# *It's just a matter of perspective(s)*: Crowd-Powered Consensus Organization of Corpora

Ayush Jain, Joon Young Seo, †Karan Goel, Andrew Kuznetsov
Aditya Parameswaran, Hari Sundaram
University of Illinois, Urbana Champaign, †IIT Delhi
{ajain42, jmseo2, akuznet2, adityagp, hs1}@illinois.edu; †kgoel93@gmail.com

## ABSTRACT

We study the problem of organizing a collection of objects—images, videos—into clusters, using crowdsourcing. This problem is notoriously hard for computers to do automatically, and even with crowd workers, is challenging to orchestrate: *(a)* workers may cluster based on different latent hierarchies or perspectives; *(b)* workers may cluster at different granularities even when clustering using the same perspective; and *(c)* workers may only see a small portion of the objects when deciding how to cluster them (and therefore have limited understanding of the "big picture"). We develop cost-efficient, accurate algorithms for identifying the consensus organization (i.e., the organizing perspective most workers prefer to employ), and incorporate these algorithms into a cost-effective workflow for organizing a collection of objects, termed ORCHESTRA. We compare our algorithms with other algorithms for clustering, on a variety of real-world datasets, and demonstrate that ORCHESTRA organizes items better and at significantly lower costs.

## 1. INTRODUCTION

*"Everything we hear is an opinion, not a fact.*
*Everything we see is a perspective, not the truth."*
— Marcus Aurelius, ca. 150 AD.

With the costs of storage rapidly decreasing, we have been amassing large volumes of images and videos within our personal computers and within shared file systems in organizations. To be able to make effective use of these images and videos, it is essential to *organize* them into clusters. Unfortunately, automated schemes perform poorly at organization since they are not able to interpret or understand content adequately. Human beings, on the other hand, can easily organize such content, but it is often impossible for any single human worker to organize a large corpus. So we turn to crowdsourcing for organizing content.

Unfortunately, employing crowdsourcing is rife with several issues, stemming from the fact that there are often many correct ways of organizing complex content such as images. To illustrate these issues (listed below), we asked 20 workers on Amazon's Mechanical Turk to cluster a stylized set of 25 images, where each image is a random combination of (SHAPE, COLOR, SIZE). Workers were allowed to create as many clusters as they wanted, and populate these clusters with the 25 images. We note that this is a simple experiment—we expect real world corpora to be more complex.

- *Issue 1: Perspectives.* Human workers often organize items using distinct organizational perspectives, rendering the answers or clusters obtained from different workers incomparable, making it hard to combine opinions across workers. For example, in our experiment, 85% of the workers chose to organize by SHAPE, 10% by COLOR, and 5% by SIZE.
- *Issue 2: Granularities.* Even within a single organizational perspective, workers often organize at different "granularities". For instance, for workers that chose to organize based on SHAPE, some chose to create the following clusters: {`Polygons`, `Ellipses`}, while others chose to split the `Polygons` cluster, giving us {`Rectangles`, `Triangles`, `Ellipses`}. Consequently, the number of clusters given by the workers also varied drastically.
- *Issue 3: Limited Understanding of the "Big Picture".* To limit cognitive load, workers can only cluster or organize a small number of items at once, making it hard for them to understand how the small set of items fits in with the rest. For instance, if there were no triangles in the set of 25 items given to a worker, they would organize the items assuming that triangles did not exist in the dataset, while that might not actually be true.

*We focus on the problem of developing a cost-efficient robust workflow to perform consensus organization of large corpora*, one that majority of the workers agree with. In our experiment above, we found that majority of the workers clustered on SHAPE, and that would represent our consensus organizational perspective. Work from behavioral psychology on *free classification* has similarly demonstrated that humans have a tendency to pick a specific (likely) organizational perspective, while at the same time humans do adopt different perspectives [12, 21, 10, 17, 19].

Prior work has considered the problem of crowd clustering [9, 30, 29], falling short in three ways: *(a)* These papers do not take into account the fact that different workers may organize using different perspectives and at different granularities, leading to an organization that is sub-optimal with mixed organizational perspectives. *(b)* Prior work emphasizes the use of random sampling; however in the absence of any relationship between the subsets of samples that the workers see, randomized sampling is costly. Indeed, [9] report in their paper that they require each item to appear in 6 random samples to ensure goodness of clustering, making it impractical in terms of cost. *(c)* These papers transform the clusters provided by workers into votes on the similarity or dissimilarity of pairs of items, losing out on the overall clustering structure. This is because the eventual goal of these papers is to recover pairwise similarity or dissimilarity information, as opposed to finding a consensus organization. Due to these limitations, prior work can only organize items appropriately if there is a single perspective with no variable granularities (which is not true even in our stylized example above and certainly not true in real datasets). Indeed, we find that on real datasets, their results are much worse. We describe related work in more detail in Section 5.

Our workflow, termed ORCHESTRA, instead uses workers to repeatedly organize carefully selected groups of items. Instead of decomposing the responses from workers into pairwise comparisons, we operate on them directly. We develop algorithms to infer not just which organizational perspective a worker is clustering using but also the granularity within. We use these algorithms in conjunction with techniques to identify the maximum likelihood granularity in the maximum likelihood perspective, assembled into a workflow for organization.

There are several challenges in assembling ORCHESTRA. First, ensuring adequate coverage is hard—all clusters need to be well represented, even when individual workers may not see representatives from all clusters. Second, it is not easy to identify if workers are clustering on the same organizational perspective, especially if they are using different granularities, or combining granularities. For instance, a worker may provide triangles, squares, non-polygons as three clusters, while another worker may provide polygons, ellipses, circles as three clusters; both these workers are using different granularities on the same perspective. Third, once we identify that workers are indeed clustering using the same perspective, it is not trivial to combine information across workers. In our example given previously, no two clusters provided by workers are alike, making it challenging to combine information across them. Fourth, combining or relating information across workers is exacerbated by the fact that different workers may be clustering different sets of items; we need to identify common "pivots" that can help us relate clusters across workers on different sets of items. Last, assembling repeated worker clusterings into a cost-effective workflow, while setting the parameters that control the workflow in a principled manner, is yet another challenge.

Here is a list of technical contributions in this paper:

- We model the problem formally using *graph hierarchies* to capture the notion of perspectives, and *frontiers* on the hierarchies to capture the notion of granularities. (Section 2)
- We design, ORCHESTRA, a *robust, low-cost workflow for organization* comprising the following algorithmic components:
  - We develop techniques to map worker clusterings to hierarchies (to identify worker perspectives), and formalize the identification of the consensus or the maximum likelihood hierarchy as a MAX-CLIQUE problem. (Section 3.1)
  - We develop probabilistic techniques to ensure that our maximum likelihood hierarchy has *adequate coverage* of the space of all concepts in the dataset. (Section 3.2)
  - We develop the notion of a *kernel* to relate worker clusterings on different samples of items to the maximum likelihood hierarchy. (Section 3.3)
  - We design techniques to *extend* the current maximum likelihood hierarchy by merging worker responses on new items to the existing hierarchy. (Section 3.4)
  - We develop algorithms that operate *bottom-up* to identify the maximum likelihood frontier on the maximum likelihood hierarchy, which can then be leveraged for *categorization*, providing further savings on cost and improved accuracies. (Section 3.5)
- We further couple these algorithmic contributions with experiments on three real datasets on Amazon's Mechanical Turk (Section 4), and demonstrate that our techniques lead to better quality clusterings, when compared both to prior work in this space, as well more primitive versions of ORCHESTRA.

## 2. PRELIMINARIES

In this section we discuss some essential concepts and ideas. In Section 2.1, we present a sequence of definitions that helps formalize the problem we address in this paper. In Section 2.2, we describe our model for worker behavior and our interfaces, and in Section 2.3, we describe the ORCHESTRA workflow at a high level. Finally, in Section 2.4, we provide a breakdown of the clustering phase of ORCHESTRA that will be our focus in the next section.

### 2.1 Data Model

In this subsection, we provide a series of definitions related to four ideas: clusterings, hierarchies, frontiers and complete frontiers. First, we begin with a formal definition of clustering.

**Definition 2.1 (Clustering).** *Given a set of items $\mathcal{D}$, a clustering is a partitioning of $\mathcal{D}$ into clusters $C_1, \ldots, C_k$ such that,*

*(1)* $C_i \cap C_j = \emptyset$ $\forall\, i \neq j \in \{1, \ldots, k\}$ *(2)* $\bigcup_{i=1}^{k} C_i = \mathcal{D}$

Every cluster in a clustering (and by consequence any set of items) can be associated with an *underlying latent concept*. Intuitively, a concept is a description that is satisfied by each item in a cluster. For example, in Figure 1(f), the clusters—from top to bottom, one corresponding to each row—represent the concepts `Triangles`, `Quadrilaterals` and `Ellipses`. Formally, a concept describes the set of common attributes shared by all items in a cluster. We say that the items in a cluster are *instances* of its latent concept. Anything that holds true for a concept, also holds true for the cluster that it represents.

Concepts may have subset-superset relationships among them. Formally, we say that concept $B$ *generalizes* concept $A$ (denoted $B \succ A$) if every item in $\mathcal{D}$ that is an instance of $A$ is also an instance of $B$. For example, the concept `Quadrilaterals` generalizes `Rectangles`. We introduce the concept `Universe`, which describes any item in the corpus $\mathcal{D}$. By definition, `Universe` generalizes every concept associated with any subset of $\mathcal{D}$.

We can organize concepts based on the *generalize* relationship into a rooted concept tree. We call this concept tree a *hierarchy*.

**Definition 2.2 (Hierarchy).** *For the set of items $\mathcal{D}$, a hierarchy $\mathcal{T}(\mathcal{D})$ is a rooted concept tree where*

*(1) A concept $A \in \mathcal{T}(\mathcal{D})$ is a parent of another concept $B \in \mathcal{T}(\mathcal{D})$ if $A \succ B$ and there exists no $C \in \mathcal{T}(\mathcal{D})$ such that $A \succ C$ and $C \succ B$*

*(2)* `Universe` *is the root node of $\mathcal{T}$*

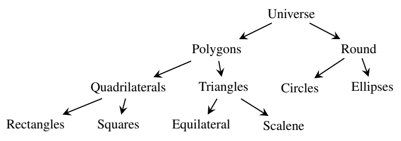
*(3) Every instance of $C \in \mathcal{T}(\mathcal{D})$ is also an instance of exactly one of its children in $\mathcal{T}$*

*(4) For every $C \in \mathcal{T}(\mathcal{D})$, at least one item in $\mathcal{D}$ is an instance of $C$.*

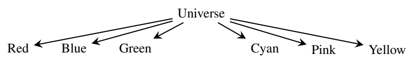
Intuitively, a hierarchy is a concept tree in which every item of $\mathcal{D}$ can be assigned to exactly one of the leaf nodes (and consequently all of its ancestors), and no leaf node is empty. Multiple datasets may have the same hierarchy, and a dataset may be representable by multiple hierarchies.

