $\mathbb{P}_\lambda$ denotes the modified prior on tree T_λ produced after N-WDP's initial preprocessing (Step 1 in the main body of the paper).

Recall that, with probability $\geq 1 - \delta$, the behavior of $n - wdp$ on T with prior $\mathbb{P}(\cdot)$ is the same as that of wdp on T_λ with the updated prior $\mathbb{P}_\lambda(\cdot)$. Then we can use Lemma 3 by replacing c^* with $\widehat{c}$ to claim that the number of paths selected by N-WDP before stopping is $\widetilde{K}(T_\lambda, \widehat{c})$, and then Lemma 4 to conclude that the expected (*w.r.t.* $\mathbb{P}(\cdot)$) number of queries made by N-WDP is upper bounded with probability $1 - \delta$ (over the noise in the labels) by

$$\mathcal{O}\left(f(n, \lambda, \delta)\, \mathbb{E}_{\mathbb{P}_\lambda} \widetilde{K}(T_\lambda, \widehat{c}) \log(h(T_\lambda))\right) = \mathcal{O}\left(\frac{\log(n/\delta)}{(1-2\lambda)^2}\, \mathbb{E}_{\mathbb{P}} \widetilde{K}(T, c^*) \log h\right) .$$

We conclude the proof by showing that with probability at least $1 - \delta$ we have $d_H(\Sigma^*, \widehat{\mathcal{C}}) = \mathcal{O}\left(\frac{n(\log(n/\delta))^{3/2}}{(1-2\lambda)^3}\right)$. Since all labels requested by N-WDP are simultaneously correct with probability at least $1-\delta$, the distance $d_H(\Sigma^*, \widehat{\mathcal{C}})$ is upper bounded with the same probability by $\sum_{i \in L(T_\lambda)} |L(i)|^2$. For each tree T_λ constructed by N-WDP , and any $i \in L(T_\lambda)$, we have

$$\mathcal{O}(f(n, \lambda, \delta)) = |L(i)| = \Omega\left(f(n, \lambda, \delta)^{1/2}\right) .$$

Hence, the maximum number of leaves of T_λ is $\mathcal{O}\left(\frac{n}{(f(n,\lambda,\delta))^{1/2}}\right)$, and the quantity $\sum_{i \in L(T_\lambda)} |L(i)|^2$, contributing to $d_H(\Sigma^*, \widehat{\mathcal{C}})$ is upper bounded by

$$\mathcal{O}(n\left(f(n, \lambda, \delta)\right)^{3/2}) = \mathcal{O}\left(\frac{n(\log(n/\delta))^{3/2}}{(1-2\lambda)^3}\right) ,$$

as claimed. □

C Missing material from Section 4

C.1 Proof sketch of Theorem 5

Proof. The proof follows from Theorem 2 and 3 in [6], together with the following observations.

1. For any tree T with n leaves, we have $|\mathbb{C}(T,K)| = \mathcal{O}(n^K)$.
2. When $\mathcal{D}$ is uniform, the disagreement coefficient $\theta = \theta(\mathbb{C}(T,K),\mathcal{D})$ is $\mathcal{O}(K)$. To show this statement, consider the following. For any $c^* \in \mathbb{C}(T,K)$ and $r > 0$, let

$$\textit{DIS}(c^*,r) = \Big\{(x_1,x_2) \in L \times L \,:\, \exists c' \in \mathbb{C}(T,K) \,:\sigma_{\mathcal{C}(c')}(x_1,x_2) \neq \sigma_{\mathcal{C}(c^*)}(x_1,x_2) \\ \wedge\, d_H(\Sigma_{\mathcal{C}(c')},\Sigma_{\mathcal{C}(c^*)}) \leq r\Big\}\,.$$

   Then in our case θ is defined as

$$\theta = \sup_{r>0} \frac{|\{(x_1,x_2) \in \textit{DIS}(c^*,r)\}|}{r\,n^2}\,.$$

