

© 2016 Ayush Jain

TOWARDS OPEN-ENDED CROWD-POWERED DATA  
PROCESSING: A CASE STUDY OF CLUSTERING AND  
COUNTING

BY

AYUSH JAIN

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Computer Science  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Professor Aditya G. Parameswaran

# Abstract

Due to the widespread use and importance of crowdsourcing in gathering training data at scale, the data management community has devoted its efforts in understanding and optimizing fundamental primitives like filters and joins. These primitive *boolean operations*, where the human responses come from a small, finite space of possible answers, are inadequate for a number of data analysis tasks, especially those involving images, videos and maps. There is, thus, a need for *open-ended crowdsourcing* in order to get more fine-grained information from humans that can be used in developing sophisticated AI systems. In this thesis, we study two popular open-ended crowdsourcing problems. The first, *clustering*, is the problem of organizing a collection of objects (images, videos) by allowing workers to form as many clusters as they would like and organize items across them. The second, *counting*, is the problem of counting objects in images. In this thesis, we develop models to reason about human behavior for both problems, and use these models to design provably cost-efficient algorithms that provide high-quality results, as compared to currently available approaches.

*To my parents, who taught me to always do my best.*

# Acknowledgments

In the first place, I'd like to express my gratitude to Aditya Parameswaran for being a wonderful mentor and not just an advisor. Aditya is a clear thinker and has the ability to simplify even the most convoluted ideas and problems — I invariably left every meeting with simple insights (even from my half-baked ideas!) and clear directions to explore. In addition, Aditya has always been available for advice, be it job hunting or even exploring the Urbana-Champaign area. I have gained a lot from my interactions and time with Aditya – his dedication and commitment to work has been inspiring and will stay with me for a long time. I am sure I have a lot more to learn from him.

I'd also like to thank my collaborators – Hari Sundaram, Jennifer Widom and Dan Roth – for their insightful comments and stimulating discussions over the last two years that have helped make this thesis richer.

I have also had the pleasure of collaborating with other students who made working on these problems more fun and rewarding. I'd like to thank Akash Das Sarma, Joon Young Seo, Karan Goel and Andrew Kuznetsov.

I'd also like to thank other members of the group - Tarique Siddiqui, Sajjadur Rahman, Silu Huang, Yihan Gao, Vipul Venkataraman, Liqi Xu, Stephen Macke, Himel Dev, among many others - for stimulating intellectual conversations over lunch and coffee.

My heartfelt gratitude to my parents and family for encouraging me to aim higher. Their support and encouragement over the years has been the greatest gift I have received.

Lastly, to Aparna, for everything.

# TABLE OF CONTENTS

List of Tables . . . . .	vi
List of Figures . . . . .	vii
CHAPTER 1 Introduction . . . . .	1
CHAPTER 2 ORCHESTRA: Crowd-powered Consensus Organization of Corpora . . . . .	3
2.1 Introduction . . . . .	3
2.2 Preliminaries . . . . .	5
2.3 Hierarchy Construction . . . . .	11
2.4 Extending the Hierarchy . . . . .	20
2.5 Categorization . . . . .	25
2.6 Experiments . . . . .	26
2.7 Related Work . . . . .	43
2.8 Summary . . . . .	45
CHAPTER 3 JELLYBEAN: Crowd-Vision-Hybrid Counting Algorithms . . .	46
3.1 Introduction . . . . .	46
3.2 Preliminaries . . . . .	49
3.3 Crowdsourcing-Only Solution . . . . .	51
3.4 Incorporating Computer Vision . . . . .	57
3.5 Experimental Study . . . . .	65
3.6 Related Work . . . . .	76
3.7 Summary . . . . .	78
CHAPTER 4 Conclusions . . . . .	79
REFERENCES . . . . .	80
APPENDIX A Worker Behavior in Counting . . . . .	86

# List of Tables

2.1	Quantitative Comparison of algorithms . . . . .	30
2.2	Comparison of clusterings by workers on Amazon MTurk . . . . .	32
2.3	The mean and variance of ORCHESTRA’s quality across 50 cluster- ing HITs for each dataset . . . . .	38
2.4	Human Evaluation of organizations provided by algorithms for scenes dataset . . . . .	41
2.5	Human Evaluation of organizations provided by algorithms for im- agenet dataset . . . . .	42
3.1	Notations . . . . .	51
A.1	Validation of Interaction Model and Worker Error Model . . . . .	88

# List of Figures

2.1	Distinction between concept trees and hierarchies . . . . .	7
2.2	Clustering interface . . . . .	9
2.3	Categorization interface . . . . .	9
2.4	ORCHESTRA Workflow . . . . .	10
2.5	An example demonstrating our iterative workflow approach on the Shapes dataset of Figure 2.1(f). . . . .	12
2.6	Variation in the probability of ORCHESTRA providing a clustering based on SHAPE with the number of worker responses . . . . .	16
2.7	(a) Plot showing sample size $n$ vs. size of complete frontier $f$ to ensure $\delta$ expected coverage. (b) Plot showing lower bound on ex- pected coverage vs. actual expected coverage over 1000 trials – the dotted line is the 45 deg line. . . . .	22
2.8	Sample Images from the Datasets . . . . .	26
2.9	Qualitative Comparison of clusters provided by different algorithms on <i>scenes</i> . . . . .	31
2.10	Qualitative Comparison of Clusters provided by different algorithms on <i>imagenet</i> . . . . .	31
2.11	Hierarchy constructed by ORCHESTRA on the <i>Scenes</i> dataset . . . . .	32
2.12	Performance of prior work with best-case future worker responses . . . . .	33
2.13	Perspective Disambiguation for the shapes dataset . . . . .	34
2.14	Clusters provided by ORCHESTRA on the <i>Scenes</i> dataset . . . . .	35
2.15	Advantage of using kernel-based GENERATESAMPLE . . . . .	35
2.16	Clustering: Performance of prior work with varying budget . . . . .	37
2.17	ORCHESTRA’s robustness to variations in item sampling . . . . .	39
2.18	ORCHESTRA’s robustness to variations in number of worker re- sponses for the <i>scenes</i> dataset . . . . .	39
2.19	Effect of drill-down . . . . .	40
3.1	Counting: Challenging image for Machine Learning . . . . .	47
3.2	Counting: Worker Error . . . . .	47
3.3	Segmentation Tree . . . . .	50
3.4	Biological image (a) before and (b) after partitioning . . . . .	57
3.5	Articulation Point . . . . .	60



3.6	Performance of algorithms for merging partitions . . . . .	64
3.7	Images in the crowd dataset . . . . .	66
3.8	Counting Interface: Sample Image shown to workers . . . . .	67
3.9	Counting Interface: Instructions shown to workers . . . . .	68
3.10	Performance of Face-ML . . . . .	71
3.11	Counting Accuracy . . . . .	72
3.12	Cost of counting for the crowd dataset . . . . .	72
3.13	Aggregating worker answers . . . . .	75
A.1	Worker errors on biological dataset . . . . .	87
A.2	Worker Behavior . . . . .	88

# CHAPTER 1

## Introduction

With the increasing intelligence of AI systems and advancements in hardware to enable processing of big data, machine learning has shown tremendous promise in solving the most pressing challenges of our time. It is thus no surprise that crowdsourcing has evolved as the primary means to gather or generate annotated training data at scale for these new applications. Starting from identifying spam emails to now scraping satellite images to find a lost aircraft [1] and detecting cancer in tissue images [2], crowdsourcing applications have witnessed a manifold increase even in its nascent stage of development.