Note that while a hierarchy is defined in terms of concepts, each concept can be replaced by the cluster that it describes, to get a hierarchy of clusters, built on the subset relation. We will treat these hierarchies as equivalent.

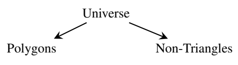
Figure 1 shows some concept trees for the Shapes dataset items shown in Figure 2. Figures 1(a) and 1(b) are hierarchies as every item in the dataset can be assigned to one of the leaf nodes. Other trees, shown in Figure 1(c), 1(d) and 1(e), are not hierarchies. Figure 1(c) is not a hierarchy because the concepts `Polygons` and `Non-Triangles` are not disjoint. Rectangles in the dataset are instances of both concepts and cannot lie in exactly one of them. In 1(d), the concept `Round` does not cover all instances of its parent concept `Non-Triangles`. The dataset has a `Quadrilaterals` concept in addition to `Round`. Figure 1(e) is also not a hierarchy as there are no instances of `Hexagons` in the dataset.



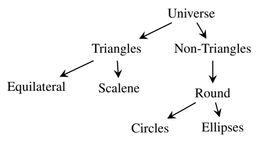
*(a) Hierarchy*



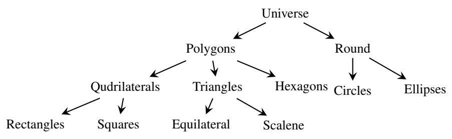
*(b) Hierarchy*



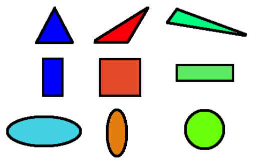
*(c) Not a hierarchy*



*(d) Not a hierarchy*



*(e) Not a hierarchy*



*(f) Shapes*

*Figure 1: (a) – (e): Concept trees for the clustering example shown in Figure 2 — (a) and (b) are hierarchies; (c) is not a hierarchy since it violates (3) in Definition 2.2 — quadrilaterals in the dataset are instances of both children of* `Universe`*; (d) is not a hierarchy since it violates (2) — quadrilaterals in the dataset are instances of* `Non-Triangles` *but not of any children; (e) is not a hierarchy —* `Hexagons` *is a superfluous concept for this dataset. (f) Some examples of items in our Shapes dataset, which we use as a running example in this paper*

We now describe a method to find the hierarchy corresponding to any subset of $\mathcal{D}$, when $\mathcal{T}(\mathcal{D})$ is given. Let there be some set of items $\mathcal{S} \subseteq \mathcal{D}$ associated with $C \in \mathcal{T}(\mathcal{D})$ such that every item in $\mathcal{S}$ is an instance of $C$. $\mathcal{S}$ may or may not contain every instance of $C$. Consider the subtree of $\mathcal{T}(\mathcal{D})$ rooted at $C$. If we enforce condition (2) and (4) in our definition of a hierarchy — replacing $C$ by the `Universe` placeholder, and dropping superfluous concept nodes in this subtree — the resulting tree will be a hierarchy $\mathcal{T}(\mathcal{S})$. For instance, in Figure 1(a), the subtree rooted at `Polygons` is a hierarchy if $S$ is the set of all polygons in the dataset. If $S$ only contains squares and all triangles, then we would remove `Rectangles` as it is now a superfluous concept, and the leftover tree would be a hierarchy. We now define the concept of a frontier.

**Definition 2.3 (Frontier).** *A frontier $F$ is a set of disjoint concepts $\{C_1, \ldots, C_k\}$ in a hierarchy $\mathcal{T}(\mathcal{D})$ such that:*

$$\nexists\, i, j \in \{1, \ldots, k\} : C_i \succ C_j$$

In words, a frontier is a set of disjoint concepts such that no two concepts in a frontier are connected by the *generalizes* relationship. For the hierarchy shown in Figure 1(b), {`Red`, `Green`, `Blue`} forms a valid frontier. Since concepts in $F$ are disjoint, an item in $\mathcal{D}$ can be an instance of *atmost* one concept in $F$.

**Definition 2.4 (Complete Frontier).** *A frontier $F$ in $\mathcal{T}(\mathcal{D})$ is said to be complete if* $\bigcup_{i=1}^{k} C_i =$ `Universe`

In other words, $F$, it is said to be a complete frontier if every item in $\mathcal{D}$ is an instance of *exactly* one concept in $F$. For the hierarchy of Figure 1(b), the frontier {`Red`, `Blue`, `Green`}, when expanded to {`Red`, `Blue`, `Green`, `Cyan`, `Pink`, `Yellow`} becomes complete as every item in the dataset is an instance of exactly one of these concepts. Similarly, for Figure 1(a), {`Polygons`, `Circles`, `Ellipses`}, {`Quadrilaterals`, `Triangles`, `Round`}, {`Rectangles`, `Squares`, `Equilateral`, `Scalene`, `Circles`, `Ellipses`} are all complete frontiers.

Notice the similarities in the definition of clustering and that of a complete frontier. Just as a cluster operationalizes a concept, a clustering can be viewed as an operationalization of a complete frontier on a set of items. Thus, a complete frontier is associated with a clustering of the dataset.

## 2.2 Interacting with Workers

We use two interfaces to interact with workers. The first interface is a *clustering interface*. Here, workers are presented with a carousel of items, which they can drag into as many clusters as they like. This interface allows us to generate partial clusterings for a small set of items. See Figure 2 for an example worker session.

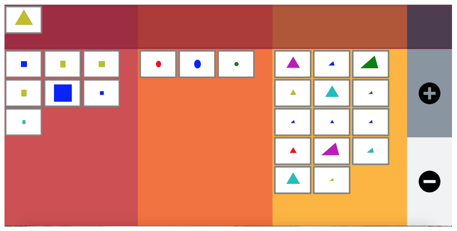


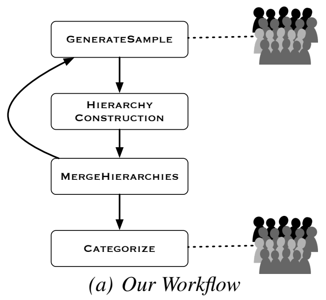
*Figure 2: Our clustering interface. In this example, workers are asked to organize shapes into multiple clusters. They can determine the number of clusters by using the '+' and the '-' buttons seen on the right.*

We model the response to this interface, resulting in a clustering, as a frontier in some latent, underlying hierarchy. Different workers may have completely different latent hierarchies in mind; for instance, Figures 1(a) and 1(b) are both valid hierarchies for the data shown in Figure 2. Thus, the worker clustering process can be modeled as follows. First, given a subset $\mathcal{S} \in \mathcal{D}$, a worker picks some latent hierarchy $\mathcal{T}(\mathcal{S})$. Then, the worker chooses a complete frontier $F$ in $\mathcal{T}(\mathcal{S})$. Notice that while $F$ is complete in $\mathcal{T}(\mathcal{S})$, it will not in general be complete in $\mathcal{T}(\mathcal{D})$. Finally, the output of the worker is the clustering of $\mathcal{S}$ associated with $F$.

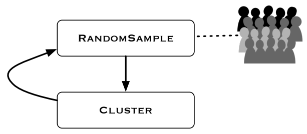
We also use a *categorization interface*, which is similar to the clustering interface except that a fixed number of clusters are shown, and each cluster is pre-populated with a fixed set of items. Workers are asked to drag the new items into one of these existing clusters, thereby categorizing them. In this case, workers no longer have the freedom to select their own latent hierarchy for organization and must instead use the clustering already provided.

## 2.3 Overall Workflow for ORCHESTRA

Our overall workflow comprises of two phases: the clustering phase and the categorization phase. The clustering phase discovers a consensus organization of the data using just a small fraction of items from the corpus. Once the consensus set of clusters are determined, most of the items are then organized in the categorization phase, where we place items into clusters with which they share greatest similarity. Unlike previous work [9, 29], we *don't* make workers cluster every item in the dataset, which allows us to cut costs significantly. Also unlike previous work, *we do not randomly sample* items in each iteration. Instead, we systematically pick some items that are already part of the hierarchy, so that new clusterings can be easily integrated into it. See Figures 3(a) and 3(b) for a graphical comparison between ORCHESTRA and prior work. In Figure 3(a), the first three boxes refer to the clustering phase, while



*(a) Our Workflow*



*(b) Prior Work Workflow*

*Figure 3: Comparison of Workflows*

the last one refers to the categorization phase.

The categorization phase is straightforward, with the only goal being to categorize the remaining items in the dataset; categorization will be applied to the majority of the items. The transition from the clustering to the categorization phase will depend on the dataset complexity. Our primary focus will be the clustering phase; we describe how it is broken down, next.

## 2.4 Clustering Phase for ORCHESTRA

Given a dataset $\mathcal{D}$, the goal of the clustering phase is to recover the maximum likelihood latent hierarchy $\mathcal{T}_{ML}(\mathcal{D})$. This hierarchy has maximum likelihood in the sense that a worker clustering the entire dataset would pick $\mathcal{T}_{ML}(\mathcal{D})$ as the latent organizational hierarchy with the highest probability.

We need to generate $\mathcal{T}_{ML}(\mathcal{D})$ across multiple samples of the dataset. This is because in any realistic setting with large datasets, workers will cluster a dataset $\mathcal{S}$ where $\mathcal{S} \subset \mathcal{D}$, and indeed, in general it is likely that $|\mathcal{S}| \ll |\mathcal{D}|$. Given this, we must find $\mathcal{T}_{ML}(\mathcal{D})$ by generating multiple samples and aggregating worker responses across them.