   Now, for any budget r in $\textit{DIS}(c^*,r)$, and any $c^* \in \mathbb{C}(T,K)$, the number of times we can replicate the perturbation of c^* so as to obtain c' satisfying $d_H(\Sigma_{\mathcal{C}(c')},\Sigma_{\mathcal{C}(c^*)}) \leq r$ is at most K. This is because any such perturbation will involve a different cluster of $\mathcal{C}(c^*)$, and therefore disjoint sets of leaves. Moreover, each such perturbation covers rn^2 leaves. The worst case that makes $\theta = K$ is when T is a full binary tree, and $\mathcal{C}(c^*)$ has equally-sized clusters. In all other cases $\theta \leq K$.
3. Regarding the expected running time per round, we give the pseudocode (see Algorithm 2 in this appendix) of a sequential algorithm, which operates as follows. In a preliminary phase the input tree T is preprocessed in order to be able to find in *constant time* at any time t **(i)** the leftmost and rightmost descendent leaf of any internal node of T, and **(ii)** the lowest common ancestor of any two given leaves.[6] At each time t, it receives $\langle (x_{i_t},x_{j_t}),\sigma_t,w_t\rangle$, for some weight $w_t \geq 0$, and label $\sigma_t \in \{-1,+1\}$, and outputs $\mathrm{err}_t(\mathcal{C}(\hat{c}_{t+1}))$, based on the past computation of $\mathcal{C}(\hat{c}_t)$ and $\mathrm{err}_{t-1}(\mathcal{C}(\hat{c}_t))$. This can be directly used to compute at each round $\mathrm{err}_{t-1}(\mathcal{C}(\hat{c}_t))$ needed by the algorithm, but also the perturbed cut $\hat{c}'_t$ and its associated empirical error $\mathrm{err}_{t-1}(\mathcal{C}(\hat{c}'_t))$, once we repeat the computation by perturbing the last item $\langle (x_{i_t},x_{j_t}),\sigma_t,w_t\rangle$ in the training set as follows: $\sigma_t = -\sigma_{\mathcal{C}(\hat{c}_t)}(x_{i_t},x_{j_t})$, and $w_t = \infty$. In turn, the above can be used to compute $d_t = \mathrm{err}_{t-1}(\mathcal{C}(\hat{c}'_t)) - \mathrm{err}_{t-1}(\mathcal{C}(\hat{c}_t))$ and probability p_t.

   The cornerstone of this procedure is to maintain updated over time for each internal node v of T a record storing eight values:

   - *1st, 2nd, 3rd and 4th values*: positive and negative inter-cluster total weight of all leaves in $L(\mathrm{left}(v))$ and $L(\mathrm{right}(v))$;
   - *5th and 6th values*: positive and negative inter-cluster sum of weights $w(x_i,x_j)$ for all $x_i \in L(\mathrm{left}(v))$ and all $x_j \in L(\mathrm{left}(v))$, and
   - *7th and 8th values*: total intra-cluster negative weight of all the clusters of leaves in $L(\mathrm{left}(v))$ and $L(\mathrm{right}(v))$.

   When this procedure receives in input triplet $\langle (x_{i_t},x_{j_t}),\sigma_t,w_t\rangle$, it finds $a_t = \mathrm{lca}(x_{i_t},x_{j_t})$. Then the eight records associated with each node on the bottom-up path $\pi(a_t,r)$ are updated in a bottom-up fashion according to the input, whenever necessary. This requires a constant time per node in $V(\pi(a_t,r))$. Finally, $\mathrm{err}_t(\mathcal{C}(\hat{c}_{t+1}))$ is obtained by simply summing the total intra-cluster negative weight of all clusters of leaves in $L(\mathrm{left}(r))$ and $L(\mathrm{right}(r))$ to the total inter-cluster positive weight of all leaves in $L(\mathrm{left}(r))$ and $L(\mathrm{right}(v))$, plus the inter-cluster sum of positive weights of the pairs $w(x_i,x_j)$ for all $x_i \in L(\mathrm{left}(r))$ and $x_j \in L(\mathrm{left}(r))$. In the special case where the updated clustering is made up of a single cluster containing all leaves of T, the procedure outputs the sum of all negative values in the record associated with r. In any event, computing this sum requires constant time.

   Hence the total time required for performing all operations required at any time t is simply $\mathcal{O}(|V(\pi(a_t,r)|)$.