Over the last few years, researchers in crowdsourcing have made rapid progress to meet the demands posed by evolving applications. As a result, the community has now developed optimized algorithms for fundamental tasks like filters and joins [3, 4, 5, 6, 7, 8]. These works have striven to provide accurate answers while keeping costs low, for basic tasks where human responses are derived from a small space of possible answers. We call these *boolean operators*, as natural analogs to computer-like operators.

These boolean operators, while being easy to abstract and develop algorithms for, do not make the best use of human time or ability. As an example, consider the task of organizing a collection of images. Current state-of-the-art techniques only allow humans to compare two images and make a judgement about whether they are similar or not. The restriction of looking at only two images does not completely utilize humans ability to look at a collection of images and identify different themes in the collection, thereby organizing them better. Furthermore, by providing humans with the ability to organize images into clusters, humans can provide us more information in lesser time through smaller number of tasks.

This warrants the study and development of *open-ended crowdsourcing*, where humans perform tasks where their responses do not necessarily come from a small, finite set of possibilities. In addition to making the best use of human ability, open-ended crowdsourcing is also relevant in present context. One, our studies have shown that a significant fraction of tasks – 47% of the tasks on images and text – on a typical crowdsourcing platform are open-ended. Two, as machine intelligence strives to come closer to human intelligence, it must be trained on data incorporating fine-grained human reasoning instead of coarse-grained machine responses.

The goal of this thesis is to study two popular open-ended crowdsourcing applications, and for each (a) develop formalism and characterize open-ended operators, and

(b) design algorithms that are cost-efficient while providing provably accurate answers.

Our first application, motivated by the example above, is *clustering*, i.e, the problem of organizing a collection of objects (images, videos) by allowing workers to form as many clusters as they would like and organize items across them. This is especially useful within commercial, government, humanitarian, and educational institutions that have seen a proliferation of images and videos with the decreasing cost of storage. Two factors make this human-powered data organization challenging. First, when asked to organize the same set of items, different people may propose different organizations. As an example, to organize a collection of travel photos, some might prefer to organize by location, whereas others might prefer to organize them as *beach/downtown/hotel*. We formalize and address these issues, and develop our workflow ORCHESTRA to *orchestrate* crowdsourced data organization in Chapter 2.

The second application, *counting*, is the problem of counting the number of objects of a particular type in an image. This is a ubiquitous problem with many applications – biologists are often interested in counting the number of cell colonies in periodically captured photographs of petri dishes; counting the number of individuals at concerts or demonstrations is often essential for surveillance and security [9]; counting is often necessary in military applications; counting nerve cells or tumors is standard practice in medical applications [10]; and counting the number of animals in photographs of ponds or wildlife sanctuaries is often essential for animal conservation [11]. Furthermore, counting is a prerequisite to other, more complex computer vision problems requiring a deeper, more complete understanding of images. In spite of being useful in a variety of areas, counting is still incredibly hard for automated algorithms. As an example, most object detectors fail to detect objects that are hidden behind (or *occluded by*) other objects. On the other end of the spectrum, even humans have trouble in counting - we found that humans can only count accurately up to a limit. Therefore, in Chapter 3 (reference: [12]), we present our JELLYBEAN suite of algorithms, that use the best of crowds and computer vision, and judiciously decomposes images to elicit accurate counts from workers at low costs.

## CHAPTER 2

# ORCHESTRA: Crowd-powered Consensus Organization of Corpora

### 2.1 Introduction

The decreasing cost of storage has led to the proliferation of images and videos within commercial, government, humanitarian, and educational institutions. Unfortunately, automated schemes perform poorly at organizing this data since they are not able to interpret or understand content adequately. Human beings, on the other hand, can easily interpret content, but the scale of these collections precludes the use of any single human individual for manual organization. So we turn to crowdsourcing for organizing content.

However, 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 the issues, we asked 20 workers on Amazon’s Mechanical Turk to each cluster a stylized set of 25 images, where each image shows an object with a random shape, color, and size. For instance, one of the images was a large blue circle. 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 much 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 address these issues in this chapter, developing a cost-efficient, accurate, and 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 organizational perspective, while at the same time humans do adopt different perspectives [13, 14, 15, 16, 17].*

Prior work has considered the problem of crowd clustering [18, 19, 20]. However, this line of work falls 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 have workers cluster random samples of objects; however in the absence of any relationship between the samples that the workers are asked to cluster, this can be costly. Indeed, [18] report in their paper that they require each item to appear in a large number of 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 transformation is done 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 2.7.

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 that perspective. We use these algorithms in conjunction with techniques to identify the maximum likelihood granularity organization 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, even if 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, 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 chapter:

- We model the notions of perspectives, and granularities formally. (Section 2.2)
- We design, ORCHESTRA, a *robust, low-cost workflow for organization* comprising the following algorithmic components:
  - We develop techniques to relate worker clusterings to each other (to identify worker perspectives), formalize the identification of the consensus perspective using maximum likelihood. We demonstrate its equivalence to the MAX-CLIQUE problem, thereby showing that it is NP-HARD. (Section 2.3.1)
  - We formulate the problem of identifying consensus in the presence of worker errors or ambiguous items as a novel N-CONSISTENCY problem and show that it is NP-HARD. (Section 2.3.3)
  - We develop probabilistic techniques to ensure that our maximum likelihood perspective has *adequate coverage* of the space of all concepts in the dataset. (Section 2.4.1)
  - We develop the notion of a *kernel* to relate worker clusterings on different samples of items to the maximum likelihood hierarchy. (Section 2.4.2)
  - We design techniques to *extend* the current maximum likelihood hierarchy by merging worker responses on new items to the existing hierarchy. (Section 2.4.3)
  - 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 2.5)
- We further couple these algorithmic contributions with experiments on three real datasets on Amazon’s Mechanical Turk (Section 2.6), and demonstrate that our techniques lead to better quality clusterings (up to a factor of 4 on accuracy and a factor of 6 on recall), when compared to prior work in this space.