To find $\mathcal{T}_{ML}(\mathcal{D})$, ORCHESTRA has an iterative refinement procedure that performs repeated iterations of (GENERATESAMPLE $\rightarrow$ CONSTRUCTHIERARCHY $\rightarrow$ MERGEHIERARCHIES $\rightarrow \ldots$), to generate a final hierarchy. At the end of each iteration, we generate a new estimate for $\mathcal{T}_{ML}(\mathcal{D})$. We give an intuitive explanation for these algorithms below; a detailed description is given in the next section.

- GENERATESAMPLE. Any sample of items that we generate must contain some item overlap with previously generated samples, as well as contain new items that explore the dataset. The overlap helps us locate worker frontiers on this sample within the current estimate of $\mathcal{T}_{ML}(\mathcal{D})$, while the new items allow us to expand $\mathcal{T}_{ML}(\mathcal{D})$ by finding new concepts. We provide a procedure to check if two workers—working on different samples—are providing frontiers on the same latent hierarchy.
- HIERARCHYCONSTRUCTION. The construction algorithm takes as input multiple worker frontiers collected for a single sample, and outputs the dominant hierarchy. To separate the dominant hierarchy, HIERARCHYCONSTRUCTION infers whether these frontiers are chosen from the same hierarchy, or different ones.
- MERGINGHIERARCHIES. To combine hierarchies across multiple samples, the merging algorithm takes as input two hierarchies — the current estimate of $\mathcal{T}_{ML}(\mathcal{D})$, and the hierarchy constructed on the current sample. The output is a new estimate of $\mathcal{T}_{ML}(\mathcal{D})$, and is calculated by augmenting the current estimate of $\mathcal{T}_{ML}(\mathcal{D})$. The merging exploits the location of the overlap items in the current estimate of $\mathcal{T}_{ML}(\mathcal{D})$.

At the end of this iterative procedure, we return the maximum likelihood frontier in $\mathcal{T}_{ML}(\mathcal{D})$ as the consensus clustering. The quality of the consensus clustering depends on whether the number of iterations were sufficient to ensure that most items in $\mathcal{D}$ can be categorized into this consensus clustering. In the next section we provide the details of the workflow for ORCHESTRA — including a sampling guarantee that gives a lower bound on the size of the samples needed to cover *atleast* some fraction of items in $\mathcal{D}$.

## 3. ORCHESTRA WORKFLOW

As noted in the previous section, workers may choose different frontiers in different latent hierarchies when asked to cluster a set of items. The problem of finding the maximum likelihood hierarchy is then equivalent to finding a hierarchy that *best explains* the most worker clusterings. In this section, we provide algorithms to find this hierarchy, as well as the consensus clustering within that hierarchy. We also give theoretical results that allow us to limit the number of iterations in the ORCHESTRA workflow. First, in Section 3.1, we describe the HIERARCHYCONSTRUCTION algorithm that finds the most likely hierarchy under the assumption that all workers cluster the same set of $n$ items. Then, in Section 3.2, we provide a guarantee that helps us fix a reasonable value for $n$. Section 3.3 lays out the GENERATESAMPLE algorithm, and in Section 3.4, we generalize our setting with the MERGINGHIERARCHIES algorithm, allowing workers to cluster different subsets of items, and aggregating their clusterings to get a single hierarchy. Finally, in Section 3.5, we present a procedure to find the consensus clustering from the final hierarchy that our iterative workflow generates, as well as describing how we categorize items.

### 3.1 The HIERARCHYCONSTRUCTION Algorithm

Given a set of items $\mathcal{S} = \{x_1, \ldots, x_n\} \subseteq \mathcal{D}$, we ask $m$ workers to cluster the items in $\mathcal{S}$. We denote the set of worker clusterings by $\mathfrak{C} = \{\mathbb{C}_1, \ldots, \mathbb{C}_m\}$, where $\mathbb{C}_i = \{C_{i,1}, ..., C_{i,k_i}\}$ is the set of clusters proposed by worker $i$. Note that workers can give as many clusters as they like, but no cluster is allowed to be empty. Figure 4(a) shows some clusterings proposed by workers on the sample of items shown in Figure 2.

**Problem 3.1 (Hierarchy Construction).** *Given the clusterings $\mathfrak{C}$ on a set of items $\mathcal{S}$, find a hierarchy $\mathcal{T}(\mathcal{S})$ such that the number of clusterings from $\mathfrak{C}$ that can be associated with complete frontiers in $\mathcal{T}(\mathcal{S})$ is maximum.*

Intuitively, we would like to find the maximum likelihood hierarchy, *i.e.*, one that contains the maximum number of clusterings as complete frontiers. For instance, clustering 5 in Figure 4(a) can be associated with Figure 4(b) as a complete frontier, covering all items in the dataset.

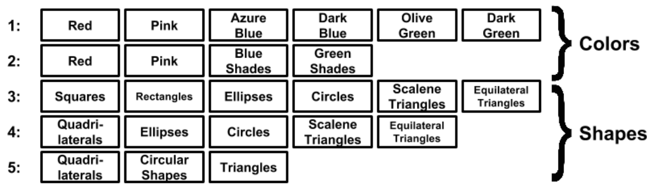
We will show that Problem 3.1 is equivalent to the MAX-CLIQUE problem. MAX-CLIQUE refers to the problem of finding the maximum sized clique in a graph $G$, and is a well-known NP-HARD problem. Consequently, the optimal solution takes exponential time to compute. However, in our case, the graph for which MAX-CLIQUE must be solved is small, so the computation is still feasible. We will prove the equivalence to MAX-CLIQUE via a constructive proof. We first provide some definitions that will help us carry out the construction.

**Definition 3.1 (Consistency of Clusterings).** *Clusterings $\mathbb{C}_i = \{C_{i,1}, \ldots, C_{i,k_i}\}$ and $\mathbb{C}_j = \{C_{j,1}, \ldots, C_{j,k_j}\}$ are said to be consistent if and only if for every $(s,t) \in \{1, \ldots, k_i\} \times \{1, \ldots, k_j\}$, one of the following holds:*

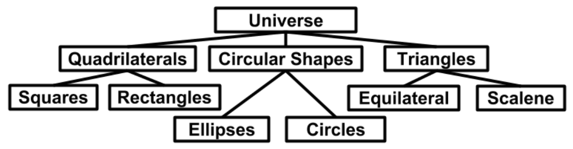
*(1)* $C_{i,s} \cap C_{j,t} = \phi$ *(3)* $C_{i,s} \supset C_{j,t}$

*(2)* $C_{i,s} \subset C_{j,t}$ *(4)* $C_{i,s} = C_{j,t}$

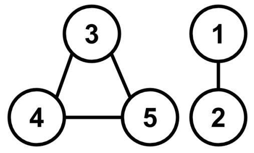
In Figure 4(a), the worker clusterings 1 & 2 are consistent — `Blue Shades` decomposes perfectly into `Azure Blue` and `Dark Blue`, as does `Green Shades` — while 1 is inconsistent with 3, 4, 5. Since every clustering is associated with a frontier, we can also define a



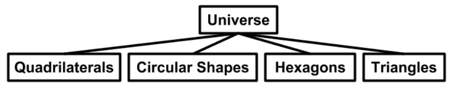
*(a) Examples of real worker clusterings for the dataset in Figure 2.*



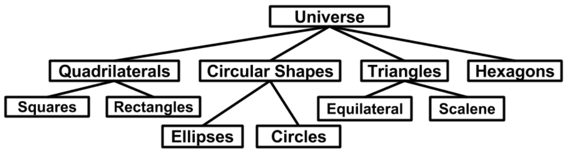
*(b) The hierarchy $\mathcal{T}$ corresponding to the maximum sized clique $3, 4, 5$ in (c) using* ConstructHierarchy.



*(c) The clustering graph for the worker clusterings shown in (a).*



*(d) A hypothetical hierarchy $\mathcal{T}(\mathcal{S})$ constructed in the 2nd iteration of our workflow, which contains an extra* `Hexagons` *concept.*



*(e) The hierarchy $\mathcal{T}'$ constructed by merging (b) and (d) using* MergingHierarchies *after 2 iterations.*

*Figure 4: An example demonstrating our iterative workflow approach on the Shapes dataset of Figure 1(f).*

corresponding notion of *consistent frontiers*: we simply replace $\supset$ by $\succ$ in Definition 3.1. It is useful to note that any two complete frontiers in the same hierarchy will always be consistent. In Figure 1(a), the complete frontiers {`Quadrilaterals`, `Triangles`, `Ellipses`, `Circles`} and {`Squares`, `Rectangles`, `Triangles`, `Round`} are consistent.

**Definition 3.2 (Clustering Graph).** *Clustering graph $G_{\mathfrak{C}} = (\mathfrak{C}, E)$ is an undirected graph, where each clustering in $\mathfrak{C}$ corresponds to a unique vertex in $G$ and there is an edge between $\mathbb{C}_i$ and $\mathbb{C}_j$ $\forall\, i, j \in \{1, \ldots, m\}$ if and only if $\mathbb{C}_i$ and $\mathbb{C}_j$ are consistent.*

Figure 4(b) depicts the clustering graph for the clusterings shown in Figure 4(a). Each worker clustering corresponds to a node in the graph. Notice how there is no edge from 1 to any of 3, 4, 5, since they are mutually inconsistent.

Let $\mathfrak{C}_{\text{CLIQUE}} \subseteq \mathfrak{C}$ be a clique in $G_{\mathfrak{C}}$. Let the set of all *unique* clusters in $\mathfrak{C}_{\text{CLIQUE}}$ be $\mathcal{H} = \{C_{i,j} \mid C_{i,j} \in \mathbb{C}_i, \forall \mathbb{C}_i \in \mathfrak{C}_{\text{CLIQUE}}\}$. $\mathcal{H}$ can be organized into a hierarchy $\mathcal{T}_{\mathcal{H}}$ as follows: for every cluster $C_{i,j} \in \mathcal{H}$, find the smallest cluster in $\mathcal{H} \cup$ `Universe` that is a superset of $C_{i,j}$ and mark that as the parent of $C_{i,j}$ in $\mathcal{T}_{\mathcal{H}}$. Algorithm 1 shows the pseudocode for this HierarchyConstruction algorithm.