This concludes the proof. □

Finally, in order to compute the clustering at the end of the training phase, it suffices to perform a breadth-first visit of T to find all leaves of T'_{c^*}. This requires a time linear in the number of clusters of the clustering found by the algorithm. Then the algorithm outputs the indices of the leftmost and rightmost descendant of each leaf of T'_{c^*}, which requires $\Theta(1)$ time per cluster. The total time for giving in output the computed clustering is therefore linear in the number of its own clusters.

[6] Note that a_t can always be found in constant time after a $\Theta(n)$ time preprocessing phase of T – see [19].

C.2 Pseudocode of the NR algorithm in the non-realizable setting

Each internal node of T is associated with a record containing eight values that are maintained updated over time. We start by providing the semantics of these eight values:

- $\mathrm{weight}(v, \mathrm{left}, -1)$ and $\mathrm{weight}(v, \mathrm{left}, +1)$: negative and positive inter-cluster total weight of leaves in $L(\mathrm{left}(v))$.
- $\mathrm{weight}(v, \mathrm{middle}, -1)$ and $\mathrm{weight}(v, \mathrm{middle}, +1)$: negative and positive inter-cluster sum of weights $w(\ell_l, \ell_r)$, where $\ell_l \in L(\mathrm{left}(v))$ and $\ell_r \in L(\mathrm{right}(v))$, respectively.
- $\mathrm{weight}(v, \mathrm{right}, -1)$ and $\mathrm{weight}(v, \mathrm{right}, +1)$: negative and positive inter-cluster total weight of leaves in $L(\mathrm{right}(v))$.
- $\mathrm{cost}(v, \mathrm{left})$ and $\mathrm{cost}(v, \mathrm{right})$: intra-cluster total negative weight of clusters of leaves in $L(\mathrm{left}(v))$ and $L(\mathrm{right}(v))$, respectively.

Finally, for any internal node v of T, we denote by $s(v)$ the following sum:

$$\begin{aligned} s(v) \stackrel{\mathrm{def}}{=\!=} & \ \mathrm{weight}(v, \mathrm{left}, -1) + \mathrm{weight}(v, \mathrm{left}, +1) + \mathrm{weight}(v, \mathrm{middle}, -1) \\ & + \mathrm{weight}(v, \mathrm{middle}, +1) + \mathrm{weight}(v, \mathrm{right}, -1) + \mathrm{weight}(v, \mathrm{right}, +1) \ . \end{aligned}$$

Algorithm 2: Sequential algorithm for the non-realizable case (NR).

▷ **INPUT** : Sequence of pairs of labeled leaves of the form $\langle (\ell, \ell'), \sigma(\ell, \ell') \rangle$
▷ **OUTPUT** : Clustering $\mathcal{C}$ with minimum cost over all clusterings realized by T.

Init:

- **for** $v \in V$ **do** **if** $v \in L$ is_cluster$(v) \leftarrow 1$; **else** is_cluster$(v) \leftarrow 0$;
- current_tot_cost $\leftarrow 0$;
- Preprocess T in a bottom-up fashion and store for each internal node of T the leftmost and rightmost leaf descendant index. `/* Necessary to output` $\mathcal{C}$ `in linear time */`
- Preprocess T to find the lowest common ancestor of any pair of leaves in constant time.