## 2.2 Preliminaries

In this section we introduce the problem of consensus organization, and provide a high-level overview of our solution. In Section 2.2.1, we present a sequence of definitions

to help formalize our problem. In Section 2.2.2, we describe our model for worker behavior and our interfaces, and in Section 2.2.3, we describe the ORCHESTRA workflow at a high level. Finally, in Section 2.2.4, we provide a breakdown of the ORCHESTRA clustering phase that will be our focus in the subsequent sections.

### 2.2.1 Data Model

In this subsection, we provide a series of definitions related to four ideas: clusterings, hierarchies, frontiers and complete frontiers. At a high level, workers provide clusterings, hierarchies refer to organization of clusters via the subsumption relationship, and frontiers are portions of a hierarchy that are disjoint. We begin first with a formal definition of a clustering.

**Definition 1 (Clustering).** *Given a set of items  $\mathcal{D}$ , a clustering is a partitioning of  $\mathcal{D}$  into clusters  $C_1, \dots, C_k$  such that,*

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

Every cluster in a clustering (and by consequence any set of items) can be associated with an *underlying latent concept*. Informally, a concept is a property that is satisfied by each item in a cluster. For example, in Figure 2.1(f), the clusters—from top to bottom, one corresponding to each row—represent the concepts `Triangles`, `Quadrilaterals` and `Ellipses`. 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; we may therefore use clusters and concepts interchangeably.

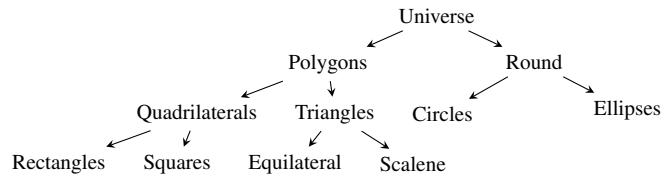
Concepts may have subset-superset relationships among them. Formally, we say that concept  $B$  *generalizes* concept  $A$  (denoted  $B \succ A$ ) if every item in  $\mathcal{D}$  that is an instance of  $A$  is also an instance of  $B$ . For example, the concept `Quadrilaterals` generalizes `Rectangles`.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

We can organize concepts based on the *generalize* relationship into a tree with `Universe` as its root. We call this tree a *hierarchy*.

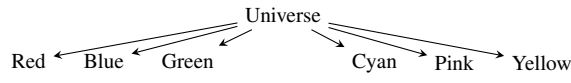
**Definition 2 (Hierarchy).** *For the set of items  $\mathcal{D}$ , a hierarchy  $\mathcal{T}(\mathcal{D})$  is a rooted concept tree where*

- (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) *Every instance of  $C \in \mathcal{T}(\mathcal{D})$  is also an instance of exactly one of its children in  $\mathcal{T}$*
- (3) *For every  $C \in \mathcal{T}(\mathcal{D})$ , at least one item in  $\mathcal{D}$  is an instance of  $C$ .*

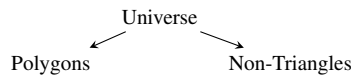
Intuitively, a hierarchy is a concept tree in which every item of  $\mathcal{D}$  can be assigned to exactly one of the leaf nodes (and consequently all of its ancestors), and no leaf node is empty. Note that a dataset may be representable by multiple hierarchies.



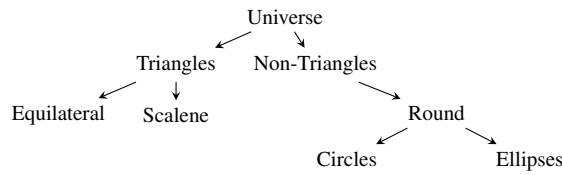
(a) Hierarchy



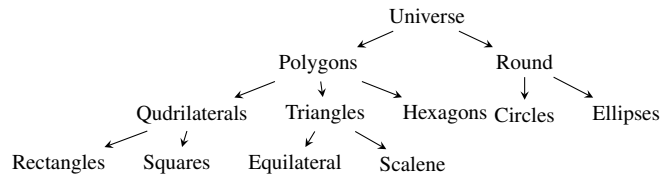
(b) Hierarchy



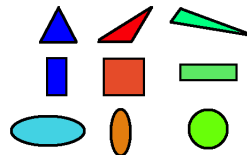
(c) Not a hierarchy



(d) Not a hierarchy



(e) Not a hierarchy



(f) Shapes

Figure 2.1: Distinction between concept trees and hierarchies. (a) – (e): Concept trees for the clustering example shown in Figure 2.2 — (a) and (b) are hierarchies; (c) is not a hierarchy since it violates (3) in Definition 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 chapter



Figure 2.1 shows some concept trees for the dataset of shapes in Figure 2.1(f). Figures 2.1(a) and 2.1(b) depict hierarchies since every item in the dataset can be assigned to one of the leaf nodes. Other trees, shown in Figure 2.1(c), 2.1(d) and 2.1(e), are not hierarchies. Figure 2.1(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 2.1(d), the concept Round does not cover all instances of its parent concept Non-Triangles since the dataset has a Quadrilaterals concept in addition to Round. Figure 2.1(e) is also not a hierarchy as there are no instances of Hexagons in the dataset.

Notice 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, all built on the subset relation. We will treat these hierarchies as equivalent.

We now describe a method to find the hierarchy corresponding to any subset of items  $S \subseteq \mathcal{D}$ , when a global hierarchy  $\mathcal{T}(\mathcal{D})$  is given. This is important because workers often see a small number of items at a time, hence knowing how to *project* a hierarchy to a subset of items is necessary. Say  $S$  is associated with a concept  $C \in \mathcal{T}(\mathcal{D})$  such that every item in  $S$  is an instance of  $C$ . (It is easy to see that there has to be such a concept.)  $S$  may or may not contain every instance of  $C$ . Consider the subtree of  $\mathcal{T}(\mathcal{D})$  rooted at  $C$ . If we enforce Universe as the root and condition (3) 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  $\mathcal{T}(S)$ . For instance, in Figure 2.1(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 3 (Frontier).** A frontier  $F$  is a set of disjoint concepts  $\{C_1, \dots, C_k\}$  in a hierarchy  $\mathcal{T}(\mathcal{D})$  such that:

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

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 2.1(b),  $\{\text{Red, Green, Blue}\}$  forms a valid frontier. Since concepts in a frontier  $F$  are disjoint, an item in  $\mathcal{D}$  can be an instance of *at most* one concept in  $F$ .