Consider the clique 3, 4, 5 in the clustering graph of Figure 4(b). $\mathcal{H}$ contains a total of 14 clusters as shown in Figure 4(a). Suppose we wanted to find the parent of `Rectangles`; the smallest cluster in $\mathcal{H} \cup$ `Universe` containing `Rectangles` is `Quadrilaterals`. The cluster `Universe` also contains `Rectangles` but it is not the smallest such cluster. Thus, we make `Quadrilaterals` the parent of `Rectangles`, as shown in Figure 4(c). Similarly, `Universe` becomes the parent of `Quadrilaterals`. The hierarchy after this construction is shown in Figure 4(c).

We state the following lemma and theorem which show that our construction is valid, and omit the proof. As mentioned earlier, all proofs can be found in our extended technical report [13].

**Lemma 3.1.** *For any $C_{i,j} \in \mathcal{H}$, the smallest cluster in $\mathcal{H} \cup$* `Universe` *that is a superset of $C_{i,j}$, is unique.*

*Proof.* Since `Universe` is the superset of all clusters in $\mathcal{H}$, every $C_{i,j}$ has at least one superset in $\mathcal{H} \cup$ `Universe`. Assume that there are two distinct smallest clusters $C_{1,x}$ and $C_{2,y}$ that are both supersets of $C_{i,j}$. It follows that the clusterings to which $C_{1,x}$ and $C_{2,y}$ belong *i.e.* $\mathbb{C}_1$ and $\mathbb{C}_2$ cannot be consistent. This can be seen by noting that $C_{1,x}$ and $C_{2,y}$ do not satisfy any of the four conditions of Definition 3.1. This contradicts the fact that $\mathbb{C}_1$ and $\mathbb{C}_2$ are part of the same clique in the clustering graph, and the result follows. ☐

**Algorithm 1** `HierarchyConstruction`$(\mathcal{H})$

```
Require: Set of clusters $\mathcal{H}$
Ensure: Hierarchy $\mathcal{T}_{\mathcal{H}}$
  $\mathcal{T}_{\mathcal{H}}(\mathcal{V}) \leftarrow \{$Universe$\} \cup \mathcal{H}$
  for each $H_i \in \mathcal{H}$ do
    $P \leftarrow$ smallest sized $H_j \in \mathcal{H}$ that is superset of $H_i$
    if $P$ is null then
      Parent$(H_i) \leftarrow$ Universe
    else
      Parent$(H_i) \leftarrow P$
    end if
  end for
```

**Theorem 3.1.** *$\mathcal{T}_{\mathcal{H}}$ is a hierarchy.*

*Proof.* By Lemma 3.1, it is easy to see that $\mathcal{T}_{\mathcal{H}}$ is a tree with `Universe` as its root. Let $C$ be a cluster in $\mathcal{T}_{\mathcal{H}}$, and denote by $\{C_1, \ldots, C_k\}$ the children of $C$ in $\mathcal{T}_{\mathcal{H}}$. To prove that $\mathcal{T}_{\mathcal{H}}$ is a hierarchy, we must show that for every such $C$, (i) $C_i \cap C_j = \phi \;\forall\, i \neq j \in \{1, \ldots, k\}$ and (ii) $\bigcup_{i=1}^{k} C_i = C$.

For (i), 2 cases arise: either $C_i$ and $C_j$ are both from the same clustering and are disjoint by definition, or they come from different clusterings, in which case their corresponding clusterings would not be consistent if $C_i \cap C_j \neq \phi$.

For (ii), we know that $\bigcup_{i=1}^{k} C_i \subseteq C$ by construction. Now suppose $\bigcup_{i=1}^{k} C_i \neq C$ and let $X = C \setminus \bigcup_{i=1}^{k} C_i$. Items in $X$ are not see in any child of $C$.

Let $\mathbb{C}_1, \ldots, \mathbb{C}_k$ be clusterings that contain $C_1, \ldots, C_k$ respectively. Each $\mathbb{C}_i$ contains atleast $C_i$. $C$ cannot be in any $\mathbb{C}_i$, since that $\mathbb{C}_i$ would no longer remain disjoint. Every $\mathbb{C}_i$ is a clustering on $\mathcal{S}$ and therefore cluster all items in $X$.

For every $\mathbb{C}_i$, items in $X$ cannot lie in $C_i$ and must lie in other clusters that are not children of $C$. For any item $x \in X$, consider the largest cluster $C_{\text{large}}$ that contains $x$ across $\mathbb{C}_1, \ldots, \mathbb{C}_k$. Since $C_{\text{large}}$ is the largest such cluster, its parent — from our construction — cannot lie in $\mathbb{C}_1, \ldots, \mathbb{C}_k$. It is easy to see that the smallest sized cluster that contains it must be $C$. Therefore, $C_{\text{large}}$ is a child of $C$ which leads us to a contradiction. ☐

Suppose we pick $\mathfrak{C}_{\text{CLIQUE}}$ to be the maximum sized clique in $G_{\mathfrak{C}}$, and let $\mathcal{T}_{\max}$ be the hierarchy that is generated using this clique. We now state an important result.

**Theorem 3.2.** *Suppose every clustering $\mathbb{C} \in \mathfrak{C}$ lies in exactly one maximal clique. Also suppose the total number of latent hierarchies is $k$. Then, $\mathcal{T}_{\max}$ is the maximum-likelihood hierarchy with probability atleast $\left[1 - \left(1 - \frac{1}{k}\right)^m\right]$, where $m$ is the number of workers.*

*Proof.* First assume that the maximum likelihood hierarchy has not gone undiscovered. Notice that every maximal clique in $G_{\mathfrak{C}}$ will correspond to a single, distinct hierarchy. Denote by $M_1, \ldots, M_k$, the maximal cliques in $G_{\mathfrak{C}}$, where $M_i \subseteq \mathfrak{C}$. Let $\mathcal{T}_1, \ldots, \mathcal{T}_k$ be the set of hierarchies corresponding to these maximal cliques *i.e.* $\mathcal{T}_i$ corresponds to $M_i$.

Since every clustering lies in exactly one maximal clique $M_i$, each worker's clustering can be viewed as a vote for that hierarchy $\mathcal{T}_i$. We can then define a multinomial distribution $(m, p_1, \ldots, p_k)$ that captures worker tendency to pick a particular latent hierarchy - $p_i$ is the probability that a worker picks the hierarchy $\mathcal{T}_i$.

The likelihood function corresponding to the $m$ trials (clusterings) can be written as,

$$\mathcal{L} \propto p_1^{|M_1|} p_2^{|M_2|} \ldots p_k^{|M_k|}$$

It is easy to show that the maximum likelihood solution is simply $p_i = \frac{|M_i|}{m}$. Since $\mathcal{T}_{\max}$ corresponds to the maximum clique, its prior probability will be maximum *i.e.* workers have greatest tendency to pick this hierarchy.

Now on the contrary assume that the maximum likelihood hierarchy has gone undiscovered. Since there are $k$ latent hierarchies, the maximum likelihood hierarchy must have probability atleast $\frac{1}{k}$ of being discovered, since otherwise it could not be the maximum likelihood hierarchy. The probability that the maximum likelihood hierarchy goes undiscovered after $m$ worker responses is upper bounded by $(1 - \frac{1}{k})^m$, and the result follows. $\square$

Figure 4(c) shows the maximum likelihood hierarchy corresponding to the maximal clique 3, 4, 5 in the clustering graph of Figure 4(b). Notice that this hierarchy corresponds to organizing the dataset on SHAPE.

Identifying the most probable hierarchy is thus equivalent to finding the maximum sized clique in $G_{\mathfrak{C}}$. The size of $G_{\mathfrak{C}}$ is atmost the number of workers $m$, since we can combine identical worker clusterings into a single node. Since $m$ is typically small ($\leq 15$), solving MAX-CLIQUE is quite tractable.

We note that we do not provide an explicit mechanism for worker mistakes. In our experiments, we find that workers made no errors with respect to the organization they had in mind. Even if some workers do make errors, our maximum likelihood hierarchy only contains those workers who clustered items consistently, and so either these errors would not be incorporated, or a large number of workers would have to make these errors in the same way, which is unlikely. In the future, we plan to relax our definition of consistency of clusterings, to admit a small tolerance threshold.

## 3.2 Sampling Guarantee

As discussed in Section 2, our approach is to first construct a maximum likelihood hierarchy using several samples and then categorize the remaining items into the maximum likelihood frontier in this hierarchy. Recall that for $\mathcal{S} \subseteq \mathcal{D}$, complete frontiers in a hierarchy $\mathcal{T}(\mathcal{S})$ are not generally complete in $\mathcal{T}(\mathcal{D})$. This is because some concepts may have instances in $\mathcal{D}$, but not in $\mathcal{S}$. For instance, suppose $\mathcal{D}$ contains instances of `Ellipses` and `Circles` while $\mathcal{S}$ only contains items from `Ellipses`, but no instance of `Circles`. Then, $\mathcal{T}(\mathcal{S})$ would not contain the `Circles` concept. `Circles` would therefore go *undiscovered* in our sample. We now prove a guarantee that allows us to make a suitable choice for the parameter $n$; $|\mathcal{S}| = n$.

Intuitively, if the size of $\mathcal{S}$ is large, we can be confident that the sample will *discover* the concepts that occur frequently in the dataset, *i.e.,* the concepts that have many instances in the dataset. On the contrary, when $\mathcal{S}$ is small, a large number of concepts are likely to go undiscovered. Thus the hierarchy constructed on a small sample may not be representative of the entire dataset. We formalize the notion of *representativeness* below.

**Definition 3.3 (Concept Coverage).** *The coverage of a concept $C$ is the fraction of items in $\mathcal{D}$ that are instances of $C$.*

We say that a sample $\mathcal{S}$ *discovers* $C$ if and only if there is an item $s \in \mathcal{S}$ that is an instance of $C$.

**Definition 3.4 (Frontier Coverage).** *The coverage of a frontier $F$ with respect to a sample $\mathcal{S}$ is the sum of coverages of the concepts in $F$ that $\mathcal{S}$ discovers.*

Suppose that $\mathcal{S}$ contains $n$ randomly sampled items. Let $\mathcal{T}(\mathcal{D})$ be a hierarchy constructed on $\mathcal{D}$ and let $F$ be a complete frontier in $\mathcal{T}(\mathcal{D})$ containing $f$ concepts. We give a lower bound on the expected coverage of $F$ with respect to $\mathcal{S}$, $\mathbb{E}_{\mathcal{S}}[X_F]$ below.