for $t = 1$ **to** $\ldots$ **do**
  Receive pair of leaves (ℓ, ℓ');
  $w(\ell, \ell') \leftarrow 0$; `/* initialize` $w(\ell, \ell')$ `*/`
  $a \leftarrow$ lowest common ancestor of ℓ and ℓ'; `/* we assume` $\ell \neq \ell'$ `*/`
  `/* save all records for the rollback that will be done later */`
  $\mathcal{S} \leftarrow$ list of saved records (eight values per node) of the path $\pi(a, r)$;
  `/* ----- verify whether` ℓ `and` ℓ' `are in the same cluster of the current optimal clustering ----- */`
  while $a \neq r \wedge$ is_cluster$(a) = 0$ **do**
    $a \leftarrow$ par(a);
  if is_cluster$(a) = 1$ **then** same_cluster$(\ell, \ell') \leftarrow 1$; **else** same_cluster$(\ell, \ell') \leftarrow 0$;
  `/* ----- compute optimal cost under constraint ----- */`
  if same_cluster$(\ell, \ell') = 1$ **then**
    `/* compute the optimal cost of the current clustering constrained by the assumption that` ℓ `and` ℓ' `are in different clusters;` $-\infty$ `is simulated using a very large negative number */`
    total_modified_cost $\leftarrow$`add_weight`$(\ell, \ell', -\infty)$;
  else
    `/* compute the optimal cost of the current clustering constrained by the assumption that` ℓ `and` ℓ' `are in the same cluster;` $+\infty$ `is simulated using a very large positive number */`
    total_modified_cost $\leftarrow$`add_weight`$(\ell, \ell', +\infty)$;
  `/* rollback of the clustering preceding the add of weight` $-/+\infty$ `*/`
  Restore all records of $\mathcal{S}$;
  `/* ----- add weight` $w(\ell, \ell')$ `if necessary ----- */`
  Set:
    - Difference $d_t \leftarrow \frac{1}{t-1}$ (total_modified_cost $-$ current_tot_cost) ;
    - Probability p_t as a function of d_t as in Eq. (4);
    - $w(\ell, \ell') \leftarrow \frac{\sigma(\ell, \ell')}{p_t}$;
    - With probability p_t, current_tot_cost $\leftarrow$`add_weight`$(\ell, \ell', w(\ell, \ell'))$;

`/* ----- find the current optimal clustering/partition of` L `----- */`
Perform a breadth-first search on T, starting from its root r, to create the set V' formed by all nodes $v \in V$ such that is_cluster$(v) = 1$ and for all ancestors a of v we have is_cluster$(a) = 0$;

$\mathcal{C} \leftarrow \emptyset$;
for $v \in V'$ **do**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{L(v)\}$;
return $\mathcal{C}$.

Procedure Procedure `add_weight`$(\ell, \ell', w(\ell, \ell'))$

▷ **INPUT** : Pair of leaves $\ell, \ell' \in L$ (with $\ell \neq \ell'$) and weight $w(\ell, \ell')$
▷ **OUTPUT** : Total clustering cost after adding weight $w(\ell, \ell')$

$a \leftarrow$ lowest common ancestor of ℓ and ℓ';

/* update middle weight record of node a */
weight$(a, \text{middle}, \text{sgn}(w(\ell, \ell'))) \leftarrow$ weight$(a, \text{middle}, \text{sgn}(w(\ell, \ell'))) + w(\ell, \ell')$;

/* set cluster flag of node a */
if $s(a) \geq 0$ **then**
  is_cluster$(a) \leftarrow 1$;
else
  is_cluster$(a) \leftarrow 0$;

/* ----- record update of all a's ancestors ----- */
while $a \neq r$ **do**
  /* set variable dir to left or right direction from par(a) to a */
  if $a =$ left(par(a)) **then**
    dir $\leftarrow$ left;
  else
    dir $\leftarrow$ right;

  /* update positive and negative inter-cluster weights of node par(a) */
  for $\sigma \in \{+1, -1\}$ **do**
    if is_cluster$(a) = 0$ **then**
      weight$(\text{par}(a), \text{dir}, \sigma) \leftarrow$
       weight$(a, \text{left}, \sigma) +$ weight$(a, \text{middle}, \sigma) +$ weight$(a, \text{right}, \sigma)$;
    else
      weight$(\text{par}(a), \text{dir}, \sigma) \leftarrow 0$;

  /* update par(a)'s cost record relative to node a */
  if is_cluster$(a) = 0$ **then**
    cost$(\text{par}(a), \text{dir}) \leftarrow$ cost$(a, \text{left}) +$ cost(a, right);
  else
    cost$(\text{par}(a), \text{dir}) \leftarrow$ cost$(a, \text{left}) + |\text{weight}(a, \text{left}, -1)| + |\text{weight}(a, \text{middle}, -1)| +$
     $|\text{weight}(a, \text{right}, -1)| +$ cost(a, right);