**Definition 4 (Complete Frontier).** A frontier  $F$  in  $\mathcal{T}(\mathcal{D})$  is said to be complete if  $\bigcup_{i=1}^k C_i = \text{Universe}$

In other words,  $F$  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 2.1(b), the frontier  $\{\text{Red, Blue, Green}\}$ , when expanded to  $\{\text{Red, Blue, Green, Cyan, Pink, Yellow}\}$  becomes complete as every item in the dataset is an instance of exactly one of these concepts. Similarly, for Figure 2.1(a),  $\{\text{Polygons, Circles, Ellipses}\}$ ,  $\{\text{Quadrilaterals, Triangles, Round}\}$ ,  $\{\text{Squares, Rectangles, Equilateral, Scalene, Circles, Ellipses}\}$  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 specific clustering of the set of items can be viewed as an operationalization of a complete frontier on a set of items. Thus, a complete frontier is associated with a specific clustering or organization of the dataset.

### 2.2.2 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 2.2 for an example worker session using our clustering interface.

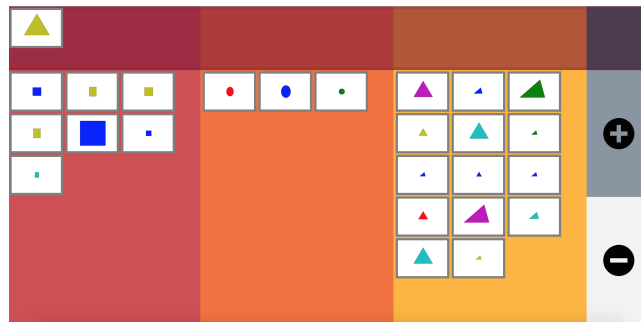


Figure 2.2: Clustering interface. In this example, workers are asked to organize shapes into multiple clusters. They can determine the number of clusters by using the '+' and the '-' buttons seen on the right.

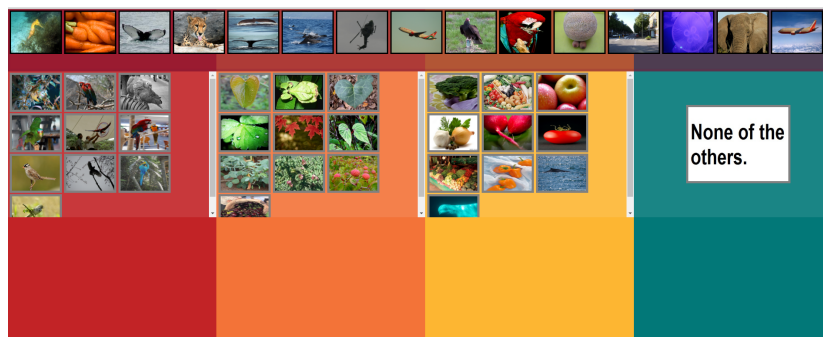


Figure 2.3: Categorization interface. In this example, workers are asked to organize shapes into existing clusters (Notice that '+' and '-' are missing here). Pivot items in these clusters are pre-populated. A separate none-of-these cluster option is also provided.

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 2.1(a) and 2.1(b) are both valid hierarchies

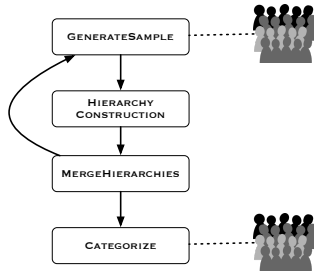


Figure 2.4: ORCHESTRA Workflow

for the data shown in Figure 2.2. 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*, shown in Figure 2.3, 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.

### 2.2.3 Overall Workflow for ORCHESTRA

Our overall workflow comprises of two phases: the clustering phase and the categorization phase. The clustering phase discovers a consensus hierarchy of the data using just a small fraction of items from the corpus. Once the consensus organization on this hierarchy is 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 [18, 20], 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. Figure 2.4 shows our workflow; the first three boxes refer to the clustering phase, while 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.

### 2.2.4 Clustering Phase for ORCHESTRA

Given a dataset  $\mathcal{D}$ , the goal of the clustering phase is to recover the most likely hierarchy  $\mathcal{T}_{ml}(\mathcal{D})$ , such that the probability of a worker choosing this hierarchy's or-

ganizational perspective is maximum. To find  $\mathcal{T}_{ml}(\mathcal{D})$ , ORCHESTRA has an iterative refinement procedure that performs repeated iterations of (GENERATESAMPLE  $\rightarrow$  CONSTRUCTHIERARCHY  $\rightarrow$  MERGEHIERARCHIES  $\rightarrow$  ...). Each iteration first intelligently generates a sample  $\mathcal{S} \in \mathcal{D}$  for workers to cluster. Note that  $|\mathcal{S}| \ll |\mathcal{D}|$  to limit cognitive load. We then use worker clusterings on  $\mathcal{S}$  to construct a new hierarchy and merge this hierarchy into our previous estimate of  $\mathcal{T}_{ml}(\mathcal{D})$  to update  $\mathcal{T}_{ml}(\mathcal{D})$ . A description of this workflow is given below.

- **GENERATESAMPLE.** This algorithm generates samples of items for workers to cluster. Any sample of items that we generate must contain some item overlap with previously generated samples, as well as contain new items from the rest of 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 or consensus hierarchy. To separate the dominant hierarchy, HIERARCHYCONSTRUCTION infers whether the worker provided frontiers are drawn 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 — and outputs a new estimate of  $\mathcal{T}_{ml}(\mathcal{D})$ . The merging exploits the placement 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 *at least* some fraction of items in  $\mathcal{D}$ .

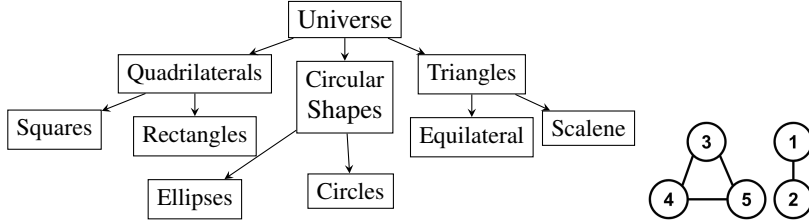
In the subsequent sections, we describe the details of different components in the ORCHESTRA workflow. In Section 2.3, we describe the HIERARCHYCONSTRUCTION algorithm. Thereafter, in Section 2.4, we provide details of GENERATESAMPLE and MERGINGHIERARCHIES, closing the iterative loop of the clustering phase. We also provide theoretical results that allow us to limit the number of iterations in the ORCHESTRA workflow. Finally, in Section 2.5, we examine the categorization phase of the workflow.