**Theorem 3.3.** $\mathbb{E}_{\mathcal{S}}[X_F] \geq 1 - \frac{f}{n+1}(1 - \frac{1}{n+1})^n$

*Proof.* Let $p_i$ be the coverage of the $i^{th}$ concept in $F$. Let $X_{F,i}$ be a random variable that equals 0 if $\mathcal{S}$ does not discover the $i^{th}$ concept of $F$ and equals $p_i$ otherwise. Using Definition 3.4, the coverage of $F$ is $X_F = \sum_{i=1}^{f} X_{F,i}$. We have,

$$\begin{aligned}\mathbb{E}[X_F] &= \sum_{i=1}^{f} \mathbb{E}[X_{F,i}] = \sum_{i=1}^{f} p_i(1 - (1 - p_i)^n) \\ &= 1 - \sum_{i=1}^{f} p_i(1 - p_i)^n \geq 1 - \max_{\substack{0 \leq p_i \leq 1 \\ \sum_i p_i = 1}} \sum_{i=1}^{f} p_i(1 - p_i)^n \\ &\geq 1 - \max_{0 \leq p_i \leq 1} \sum_{i=1}^{f} p_i(1 - p_i)^n\end{aligned}$$

Now, for $p_i \in [0, 1]$, $p_i(1 - p_i)^n$ is maximum when $p_i = \frac{1}{n+1}$ and the result follows. $\square$

In Figure 5(a), we plot $n$ vs. $f$ for different values of a threshold $\delta$, which lower bounds the expected coverage. Observe that for fixed values of $\delta$, $n$ increases linearly with $f$. However, this lower bound is not tight and the actual coverage turns out to be greater than $\delta$. To observe this, we plot $\delta$ vs. the actual coverages that we get in Figure 5(b). For each value of $\delta$, we pick 20 $(f, n)$ pairs that satisfy the bound and conduct 1000 random trials for each pair. Each trial consists of assigning a random probability distribution to $f$ bins, which gives the probability of an item in the dataset belonging to a bin (concept). We then draw $n$ samples from this distribution, and compute the actual coverage that we get.

We can similarly find an upper bound for the variance of $X_F$.

**Theorem 3.4.** $\mathrm{Var}_{\mathcal{S}}[X_F] \leq 1 - \left(1 - \frac{f}{n+1}\left(1 - \frac{1}{n+1}\right)^n\right)^2$

As a rule of thumb, we pick $\delta = 0.95$ and $f = 16$ for the experiments that we carry out, which results in $n = 115$. For $n = 115$ and $f = 16$, the variance is upper bounded by 0.1. However, we note that we cannot expect a single worker to be able to organize 115 items at once, so we describe how to iteratively cluster smaller sets of items while being able to combine the information across these iterations in the next section.

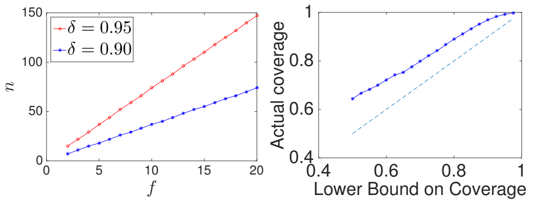


*Figure 5: (a) Plot showing sample size $n$ vs. size of complete frontier $f$ to ensure $\delta$ expected coverage. (b) Plot showing lower bound on expected coverage vs. actual expected coverage over 1000 trials – the dotted line is the* $45\deg$ *line.*

## 3.3 The GENERATESAMPLE Algorithm

As we noted earlier, it is not possible for a single worker to generate a clustering for large $\mathcal{S}$, especially one as large as $\approx$ 120. Instead, our approach will be to repeatedly instantiate smaller $\mathcal{S}$ for every iteration, while bounding its size. For example, one approach would be to instantiate four distinct sets $\mathcal{S}$ of size 30 each, each of which is organized by workers. However, due to the lack of overlap across these sets, it is impossible to relate the hierarchies constructed across these sets to one another. Intuitively, for each new iteration, it is desirable that $\mathcal{S}$ contains some item overlap with the current estimate of the maximum likelihood hierarchy (at the end of the previous iteration), so that the new hierarchy we generate can be easily merged into the current estimate of the maximum likelihood hierarchy. We now provide a mechanism to fix the size of this overlapping set, which we call the *kernel* of $\mathcal{S}$. Each sample $\mathcal{S}$ consists of some new items, not encountered before, along with items that have already been organized in the current maximum likelihood hierarchy, which constitute $K$, the kernel of $\mathcal{S}$.

To set a reasonable value for the kernel $K$, we need it to be large enough to allow us to perform hierarchy merging at each iteration. It also needs to be small enough to allow introduction of some new items into our sample. Our strategy for picking the kernel items is to pick a single item from each of the leaf nodes in the current hierarchy estimate. Therefore, we set $|K| =$ # of leaves and randomly sample the rest of the items in $\mathcal{S}$ from $\mathcal{D}$.

The justification for this strategy is that sampling a single item from every leaf allows us to determine any concept a worker generates — whether an internal node in the hierarchy, or a leaf in our current maximum likelihood hierarchy. If a worker combines some kernel items into a single cluster, we can infer the concept of the entire cluster (which includes some new items) by finding where these kernel items occur together in our hierarchy. We will make this idea more precise in the MERGINGHIERARCHIES algorithm. Algorithm 2 shows the pseudocode for the GENERATESAMPLE algorithm.

**Algorithm 2** `GenerateSample`$(|K|, \mathcal{T}, h)$

```
Require: Kernel size $|K|$, current hierarchy $\mathcal{T}$, sample size $h$
  $\mathcal{S} \leftarrow \{\}$
  for each leaf node $C \in F$ do
    $\mathcal{S} \leftarrow \mathcal{S} \cup$ random item from $C$
  end for
  $\mathcal{S} \leftarrow \mathcal{S} \cup$ ($h$ random items from $\mathcal{D}$ not in $\mathcal{T}$)
  return $\mathcal{S}$
```

In our experiments, we have never encountered a case where $|K|$ is too large: nevertheless, in such cases, we can simply split $|K|$ up into equal sized smaller portions, and repeat the clustering of the same set of new items with these smaller portions of the kernel, such that each of the new items gets the opportunity to be associated with or clustered with any of the kernel items.

## 3.4 The MERGINGHIERARCHIES Algorithm

In this subsection, we describe our MERGINGHIERARCHIES algorithm, in which we make use of the kernel formulation that we introduced above.

Let $\mathcal{T}$ be our current estimate of the maximum likelihood hierarchy generated after the $\tau^{th}$ iteration. Suppose that we run HIERARCHYCONSTRUCTION on the sample $\mathcal{S}$ generated for the $(\tau+1)^{th}$ iteration, and get a hierarchy $\mathcal{T}(\mathcal{S})$. Using $K$, the kernel of $\mathcal{S}$, we would like to merge $\mathcal{T}(\mathcal{S})$ into $\mathcal{T}$ to generate a new hierarchy $\mathcal{T}'$, by mapping known concepts across these hierarchies. To carry out this merging, we will assume that the kernel items in a cluster represent that cluster's concept accurately, as well as any super-concepts (i.e., concepts that are ancestors of the cluster's concept).

Consider the set of leaf nodes $C_1, \ldots, C_l$ in $\mathcal{T}(\mathcal{S})$ and let the set of kernel items in $C_i$ be $K_i$. For each leaf $C_i$:

• Suppose $|K_i| > 0$ for $C_i$; we map $C_i$ to the node $C$ in $\mathcal{T}$ that contains the smallest superset of $K_i$. This is simply the lowest common ancestor of the leaf nodes in $\mathcal{T}$ that contain kernel items from $K_i$. Intuitively, if the kernel items are identical then both clusters are associated with the same concept, and can therefore be merged. All items in $C_i$ are transferred to $C$.

• Suppose $|K_i| = 0$ for $C_i$; we first find the ancestor $C_a$ (with kernel $K_a$) closest to $C_i$ (the lowest ancestor) in $\mathcal{T}(\mathcal{S})$ such that $|K_a| > 0$. Since $C_i$ contained no kernel items, it is clear that $C_i$ represents some new concept; we must search for another concept that *generalizes* $C_i$. As before, we map $C_a$ to the node $C$ in $\mathcal{T}$ that contains the smallest superset of $K_a$. However, since we need to map $C_i$ and not $C_a$, we instead insert $C_i$ as a new child of $C$ in $\mathcal{T}$.

After mapping all leaf nodes in $\mathcal{T}(\mathcal{S})$ to $\mathcal{T}$, we get a new maximum likelihood hierarchy $\mathcal{T}'$ after the $(\tau+1)$th iteration. It is easy to see that $\mathcal{T}'$ is indeed a hierarchy. The only changes we make are *(a)* adding in new items to the concept of which they are instances, which does not modify the hierarchy and *(b)* adding in new concept nodes. For *(b)*, notice that by construction, we attach the new concept $C_i$ to the lowest concept $C$ that generalizes it. $C_i$ is disjoint with respect to all other children of $C$, otherwise it would contain a kernel item. $C_i$ is also necessary to allow all items to exist at the leaf nodes, since no other child of $C$ covers the concept discovered in $C_i$. Therefore, $\mathcal{T}'$ is a hierarchy.

Figure 4 demonstrates an example of MERGINGHIERARCHIES. Figure 4(b) is our current estimate $\mathcal{T}$ to be merged with Figure 4(d), depicting $\mathcal{T}(\mathcal{S})$. The merged hierarchy $\mathcal{T}'$ is shown in Figure 4(e). By our GENERATESAMPLE algorithm, the kernel of $\mathcal{S}$ would contain 6 items, one each for the leaves of $\mathcal{T}$ in Figure 4(b). Even though $\mathcal{T}(\mathcal{S})$ combines the kernel items corresponding to `Squares` and `Rectangles` into the `Quadrilaterals` cluster in Figure 4(e), we can map `Quadrilaterals` in $\mathcal{T}(\mathcal{S})$ using these 2 kernel items, to the `Quadrilaterals` cluster in $\mathcal{T}$, the lowest node where they occur together. For the `Hexagons` cluster in $\mathcal{T}(\mathcal{S})$, we first find its deepest ancestor in $\mathcal{T}(\mathcal{S})$ that contains a kernel item, which turns out to be `Universe`. `Universe` in $\mathcal{T}(\mathcal{S})$ is mapped to `Universe` in $\mathcal{T}$, and `Hexagons` is inserted as a child, as shown in Figure 4(e).