  /* update cluster flag of par(a) */
  if $s(\text{par}(a)) \geq 0$ **then**
    is_cluster(par$(a)) \leftarrow 1$;
  else
    is_cluster(par$(a)) \leftarrow 0$;
  $a \leftarrow$ par(a);

/* ----- compute the total cost of the current optimal clustering ----- */
if is_cluster$(r) = 0$ **then**
  cost_after_adding_weight $\leftarrow$ cost$(r, \text{left}) +$ weight$(r, \text{left}, +1) +$ weight$(r, \text{middle}, +1) +$
   weight$(r, \text{right}, +1) +$ cost(r, right)
else
  cost_after_adding_weight $\leftarrow$ cost$(r, \text{left}) + |\text{weight}(r, \text{left}, -1)| +$
   $|\text{weight}(r, \text{middle}, -1)| + |\text{weight}(r, \text{right}, -1)| +$ cost(r, right);

return cost_after_adding_weight .

C.3 Missing material from Section 5

In Table 2 we report the results of our preliminary experiments. Notice that N-WDP, NR, and BF are randomized algorithms. Hence, for these three algorithms we give average results and

Tree	No. of queries / Algorithm	250	500	1000	2000	5000	10000	20000
SING	ERM	8.81	8,78	8.39	8.29	8.29	8.29	8.29
	N-WDP	8.29±0.0	8.28±0.0	8.28±0.0	8.29±0.0	–	–	–
	NR	11.0±2.0	8.77±0.0	8.43±0.0	8.31±0.0	8.29±0.0	–	–
	BF	89.0±0.0	89.0±0.0	88.0±0.0	86.0±2.0	87.0±3.0	72.0±10.0	67.0±10.0
MED	ERM	10.30	10.16	9.36	8.91	8.91	8.69	8.65
	N-WDP	9.41±0.1	9.07±0.1	8.88±0.1	8.92±0.1	8.8±0.1	8.8±0.1	8.7±0.1
	NR	10.17±0.0	9.37±0.0	9.0±0.0	8.85±3.0	–	–	–
	BF	89.4±0.0	88.1±0.0	87.0±0.0	63.1±0.0	18.2±5.0	18.0±3.0	10.9±1.0
COMP	ERM	10.65	10.30	10.04	9.26	9.06	8.99	8.93
	N-WDP	9.52±0.0	9.47±0.0	9.44±0.0	9.43±0.0	–	–	–
	NR	10.1±0.0	10.0±0.0	10.0±0.0	11.4±0.6	10.8±0.5	9.0±0.0	8.9±0.0
	BF	13.5±0.0	13.5±0.0	9.2±0.0	9.1±0.0	9.0±0.0	9.0±0.0	8.9±0.0

Table 2: Test error (in percentage) vs. number of queries for the various algorithms we tested on the hierarchies SING, MED, and COMP originating from the MNIST dataset (see main body of the paper). Standard deviations are also reported. Missing values on N-WDP are due to the fact that the algorithm stops before reaching the desired number of labels. Missing values on NR are instead due to the fact that we stopped the algorithm's execution once we obseved no further test error improvement.

standard deviation across 10 independent runs of each one of them. As a reference, consider that the performance of BEST (see Section 5 in the main body of the paper) on the three datasets is the following: SING: 8.26%, MED: 8,51%, COMP: 8.81%. Moreover, since in this dataset we have 10 class labels with approximately the same frequency, both a *random* clustering and a degenerate clustering having $n = 10000$ singletons would roughly give 10% error.

In light of the above, notice that on both SING and MED, the robust breadth-first strategy BF goes completely off trail, in that it tends to produce clusterings with very few clusters. This behavior is due to the presence in the two hierarchies of long paths starting from the root, which is in turn caused by the way the single and the median linkage functions deal with the outliers contained in the MNIST dataset.

Finally, one should take into account the fact that when training our active learning algorithms we have used the first 500 labels for parameter tuning. Hence, a fair comparison to ERM is one that contrasts the test error of N-WDP, NR, and BF at a given number of queries q to the test error of ERM at $q + 500$ queries. From Table 2 one can see that, even with this more careful comparison, N-WDP outperforms ERM. On the other hand, NR looks similar to ERM on MED and COMP, and worse than ERM on SING.