## 2.3 Hierarchy Construction

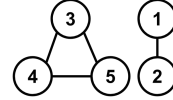
As we noted in the previous section, the goal of the clustering phase is to recover  $\mathcal{T}_{ml}(\mathcal{D})$ , the most likely hierarchy. In this section, we describe the HIERARCHYCON-

1:	Red	Pink	Azure Blue	Dark Blue	Olive Green	Dark Green	} Colors
2:	Red	Pink	Blue Shades	Green Shades			
3:	Squares	Rectangles	Ellipses	Circles	Scalene Triangles	Equilateral Triangles	} Shapes
4:	Quadrilaterals	Ellipses	Circles	Scalene Triangles	Equilateral Triangles		
5:	Quadrilaterals	Circular Shapes	Triangles				

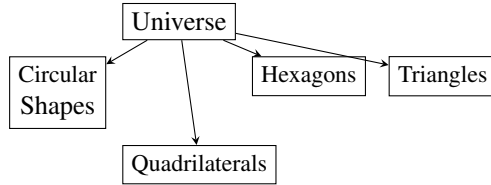
(a) Examples of real worker clusterings for the dataset in Figure 2.2.



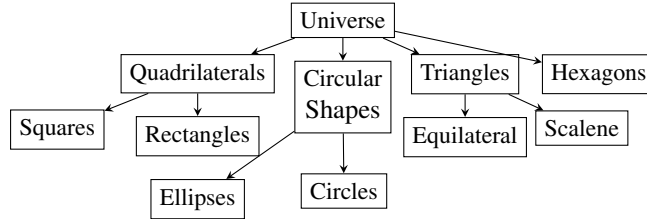
(b) The hierarchy  $\mathcal{T}$  corresponding to the maximum sized clique 3, 4, 5 in (c) using CONSTRUCTHIERARCHY.



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



(d) A hypothetical hierarchy  $\mathcal{T}(S)$  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 2.5: An example demonstrating our iterative workflow approach on the Shapes dataset of Figure 2.1(f).

STRUCTION algorithm that aims to construct the hierarchy  $\mathcal{T}_{ml}(S)$  for a small sample of items  $S \subseteq \mathcal{D}$ . Thereafter, in Section 2.4, we provide algorithms that extends this hierarchy to get an estimate of  $\mathcal{T}_{ml}(\mathcal{D})$ .

### 2.3.1 Mapping to MAX-CLIQUE

Given a set of items  $S \subseteq \mathcal{D}$ , we ask  $m$  workers to cluster the items in  $S$ . We denote the set of worker clusterings by  $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_m\}$ , where  $\mathcal{C}_i = \{C_{i,1}, \dots, C_{i,k_i}\}$  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 2.5(a) shows some clusterings proposed by workers on the sample of items shown in Figure 2.2.

**Problem 1 (Hierarchy Construction).** *Given clusterings  $\mathcal{C}$  of  $\mathcal{S}$ , find a hierarchy  $\mathcal{T}(\mathcal{S})$  that maximizes the number of clusterings in  $\mathcal{C}$  that are associated with complete frontiers in  $\mathcal{T}(\mathcal{S})$ .*

We will show that Problem 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-COMplete problem. Consequently, the optimal solution is computationally intractable for large graphs. However, in our case the graph for which MAX-CLIQUE must be solved is small, so the computation is feasible. We show the equivalence to MAX-CLIQUE below, first defining a notion of *consistency* that allows us to create a graph from our clusterings.

**Definition 5 (Consistency of Clusterings).** *Clusterings  $\mathbb{C}_i = \{C_{i,1}, \dots, C_{i,k_i}\}$  and  $\mathbb{C}_j = \{C_{j,1}, \dots, C_{j,k_j}\}$  are said to be consistent if and only if for every  $(s, t) \in \{1, \dots, k_i\} \times \{1, \dots, k_j\}$ ,  $C_{i,s} \cap C_{j,t} \neq \emptyset \implies C_{i,s} \subseteq C_{j,t} \vee C_{i,s} \supset C_{j,t}$*

Intuitively, two clusterings are consistent if every concept in one either generalizes or is a generalization of some concept in the other. In Figure 2.5(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 corresponding notion of *consistent frontiers*: we simply replace  $\supset$  by  $\succ$  in Definition 5. It is useful to note that any two complete frontiers in the same hierarchy will always be consistent. In Figure 2.5(b), the complete frontiers {Quadrilaterals, Ellipses, Circles, Triangles} and {Squares, Rectangles, Circular Shapes, Triangles} are consistent.

Next, we define a graph of clusterings using the above consistency relation that will allow us to map Problem 1 to MAX-CLIQUE on this graph.

**Definition 6 (Clustering Graph).** *The clustering graph  $G_{\mathcal{C}}$  is an undirected graph, where each clustering in  $\mathcal{C}$  corresponds to a unique vertex in  $G_{\mathcal{C}}$  and there is an edge between  $\mathbb{C}_i$  and  $\mathbb{C}_j \forall i, j \in \{1, \dots, m\}$  if and only if  $\mathbb{C}_i$  and  $\mathbb{C}_j$  are consistent.*

Figure 2.5(c) depicts the clustering graph for the clusterings shown in Figure 2.5(a). Notice that there is no edge from 1 to any of 3, 4, 5, due to pairwise inconsistency between the corresponding clusterings.

We now show that a set of clusterings can be organized into a hierarchy if and only if they form a clique in the clustering graph. Note that if the clusterings do not form a clique, then any pair of clusterings that do not share an edge can not be consistent and hence they can not represent complete frontiers from the same hierarchy. On the other hand, let  $\mathcal{C}_{\text{CLIQUE}} \subseteq \mathcal{C}$  be a clique in  $G_{\mathcal{C}}$ . Let the set of all *unique* clusters in  $\mathcal{C}_{\text{CLIQUE}}$  be  $\mathcal{H} = \{C_{i,j} \mid C_{i,j} \in \mathbb{C}_i, \forall \mathbb{C}_i \in \mathcal{C}_{\text{CLIQUE}}\}$ . We show that  $\mathcal{H}$  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

$\mathcal{H} \cup \text{Universe}$  that is a superset of  $C_{i,j}$  and mark that as the parent of  $C_{i,j}$  in  $\mathcal{T}_{\mathcal{H}}$ . Algorithm 4 shows the pseudocode for this HIERARCHYCONSTRUCTION algorithm.

Consider the clique 3, 4, 5 in the clustering graph of Figure 2.5(c).  $\mathcal{H}$  contains 9 unique (of 14 total) clusters as shown in Figure 2.5(a). Suppose we wanted to find the parent of `Rectangles`; the smallest cluster in  $\mathcal{H} \cup \text{Universe}$  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 2.5(b). Similarly, `Universe` becomes the parent of `Quadrilaterals`. The hierarchy after this construction is shown in Figure 2.5(b).