We now conclude this section with a discussion of the cost of our iterative workflow.

**Cost of Iterative Workflow.** Our sampling guarantee requires us to sample $n$ items, while the kernel overlap is fixed to be the # of leaves in the current hierarchy estimate. Notice that when fixing the value of $n$, we assumed that the size of a complete frontier in the dataset hierarchy is $f$. We do not expect our choice of $n$ to discover *any more than* an $f$-sized complete frontier. We can therefore upper-bound the value of $|K|$, the size of the kernel, to be $f$, since we would not expect the number of leaves in our maximum

**Algorithm 3** MergingHierarchies($\mathcal{T}, \mathcal{T}(\mathcal{S})$)

```
Require: current hierarchy estimate $\mathcal{T}$, generated hierarchy $\mathcal{T}(\mathcal{S})$
Ensure: Hierarchy $\mathcal{T}'$
  for each leaf node $C_i \in \mathcal{T}(\mathcal{S})$ do
    $K_i \leftarrow$ kernel items in $C_i$
    $F \leftarrow \{\}$
    if $|K_i| > 0$ then
      for $x \in K_i$ do
        $F \leftarrow F \cup$ leaf node in $\mathcal{T}$ containing $x$
      end for
      $C \leftarrow$ deepest common ancestor of $F$
      $C \leftarrow C \cup C_i$
    else
      $C_a \leftarrow$ deepest ancestor of $C_i$ with some kernel item(s) $K_a$
      for $x \in K_a$ do
        $F \leftarrow F \cup$ leaf node in $\mathcal{T}$ containing $x$
      end for
      $C \leftarrow$ deepest common ancestor of $F$
      Parent($C_i$) $\leftarrow C$
    end if
  end for
```

likelihood hierarchy to exceed $f$. Assume that in each iteration, we ask for clusterings on $h$ items. To find the total number of iterations $\tau$, we find the smallest value that satisfies $h + (h - f)(\tau - 1) \geq n$, where we have replaced $|K|$ with its upper bound $f$ in each iteration. We typically set $h = 35$. For $f = 16$ and $n = 115$, $\tau$ turns out to be 6. If each iteration is clustered by $m$ workers, the total cost of our iterative workflow becomes $\mathrm{O}\left(m\left\lceil\frac{(n-f)}{(h-f)}\right\rceil\right)$. The cost of our workflow is *independent of the size of the dataset* $\mathcal{D}$.

## 3.5 Categorization

At the end of our iterative workflow, we have a final maximum likelihood hierarchy $\mathcal{T}$, from which we must extract the consensus clustering granularity or frontier. As we stated in Section 2, the reason we construct this hierarchy is to preserve information about the granularities at which workers cluster items. We can now simply determine the consensus clustering *i.e.* the granularity workers are most likely to cluster on. This consensus clustering is the maximum-likelihood complete frontier in $\mathcal{T}$. We now outline a procedure to find this frontier. Subsequently we describe how we use this frontier for categorization.

**Maximum-Likelihood Frontier.** Suppose that $\mathcal{T}$ consists of the set of nodes $V$ with root node $R$. Associate with each node $v$, an event $E_v$ that $v$ is split by a worker *i.e.,* a worker chooses to give us nodes/concepts below $v$ in their frontier. Let $F$ be a frontier in $\mathcal{T}$ and let $A$ be the set of ancestors of $F$, excluding $R$. We define the likelihood of $F$ as,

$$\mathcal{L}(F) = p(E_R) \prod_{v \in F} p(\overline{E_v}|E_{v(1)}, \ldots, E_R) \prod_{v' \in A} p(E_{v'}|E_{v'(1)}, \ldots, E_R)$$

where $v(1), \ldots, R$ are the ancestors of $v$. $v(1)$ is the parent of $v$, $v(2)$ is the parent of $v(1)$, etc. Observe that $p(E_v|E_{v(1)}, \ldots, E_{v(k)}) = p(E_v|E_{v(1)})$ *i.e.,* $E_v$ is conditionally independent of the rest of its ancestors, given its immediate parent. We have,

$$\mathcal{L}(F) = p(E_R) \prod_{v \in F} p(\overline{E_v}|E_{v(1)}) \prod_{v \in A} p(E_v|E_{v(1)})$$

Given a set of worker responses (complete frontiers), we approximate $p(\overline{E_v}|E_{v(1)})$ as the ratio of the number of workers who gave node $v$ as a frontier to the total number of workers who gave node $v$ or its descendants as frontiers. Note that $p(E_R) = 1$. We are interested in $\underset{F}{\operatorname{argmax}}\, \mathcal{L}(F)$ which can be computed using the recurrence relation below.

$$D(v) = \max\left(p(\overline{E_v}|E_{v(1)}), p(E_v|E_{v(1)}) \prod_{u \in C(v)} D(u)\right)$$

where $C(v)$ is the set of children of $v$. Intuitively, at node $v$, there are two choices: either keep node $v$ as the frontier, or drill down and check for more likely frontiers, and using this, we can find the maximum likelihood frontier.

**Categorization on Frontier.** To carry out categorization we present workers with an interface similar to the clustering interface of Figure 2 with certain key differences. For each cluster in our consensus maximum likelihood clustering, we give a few 'pivot' items to the worker as exemplars for that cluster. Our assumption is that these pivots completely capture the concept represented by that cluster. Workers are asked to categorize items into the cluster which seems most appropriate. Once again, we do not point workers to any attributes in the data, instead relying on our pivots to allow them to infer the organization of our consensus clustering.

The cost of our categorization step is easily calculated — there are $|\mathcal{D}|-n$ items left after the clustering phase, and suppose we take $\theta$ votes per item. Typically $n \ll |\mathcal{D}|$, so the total categorization cost is then $\mathrm{O}(\theta|\mathcal{D}|)$, linear in the size of the dataset.

## 4. EXPERIMENTS

In our experiments, our goal is to qualitatively and quantitatively compare ORCHESTRA to other crowd-powered clustering algorithms.

**Datasets.** We used the following datasets in our experiments.

- Our first dataset, titled *shapes*, is a synthetic, stylized dataset consisting of shapes, our running example in Section 2. As described, each item is a random (SHAPE, SIZE, COLOR) tuple. The images from this dataset are also the ones displayed in Figure 2. We use this dataset because we can control the organizational hierarchies in this dataset, allowing us to evaluate the performance of algorithms on recovering clusterings across one or more hierarchies in the dataset.
- The second dataset, titled *scenes*, contains images from 13 categories [8] in natural and man-made surroundings. This dataset was also used in prior work on crowd clustering [9, 29].
- The third dataset, titled *imagenet*, contains images from ImageNet[6]. Images are sampled randomly from 20 categories: {buildings, cars, parrots, vulture, fruit, flower, vegetable, fighter, commercial, helicopter, ship, seahorse, whale, cheetah, lion, elephant, tiger, jellyfish, sparrow, leaves}.

Across all datasets, we conduct multiple runs of different algorithms on random subsets of the datasets.

**Algorithms.** We compare the following state-of-the-art crowd-powered clustering algorithms with ORCHESTRA:

- `CrowdClust`: This is the algorithm from Gomes et al. [9].
- `MatComp`: This is the algorithm from Yi et al. [29].

Code for both these algorithms was provided by the authors; we faithfully set the parameters as described in the papers. Both these algorithms require workers to cluster random samples of items repeatedly (recall Figure 3(b)), while ensuring that each item is assigned the same number of times across various samples.

Note that since these algorithms do not use categorization, we only compare the clustering phase of ORCHESTRA with these algorithms, enabling us to compare them on an equal footing. In all executions of these algorithms, we ensured that the algorithms had the exact same cost as ORCHESTRA, i.e., the same number of clustering tasks assigned to workers. We note that employing categorization would only lead to further reduction in cost—we study the benefits of categorization later on in this section.

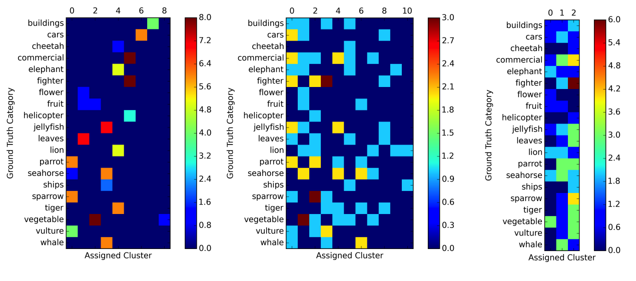
**Evaluated Aspects.** We evaluate the following aspects of ORCHESTRA:

- How do the eventual clusterings provided by ORCHESTRA compare with the clusterings provided by other algorithms both *qualitatively and quantitatively, on real-world datasets*?
- How do the eventual clusterings provided by ORCHESTRA compare with the clusterings provided by other algorithms both *qualitatively and quantitatively*, on datasets where we can *control* the organizational hierarchies?
- What is the impact, on cost and accuracy, of the *categorization* interface relative to the clustering interface?
- What is the benefit of *intelligently chosen samples* over randomly chosen ones for ORCHESTRA?
- How does the quality of clustering vary with the *number of workers* providing clusters?

**Metrics.** To quantitatively compare ORCHESTRA with prior work, we adopt two metrics that are also used in prior work: *(a) Variation of Information (VI)* [18] is an information theoretic, true distance metric used for comparing clusterings. A VI of 0 indicates a perfect match. *(b) Normalized Mutual Information (NMI)* [4] is a metric on $[0, 1]$— a perfect match gets a score of 1. In addition, for our stylized dataset, we introduce a new metric: *(c) Clustering Hierarchies*, the number of hierarchies that explain the resulting clustering returned by the algorithms. This is calculated as the minimum number of organizational hierarchies used in assigning all items to its cluster.

***How do the eventual clusterings provided by* ORCHESTRA *compare with the clusterings provided by other algorithms on real-world datasets?***

We compare the results of ORCHESTRA versus the other two algorithms on the scenes and imagenet datasets. We perform multiple runs for each dataset (5 for scenes and 2 for imagenet); each run is on a subset of $n = 115$ items (setting $n$ as described in Section 3.2) — to enable easy comparison only on clustering as opposed to clustering plus categorization. Each algorithm uses 5 samples across these $n$ items: `MatComp` and `CrowdClust` use random samples across $n$, while ORCHESTRA uses samples chosen by

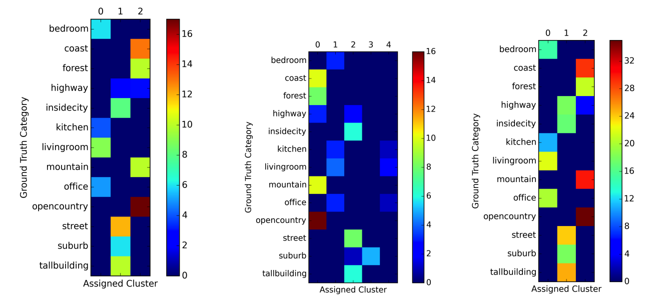


*(a)* ORCHESTRA *(b)* `CrowdClust` *(c)* `MatComp`

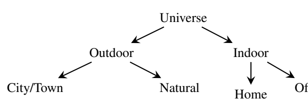
*Figure 6: Qualitative Comparison of Clusters provided by different algorithms on the ImageNet dataset*

We report the quantitative results in Table 1, where VI and NMI are computed for each algorithm relative to the ground truth for the respective datasets. We find that ORCHESTRA outperforms both `MatComp` and `CrowdClust` on both NMI and VI (recall that smaller is better for VI and worse for NMI). For instance, ORCHESTRA has an average VI of 1.241 across the five runs on the scenes dataset, which is much lower than the 1.408 of `CrowdClust` and 1.635 of `MatComp` — a 12% and 25% decrease respectively; only on the fifth run does `CrowdClust` have a better VI, and there too the difference is $< 0.02$. For NMI, ORCHESTRA consistently performs better than `CrowdClust` and `MatComp` in all runs. On the imagenet dataset, which is a much more challenging dataset, the difference is even larger, with ORCHESTRA improving over `MatComp` by more than 50% on VI, and `CrowdClust` by 11%.

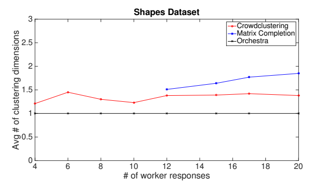
Next, we qualitatively examine the resulting clusters for the first run of the imagenet dataset via Figure 6. We depict a confusion matrix corresponding to each clustering algorithm, where each ground truth category corresponds to a row, while each cluster in the result corresponds to a column. It is therefore desirable that, in each row, there is a single column where there is non-zero presence (i.e., the color is not blue) — indicating that the entire ground-truth category appears as a whole in some cluster as opposed to being split. On examining Figure 6, it is clear that `CrowdClust` and `MatComp` have much worse matrices than ORCHESTRA. For instance, the only three rows in ORCHESTRA that have non-zero presence in more than one column are fruit, vegetable, seahorse (17/20 rows are *good*). On the other hand, for `MatComp`, all rows apart from cheetah, flower, helicopter, and ships have non-zero presence in more than one column — 4/20 rows are *good*, and for `CrowdClust`, 3/20 rows are *good*. For example, the commercial category, referring to commercial airlines, appears in clusters 0, 1, 2, 4, 5, 7 — *seven clusters* in `CrowdClust`, and in *all clusters* in `MatComp`! Thus, there is a clear benefit to ORCHESTRA in identifying and decomposing worker responses across different hierarchies and granularities, as opposed to operating on all of them at once. Furthermore, the clusters provided by ORCHESTRA have clearly associable concepts — cluster 4 consists of land animals; cluster 5, of aircraft, while there are no discernible concepts corresponding to the clustering of `MatComp` and `CrowdClust`. Similar results hold for the other runs on both datasets, indicating why ORCHESTRA does better than `MatComp` and `CrowdClust` on VI and NMI.



*(a) before categorization (b) with random sampling (c) after categorization*



*(d)* ORCHESTRA *Hierarchy*



*(e) Number of hierarchies for different algorithms*

*Figure 7: (a – d) Qualitative Comparison of Clusters provided by* ORCHESTRA *on the Scenes dataset (e) Clustering Hierarchy comparison*

| | | VI | | | NMI | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **Run #** | Orchestra | `CrowdClust` | `MatComp` | Orchestra | `CrowdClust` | `MatComp` |
| Scenes | 1 | **1.493** | 1.949 | **1.493** | **0.578** | 0.544 | **0.578** |
| | 2 | **1.232** | 1.386 | 1.381 | **0.710** | 0.691 | 0.608 |
| | 3 | **1.171** | 1.217 | 1.423 | **0.700** | **0.700** | 0.600 |
| | 4 | **1.267** | 1.461 | 2.041 | **0.689** | 0.603 | 0.360 |
| | 5 | 1.040 | **1.027** | 1.837 | **0.769** | 0.761 | 0.418 |
| | **avg** | **1.2406** | 1.408 | 1.635 | **0.689** | 0.660 | 0.513 |
| ImageNet | **avg** | **1.021** | 1.146 | 2.265 | **0.792** | 0.771 | 0.420 |

*Table 1: Quantitative Comparison of* Orchestra, `CrowdClust`, `MatComp` *on **VI** and **NMI**: For Scenes dataset, 5 runs were performed for each algorithm. The best results in each experiment are in bold.*

***How do the eventual clusterings provided by*** Orchestra ***compare with the clusterings provided by other algorithms on a stylized dataset?***

For this experiment, we take 20 different worker responses for the shapes dataset: recall that the shapes dataset is small enough to be grouped into one single clustering task. Thus, our Orchestra is restricted to the HierarchyConstruction step, and all algorithms are run on the same data, by repeatedly taking subsets of worker responses.

For each run, we compute the number of hierarchies (out of {Shape, Size, Color}) that appear in the consensus clustering. Figure 7(e) shows the average number of organizing hierarchies vs. the number of worker responses ($r$), repeated over 100 random runs. While Orchestra is always able to identify one dominant organizing hierarchy, the other algorithms tend to mix hierarchies frequently, with the average number for `MatComp` being larger than `CrowdClust`. (We were unable to run `MatComp` for $r < 12$—the spectral clustering found no principal eigencomponents due to the matrix being low-rank.) This is despite the fact that 85% of workers are actually organizing by Shape — so most samplings of worker responses get this dominant organization.

We would also like to test all algorithms in situations when there are multiple hierarchies *i.e.,* workers are equally likely to use any one of these organizations. We pick a subset of 5 responses from the 20 that we received, where 2 workers organize by Shape, 2 by Color and 1 by Size. On this data, `CrowdClust` mixes the Shape and Size organizing principles: the clustering they get is `Small Shapes`, `Big Triangles`, `Big Rectangles`. Similarly, `MatComp` is unable to come up with a consensus clustering that relies on a single hierarchy of organization. In contrast, Orchestra is able to identify a consensus clustering based on Shape.

***What is the impact, on cost and accuracy, of the* categorization *interface relative to the clustering interface?***

Earlier, we noted that the cost (per item) of the categorization phase is lower than that that in the clustering phase. In this section, we evaluate the quality of clusterings obtained following the categorization phase and the cost associated with it. For this experiment, we asked workers to categorize 175 items from the scenes dataset, organized into 5 tasks with 35 items each. We used 10 images from each consensus cluster found by Orchestra after one run on the scenes dataset, shown in Figure 7(a) as *pivots* for categorization. We asked five independent workers to answer each task; items were assigned to the cluster with the most votes.

Figures 7(a) and 7(c) show the quality of clustering before and after categorization. Essentially, the categorization stage preserves the quality of the clustering stage, *i.e.*, no new "bad" rows—corresponding to errors—are introduced. The `highway` category was split across two clusters at the end of the clustering stage. The pivots actually help the workers in placing most of the `highway` images in the `outdoor-city/town` cluster, as opposed to the `outdoor--natural` cluster.

The workers were paid 10 cents per task, as opposed to 20 cents per task for clustering. Combined with the fact that only 5 worker

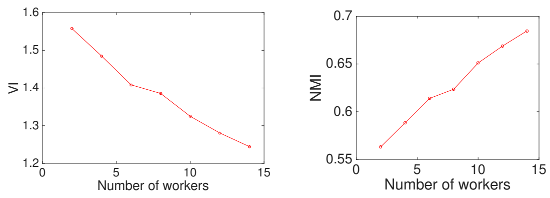


*Figure 8: (a) VI and (b) NMI with varying number of worker responses*

responses were sought (as opposed to 15 for clustering), this stage has a per-item cost that is at least 1/6-th the cost in clustering stage. Further, categorization does not require different tasks to share a common kernel of items — each item needs to appear in only one task. This significant cost saving, while retaining the quality of the clustering stage, demonstrates that categorization is a cheap, unambiguous way of clustering items, once the clusters have been discovered with reasonable confidence.

***What is the benefit of intelligent sampling (*** GenerateSample ***) compared to random sampling?***

To evaluate the impact of our GenerateSample procedure, we run our HierarchyConstruction and MergingHierarchies algorithms on randomly sampled samples for the scenes dataset. One such result is shown in Figure 7(b). While this is similar to the clustering with GenerateSample, note that clusters 1 and 4 have nearly the same set of items — `indoor`. This happens because the corresponding samples have no items from `indoor` that are common, and hence it is not possible to determine that these sets of items should be grouped together, resulting in a near-duplicate cluster. This is undesirable behavior, motivating the need for a *kernel* of items.

***How does the quality of clustering vary with the number of workers who perform clustering?***

To evaluate this, we plot the average VI and NMI for all runs of scenes dataset in Figures 8(a) and 8(b). Different samples of responses are taken 50 times, and the results are averaged over these 50 trials. The performance of Orchestra degrades gracefully as the number of worker responses is decreased. Note that even with 6 responses, our performance (in terms of VI) is better than `CrowdClust` with 15 responses (refer to Table 1). Further, Orchestra is able to outperform `MatComp` on both VI and NMI with just 2 worker responses.

The above results demonstrate that while more worker responses help in getting better clusterings, Orchestra is still able to outperform existing algorithms with fewer number of responses.

## 5. RELATED WORK

Our work is related to prior work on crowd clustering, taxonomy generation, as well as other work on crowdsourced algorithms.