---

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

---

**Require:** Set of clusters  $\mathcal{H}$   
**Ensure:** Hierarchy  $\mathcal{T}_{\mathcal{H}}$   
 $\mathcal{T}_{\mathcal{H}}(\mathcal{V}) \leftarrow \{\text{Universe}\} \cup \mathcal{H}$   
**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**

---

The following lemma and theorem show that our construction is valid.

**Lemma 1.** *For any  $C_{i,j} \in \mathcal{H}$ , the smallest cluster in  $\mathcal{H} \cup \text{Universe}$  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  $\mathcal{H} \cup \text{Universe}$ . 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 5. 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.  $\square$

**Theorem 1.**  *$\mathcal{T}_{\mathcal{H}}$  is a hierarchy.*

*Proof.* By Lemma 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  $\{C_1, \dots, C_k\}$  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)  $C_i \cap C_j = \phi \ \forall i \neq j \in \{1, \dots, k\}$  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 seen in any child of  $C$ .

Let  $\mathbb{C}_1, \dots, \mathbb{C}_k$  be clusterings that contain  $C_1, \dots, C_k$  respectively. Each  $\mathbb{C}_i$  contains at least  $C_i$ .  $C$  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  $\mathbb{C}_1, \dots, \mathbb{C}_k$ . Since  $C_{\text{large}}$  is the largest such cluster, its parent — from our construction — cannot lie in  $\mathbb{C}_1, \dots, \mathbb{C}_k$ . 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.  $\square$

We can now pick  $\mathcal{C}_{\text{CLIQUE}}$  to be the maximum sized clique in  $G_{\mathcal{C}}$ . Denote by  $\mathcal{T}_{\text{max}}$ , the hierarchy that is generated using this clique. It is easy to see that  $\mathcal{T}_{\text{max}}$  is a solution to Problem 1. We also note that since the size of  $G_{\mathcal{C}}$  is at most  $m$  (which is small as discussed later on in this section), solving MAX-CLIQUE is feasible. Figure 2.5(b) shows the maximum likelihood hierarchy constructed for the maximum clique 3, 4, 5 in Figure 2.5(c).

### 2.3.2 Worker Responses

Having proved the equivalence of Problem 1 to MAX-CLIQUE, we introduce the notion of a hierarchy’s affinity to motivate the probabilistic guarantees that  $\mathcal{T}_{\text{max}}$  provides.

**Definition 7** (Affinity of a hierarchy). *The affinity  $p$  of a hierarchy  $\mathcal{T}$  is the a-priori probability that a worker picks some frontier from  $\mathcal{T}$  while performing clustering.*

We assume that the affinities are independent of the samples seen by workers. Our goal is to construct the hierarchy with greatest affinity (the *most likely hierarchy*  $\mathcal{T}_{ml}$ ). We now concretize the connection between  $\mathcal{T}_{ml}$  and our solution  $\mathcal{T}_{\text{max}}$  to Problem 1.

**Lemma 2.** *Assume the existence of exactly  $k$  latent hierarchies  $\mathcal{T}_1, \dots, \mathcal{T}_k$  with affinities  $p_1 > \dots > p_k$ . Suppose that the hierarchies do not share any frontiers. Let  $\hat{p}_1, \dots, \hat{p}_k$  be the maximum likelihood estimates of the affinities. If  $\hat{p}_1 > \hat{p}_i \forall i$ ,  $\mathcal{T}_{\text{max}}$  (the maximum likelihood hierarchy) has maximum affinity.*

*Proof.* Notice that every maximal clique in  $G_{\mathcal{C}}$  will correspond to a single, distinct hierarchy. Let  $m_1, \dots, m_k$  be the sizes of the maximal cliques corresponding to  $\mathcal{T}_1, \dots, \mathcal{T}_k$ . If some hierarchy has no associated maximal clique, the corresponding value of  $m_i$  would be 0.

Since every clustering lies in exactly one maximal clique of size  $m_i$ , each worker’s clustering can be viewed as a vote for that hierarchy  $\mathcal{T}_i$ . We can then define a multinomial distribution  $(m, p_1, \dots, p_k)$  that captures worker tendency to pick a particular latent hierarchy.

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

$$\mathcal{L} \propto p_1^{m_1} p_2^{m_2} \dots p_k^{m_k}$$



It is easy to show that the maximum likelihood solution is simply  $\hat{p}_i = \frac{m_i}{m}$ . If  $\hat{p}_1 > \hat{p}_i \forall i$ , i.e.,  $m_1 > m_i \forall i$ , then the largest clique corresponds to  $\mathcal{T}_{max} = \mathcal{T}_{\arg \max_i \hat{m}_i} = \mathcal{T}_{\arg \max_i \hat{p}_i} = T_1$   $\square$

The lemma above shows that the convergence of maximum likelihood estimates  $\hat{p}_i$  to the true values  $p_i$  is a sufficient condition for  $\mathcal{T}_{max}$  to be  $\mathcal{T}_{ml}$ . We will thus refer to  $\mathcal{T}_{max}$  as the maximum likelihood hierarchy for the rest of the chapter. Thus, there is a clear dependence on the number of workers  $m$ ; a large  $m$  would cause the estimates to converge. We now address a natural question: how do we fix  $m$ ?

Intuitively, increasing  $m$  leads to an increase in two probabilities of interest: (a) the probability of discovering the most likely hierarchy *i.e.*, the probability that there exists a clique corresponding to  $\mathcal{T}_{ml}$ , and (b) the probability that  $\mathcal{T}_{max}$  is  $\mathcal{T}_{ml}$ .

To see this dependence for (b), we randomly subsample the worker responses on our shapes dataset (giving us different  $m$  values) and plot (in Figure 2.6) the fraction of trials (out of 200) where the maximum clique corresponded to the dominant SHAPE perspective. Since shapes is a stylized dataset with a clear dominant perspective and unambiguous items,  $\Pr(\mathcal{T}_{max} = \mathcal{T}_{ml} | m \geq 12) = 1$ . Therefore,  $m = 15$  seems a reasonable conservative choice.

However, for general datasets,  $\Pr(\mathcal{T}_{max} = \mathcal{T}_{ml} | m)$  is extremely hard to characterize because (i) the number of latent hierarchies is *unknown*, (ii) their corresponding affinities are *also unknown*, and (iii) a worker clustering may be a frontier in multiple latent hierarchies (seen as nodes belonging to multiple maximal cliques in  $G_{\mathcal{E}}$ ). We note an interesting connection: on relaxing (i) and (iii) our problem becomes identical to the *multinomial selection problem* (see [21]). We utilize a result from [21] in Theorem 3 as well as a simple guarantee for (a) in Theorem 2, to show that  $m = 15$  guarantees high probability of both discovering  $\mathcal{T}_{ml}$  and having  $\mathcal{T}_{ml} = \mathcal{T}_{max}$ , and fix this value for the purpose of this chapter.