**Crowd-Based Clustering.** Our work is most closely related to the prior work on crowd-powered clustering via a matrix completion approach, including [9, 30, 29]. In all these papers, worker clusterings are performed on randomly selected sets of items. Then, the results of worker clusterings are interpreted as pairwise comparisons: for example, if a worker placed items a, b, c in one cluster, then this is interpreted as three pairwise comparisons, between a and b, b and c, and c and a. Subsequently, matrix completion techniques are applied to infer the missing entries in the matrix. We identify multiple ways this line of work fails to take into account the complexity of crowd-powered organization: *(a)* Mixing of hierarchies and frontiers: since these papers do not interpret different worker responses as being derived from different hierarchies or frontiers within a hierarchy, they tend to provide clusterings that mix hierarchies and mix frontiers within a hierarchy, leading to poor organization. ORCHESTRA, on the other hand, carefully treats distinct hierarchies as well as frontiers within a hierarchy. *(b)* Random samples of items: unlike ORCHESTRA, which uses intelligently chosen samples of items, these papers use random samples of items. *(c)* Loss of information: since these papers interpret worker clusterings as pairwise information within the matrix, they lose valuable information, as opposed to ORCHESTRA, which operates on clusters as a whole. *(d)* No categorization: since the ORCHESTRA approach identifies the consensus hierarchy, this hierarchy can be leveraged to subsequently categorize the remaining items, providing further cost savings. None of these papers perform categorization to further save costs.

There has been other work on variants of clustering: Heikinheimo and Ukkonen [11] describe the CROWD-MEDIAN algorithm whose goal is to compute centroids, as opposed to identifying clusterings. For example, as soon as they locate *some* representative object, they can stop, instead of having to organize all the objects, like in our case. Further, they do not explicitly capture different perspectives of workers, limiting the applicability in practice. Davidson et al [5] provide theoretical guarantees for aggregation (GROUP BY) queries, where workers are asked to answer questions of the form *"are a and b of the same type"*. This paper makes a simplifying assumption that there is a correct answer (*i.e.*, there is a ground truth collection of types), with workers answering incorrectly with a fixed error probability. ORCHESTRA uses a more general question type (i.e., cluster a collection of objects) since it provides more context, and also does not make the same assumptions about worker answer correctness. [31] propose a *collaborative clustering* scheme where they discover user preferences for clustering as opposed to identifying a consensus clustering of the data, as we do.

**Crowd-Based Hierarchy Building.** A variety of papers use the crowd for hierarchy construction: Chilton et al. and Bragg et al. [3, 2] use text labels and filtering on the labels to create a hierarchy while Sun et al. and Karampinas et al. [22, 14] ask the crowd for pairwise ancestor descendant relationships, also demonstrating that identifying the optimal set of ancestor descendant questions is NP-HARD. While hierarchy construction could, in principle, be used as a precursor to clustering or organization, none of these papers take into account different organizational principles (i.e., the existence of many hierarchies); it remains to be seen if hierarchy construction can be improved by taking into account our techniques for identifying organizational principles. At the same time, it would be interesting to extend our algorithms to generate a complete hierarchy on the set of clustered items.

**Other Crowdsourcing or Active Learning Work.** Past work on active learning has utilized human workers to provide constraints for automated clustering algorithms. This work relies on human competence in making judgments for ambiguous image pairs, rather than using humans expertise in organizing data into clusters. Biswas et al. [1] obtain hard pairwise constraints in a crowdsourced setting by asking targeted questions related to an item pair. Lad et al. [15] ask humans to provide attribute-based explanations, rather than pairwise constraints, and opt to use these as soft constraints. Neither work allows humans to explicitly cluster data.

Other work focuses on learning a embedding of the data using crowd workers, and then clustering in this latent space using a standard clustering algorithm. The disadvantage of this approach is that it mashes together worker responses, and loses rich information that can be extracted from workers. Wilber et al. [28] create concept embeddings by combining human experts with automation. Tamuz et al. [23] learn a 'crowd kernel', which embeds items into a Euclidean space. Neither work explores how to cluster this embedding effectively, to extract different organizational hierarchies.

Prior work on categorization does not attempt to discover organizational principles, instead presenting a predefined organization to workers, and asking them to assign items into categories. Both papers in this space [20, 7] use graph-based approaches to carry out categorization into a taxonomy of concepts. Our work can be considered a precursor to the algorithms described in these papers, which can be integrated into our categorization step.

Work on entity resolution (ER) can be regarded as clustering with a different objective: find clusters of homogenous (identical) items, in contrast to our setting, where the organizing principle is not clear. [16, 27, 26, 24, 25] are all examples of work that rely on human judgments to carry out ER, all using pairwise comparisons.

## 6. CONCLUSIONS AND FUTURE WORK

We described ORCHESTRA, our approach to perform consensus organization of corpora using the crowd. We developed techniques for identifying maximum likelihood frontiers, for issuing additional questions from the crowd, ensuring that the eventual frontiers have high coverage, and combining information across different crowd answers. We demonstrated the benefits of ORCHESTRA versus other crowd-clustering schemes on three datasets with different characteristics. The organizations returned by ORCHESTRA are higher quality (up to 24% improvement on VI) and are more cost effective (up to a reduction of $6\times$) than other schemes.

We believe our paper raises a number of interesting unanswered questions: *(a)* Would it help to ask workers to describe, in words, the clustering that they are using, and combine that information with the hierarchy construction or merging algorithm? Once we identify the maximum likelihood hierarchy, could we ask workers to cluster on that hierarchy (in words)? *(b)* Would it help at all to drill-down on certain nodes in a given hierarchy by asking workers to only organize objects that are known to be part of the concept corresponding to that node? One drawback of this is that we may end up mixing hierarchies: if we apply drill-down to a node containing triangles, we may end up introducing size or color as the organizational principle at that point. *(c)* We observed that often the hierarchies that we obtain (corresponding to the cliques) may in fact share many clusterings: in such cases, we still just end up picking the largest clique. Would it be possible to merge these hierarchies together, despite not being part of a single clique, by using a more tolerant merging criteria—would that lead to any benefits? *(d)* Can we combine our algorithm with an automated scheme that provides prior assessments of similarity using automatically extracted features? We plan to address these, and other questions in follow-up work.

## 7. REFERENCES

[1] A. Biswas and D. Jacobs. Active image clustering with pairwise constraints from humans. *International Journal of Computer Vision*, 108(1-2):133–147, 2014.

[2] J. Bragg, Mausam, and D. S. Weld. Crowdsourcing multi-label classification for taxonomy creation. In *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2013, November 7-9, 2013, Palm Springs, CA, USA*, 2013.

[3] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay. Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1999–2008. ACM, 2013.

[4] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[5] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*, pages 225–236. ACM, 2013.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[7] J. Fan, G. Li, B. C. Ooi, K.-l. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1015–1030. ACM, 2015.

[8] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE, 2005.

[9] R. G. Gomes, P. Welinder, A. Krause, and P. Perona. Crowdclustering. In *Advances in neural information processing systems*, pages 558–566, 2011.

[10] S. Handel and S. Imai. The free classification of analyzable and unanalyzable stimuli. *Perception & Psychophysics*, 1972.

[11] H. Heikinheimo and A. Ukkonen. The crowd-median algorithm. In *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.

[12] S. Imai and W. Garner. Discriminability and preference for attributes in free and constrained classification. *Journal of Experimental Psychology*, 69(6):596, 1965.

[13] A. Jain, J. Young-Seo, K. Goel, A. Kuznetsov, A. Parameswaran, and H. Sundaram. It's just a matter of perspective(s): Crowd-powered consensus organization of corpora. *Technical Report (http://data-people.cs.illinois.edu/cluster.pdf)*, 2015.

[14] D. Karampinas and P. Triantafillou. Crowdsourcing taxonomies. In *The semantic web: Research and applications*, pages 545–559. Springer, 2012.

[15] S. Lad and D. Parikh. Interactively guiding semi-supervised clustering via attribute-based explanations. In *Computer Vision–ECCV 2014*, pages 333–349. Springer, 2014.

[16] J. Lee, H. Cho, J.-W. Park, Y.-r. Cha, S.-w. Hwang, Z. Nie, and J.-R. Wen. Hybrid entity clustering using crowds and data. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, 22(5):711–726, 2013.

[17] D. L. Medin, W. D. Wattenmaker, and S. E. Hampson. Family resemblance, conceptual cohesiveness, and category construction. *Cognitive psychology*, 19(2):242–279, 1987.

[18] M. Meilă. Comparing clusterings by the variation of information. In *Learning theory and kernel machines*, pages 173–187. Springer, 2003.

[19] F. Milton and A. Wills. The influence of stimulus properties on category construction. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 30(2):407, 2004.

[20] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *Proceedings of the VLDB Endowment*, 4(5):267–278, 2011.

[21] G. Regehr and L. R. Brooks. Category organization in free classification: The organizing effect of an array of stimuli. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 21(2):347, 1995.

[22] Y. Sun, A. Singla, D. Fox, and A. Krause. Building hierarchies of concepts via crowdsourcing. *arXiv preprint arXiv:1504.07302*, 2015.

[23] O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. T. Kalai. Adaptively learning the crowd kernel. *arXiv preprint arXiv:1105.1033*, 2011.

[24] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *Proceedings of the VLDB Endowment*, 7(12):1071–1082, 2014.

[25] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.

[26] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *Proceedings of the VLDB Endowment*, 6(6):349–360, 2013.

[27] S. E. Whang, J. McAuley, and H. Garcia-Molina. Compare me maybe: Crowd entity resolution interfaces.

[28] M. J. Wilber, I. S. Kwak, D. Kriegman, and S. Belongie. Learning concept embeddings with combined human-machine expertise. *arXiv preprint arXiv:1509.07479*, 2015.

[29] J. Yi, R. Jin, A. K. Jain, and S. Jain. Crowdclustering with sparse pairwise labels: A matrix completion approach. In *AAAI Workshop on Human Computation*, volume 2, 2012.

[30] J. Yi, R. Jin, S. Jain, T. Yang, and A. K. Jain. Semi-crowdsourced clustering: Generalizing crowd labeling by robust distance metric learning. In *Advances in Neural Information Processing Systems*, pages 1772–1780, 2012.

[31] Y. Yue, C. Wang, K. El-Arini, and C. Guestrin. Personalized collaborative clustering. In *Proceedings of the 23rd international conference on World wide web*, pages 75–84. ACM, 2014.