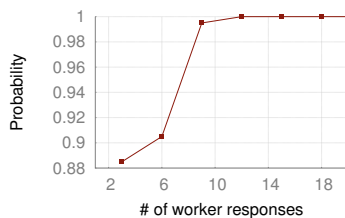


Figure 2.6: Variation in the probability of ORCHESTRA providing a clustering based on SHAPE with the number of worker responses

**Theorem 2.** *Suppose that the number of latent hierarchies is exactly  $k$  and these hierarchies do not share any frontiers. Then the hierarchy with greatest affinity corresponds to one of the cliques with probability at least  $[1 - (1 - \frac{1}{k})^m]$ .*

*Proof.* Since there are  $k$  latent hierarchies, the most likely hierarchy must have probability at least  $\frac{1}{k}$  of being discovered, since otherwise it could not be the most likely hierarchy. The probability that this hierarchy goes undiscovered after  $m$  worker responses is upper bounded by  $(1 - \frac{1}{k})^m$ , and the result follows.  $\square$

**Corollary 1.** For  $k = 4$  and  $m = 15$ , the hierarchy with greatest affinity is discovered with probability at least 0.98.

**Theorem 3** (see [21]). Let  $k = 4$  with affinities  $p_1 > \dots > p_4$  and assume that these hierarchies do not contain identical frontiers. Then,  $\Pr(\mathcal{T}_{max} = \mathcal{T}_{ml} | m = 15) \geq 0.5$  if  $\frac{p_1}{p_2} \geq 1.5$ .

*Proof.* See [21]. □

These theorems suggest that  $m = 15$  guarantees high probability of discovering and constructing  $\mathcal{T}_{ml}$  when the underlying assumptions are met. While these assumptions may not always hold, we examine the effect of varying  $m$  in Section 2.6.2 and find that in practice this value of  $m$  turns out to be sufficient.

### 2.3.3 Worker Mistakes

So far, we have assumed that workers do not make mistakes by associating items with the wrong concepts and therefore placing them in the wrong cluster either due to item ambiguity or worker fatigue. However, due to the presence of ambiguous items or human error, workers may end up placing items in incorrect clusters. For instance, a worker could add a circular shape to a cluster that was meant to represent triangles. In this situation, our ‘hard’ definition of consistency may lead to the maximum sized clique being small since not too many clusterings will be consistent. In such cases, we can relax our consistency relationship to tolerate slight worker errors in order to discover larger cliques. One such relaxation is to remove a small number of ambiguous or difficult items from the sample, whose presence causes worker answers to become inconsistent. In particular, we examine the problem of determining the minimum set of items whose removal makes the clusterings of two given workers consistent (by Definition 5).

**Problem 2 (2-CONSISTENCY).** Given two worker clusterings  $\mathcal{C}_1$  and  $\mathcal{C}_2$  on a set of items  $\mathcal{S}$ , find a minimum sized subset  $\mathcal{I} \subset \mathcal{S}$  such that  $\mathcal{C}_1$  and  $\mathcal{C}_2$  are consistent on  $\mathcal{S} - \mathcal{I}$ .

**Theorem 4.** 2-CONSISTENCY is NP-HARD.

We show this hardness via a reduction from the NP-HARD problem of finding the *smallest dominating set in bipartite graphs* [22]. The details of the reduction are available below.

We divide the proof into two parts. First, we show that finding smallest dominating set in bipartite graphs is equivalent to another problem, MINSSFCUT-BIPARTITE (described below). Thereafter, we reduce MINSSFCUT-BIPARTITE to 2-CONSISTENCY. We first define a special type of graph, spanning star forest (SSF) below.

**Definition 8 (Spanning Star Forest).** A *spanning star forest (SSF)* of a graph  $G$  is a spanning subgraph  $G_s$  of  $G$  where each connected component is a star. We define the cardinality of a star forest  $G_s$  as the number of connected components in  $G_s$ .

Intuitively, any graph can be converted into a spanning star forest by removing some edges. In the trivial case, when all edges in a graph are removed, each vertex forms a star component. We define such a set of edges whose removal makes the graph a spanning star forest as a SSF-cut.

**Definition 9** (SSF-cut). *A SSF-cut of a graph  $G$  is a set of edges in which their removal results in a SSF.*

We now formalize the problem of finding the smallest sized SSF-cut in any bipartite graph as the MINSSFCUT-BIPARTITE problem.

**Problem 3** (MINSSFCUT-BIPARTITE). *Given a bipartite graph  $G$ , find the smallest SSF-cut of  $G$ .*

Next, we show that MINSSFCUT-BIPARTITE is NP-HARD. This is done by proving that it is equivalent to the NP-HARD problem of finding smallest dominating set in the same graph [22]. This equivalence is shown by the following two lemmas. They together show that a bipartite graph  $G$  has a dominating set of size  $k$  if and only if it has an SSF-cut of size  $k - |V(G)| + |E(G)|$ .

**Lemma 3.** *A graph  $G$  has a dominating set of size  $k$  if and only if  $G$  has a SSF of cardinality  $k$ .*

*Proof.* Each element of the dominating set can be the center of a star. Conversely, the centers of the stars form a dominating set.  $\square$

**Lemma 4.** *A graph  $G$  has a SSF-cut of size  $k$  if and only if  $G$  has a SSF of cardinality  $|V(G)| - |E(G)| + k$ .*

*Proof.* The result follows immediately from the fact that a SSF of cardinality  $n$  has  $|V(G)| - n$  edges.  $\square$

The above lemmas show that an algorithm that finds the smallest SSF-cut for a bipartite graph would also correctly determine the size of the smallest dominating set in that graph. We conclude that MINSSFCUT-BIPARTITE is NP-HARD.

**Lemma 5.** MINSSFCUT-BIPARTITE is NP-HARD.

The above lemmas together prove the hardness of MINSSFCUT-BIPARTITE. Next, we prove that 2-CONSISTENCY is NP-HARD by reducing MINSSFCUT-BIPARTITE to it.

Given a bipartite graph  $G$  having disjoint sets of vertices  $\mathbf{V}_1 = \{V_{1,1}, \dots, V_{1,d_1}\}$  and  $\mathbf{V}_2 = \{V_{2,1}, \dots, V_{2,d_2}\}$ . Define two clusterings  $\mathcal{C}_1$  and  $\mathcal{C}_2$ , one each for  $\mathbf{V}_1$  and  $\mathbf{V}_2$ . For every vertex in  $V_{1,i} \in \mathbf{V}_1$ , add a cluster  $C_{1,i}$  in  $\mathcal{C}_1$ . Repeat for  $\mathcal{C}_2$ . For every edge  $(V_{1,a}, V_{2,b})$ , add a new item to both  $C_{1,a}$  and  $C_{2,b}$ .

**Lemma 6.**  *$G$  has an SSF-cut of size  $k$  if and only if there exists a set of  $k$  items whose removal makes  $\mathbb{C}_1$  and  $\mathbb{C}_2$  consistent.*

*Proof.* We first note that each item in the clusterings can be associated with a unique edge in  $G$ , by construction.

$\Leftarrow$  Assume that there exists a set of  $k$  items whose removal makes the clusterings consistent. Fix any one such set. From  $G$ , remove the  $k$  edges corresponding to these items. We claim that after this removal,  $G$  is a SSF. On the contrary, assume that  $G$  is not SSF. Then there exists a path of length  $\geq 3$  in  $G$ , say, WLOG,  $(V_{1,a}, V_{2,b}, V_{1,c}, V_{2,d})$ . For the corresponding clusters, we have:

1.  $C_{1,a} \cap C_{2,b} \neq \emptyset$
2.  $C_{2,b} \cap C_{1,c} \neq \emptyset$ ,
3.  $C_{1,c} \cap C_{2,d} \neq \emptyset$

Since  $C_{2,b} \cap C_{1,c} \neq \emptyset$  and the clusterings are consistent (after the removal of items), that means that either  $C_{2,b} \subset C_{1,c}$  or  $C_{1,c} \subset C_{2,b}$  or  $C_{2,b} = C_{1,c}$ . In either of these cases, both (1) and (3) can not be true simultaneously without violating consistency.

$\Rightarrow$  Assume that  $G$  has an SSF cut of size  $k$ . Take any such cut arbitrarily and remove the items corresponding to the edges in this SSF-cut. We claim that the clusterings are consistent after the removal of these items. On the contrary, assume that the clusterings are inconsistent. This means that  $\exists C_{1,a}, C_{2,b}$  such that

1.  $C_{1,a} \cap C_{2,b} \neq \emptyset$
2.  $C_{1,a} \setminus C_{2,b} \neq \emptyset$
3.  $C_{2,b} \setminus C_{1,a} \neq \emptyset$

Since  $C_{1,a} \setminus C_{2,b} \neq \emptyset$ , there must be some other cluster  $C_{2,t}$  apart from  $C_{2,b}$  that has a non-empty intersection with  $C_{1,a}$ . Recall that both clusterings organize the same set of items and thus the items in  $C_{1,a} \setminus C_{2,b}$  must appear in some other cluster in  $\mathbb{C}_2$ . Similarly, there must be some other cluster  $C_{1,s}$  apart from  $C_{1,a}$  that has a non-empty intersection with  $C_{2,b}$ . Consider the vertices  $(V_{2,t}, V_{1,a}, V_{2,b}, V_{1,s})$ . Consecutive vertices in this tuple must share an edge because the corresponding clusters have non-empty intersections. If the consecutive vertices share an edge, then the graph is not an SSF (the SSF-cut has already been removed). This leads to a contradiction and therefore, the clusterings must be consistent after removing the items corresponding to the SSF cut.  $\square$

■

A natural extension of 2-CONSISTENCY is to determine the smallest set of items whose removal makes the clusterings of  $N$  workers pairwise consistent *i.e.*, they form a clique in  $G_{\mathcal{C}}$ .

**Corollary 2.** N-CONSISTENCY is NP-HARD.

Given the hardness of these problems, we defer the development of algorithms to handle worker mistakes and ambiguous items to future work. In our experiments, we find that worker mistakes with respect to their chosen perspective and granularity are very infrequent. Even if some workers do make mistakes, 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. Thus, for the purposes of this chapter and the datasets we consider, we use Definition 5 for consistency.

## 2.4 Extending the Hierarchy

In the previous section, we explained the HIERARCHYCONSTRUCTION algorithm that estimates the hierarchy  $\mathcal{T}_{ml}(\mathcal{S})$  for a sample  $\mathcal{S}$ . In this section, we describe how to extend this hierarchy to  $\mathcal{T}_{ml}(\mathcal{D})$ . First, in Section 2.4.1, we provide a result that allows us to determine  $|\mathcal{S}|$  that guarantees that  $\mathcal{T}_{ml}(\mathcal{S})$  is an accurate representation of  $\mathcal{T}_{ml}(\mathcal{D})$ . When  $|\mathcal{S}|$  is large, the corresponding cognitive load on workers may necessitate splitting this large sample into smaller samples. Section 2.4.2 provides details of our algorithm GENERATESAMPLE that generates different samples for workers to cluster. Thereafter, in Section 2.4.3, we describe an algorithm that allows us to merge hierarchies  $\mathcal{T}_{ml}(\mathcal{S})$  obtained for different samples  $\mathcal{S}$ .

### 2.4.1 Sampling Guarantee

As discussed in Section 2.2, our approach is to first construct an estimate  $\mathcal{T}_{max}$  of the most likely hierarchy  $\mathcal{T}_{ml}$  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  $n$  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  $n$  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 10 (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 11 (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.*

Intuitively, frontier coverage is the fraction of items in the dataset that are *covered* by the concepts in the frontier.

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

*Proof.* Let  $p_i$  be the coverage of the  $i^{\text{th}}$  concept in  $F$ . Let  $X_{F,i}$  be a random variable that equals 0 if  $\mathcal{S}$  does not discover the  $i^{\text{th}}$  concept of  $F$  and equals  $p_i$  otherwise. Using

Definition 11, 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 2.7(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 2.7(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 6.**  $\text{Var}_{\mathcal{S}}[X_F] \leq 1 - \left(1 - \frac{f}{n+1} \left(1 - \frac{1}{n+1}\right)^n\right)^2$

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.

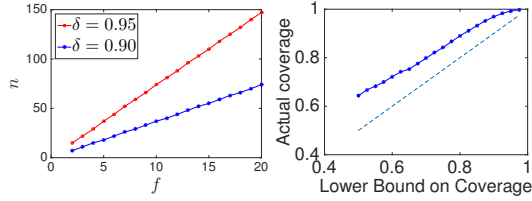


Figure 2.7: (a) Plot showing sample size  $n$  vs. size of complete frontier  $f$  to ensure  $\delta$  expected coverage. (b) Plot showing lower bound on expected coverage vs. actual expected coverage over 1000 trials – the dotted line is the 45 deg line.

## 2.4.2 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 6 shows the pseudocode for the GENERATESAMPLE algorithm.

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.

---

**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**  $\mathcal{S}$

---

### 2.4.3 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 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  $C_1, \dots, C_l$  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 2.5 demonstrates an example of MERGINGHIERARCHIES. Figure 2.5(c) is our current estimate  $\mathcal{T}$  to be merged with Figure 2.5(d), depicting  $\mathcal{T}(\mathcal{S})$ . The merged



hierarchy  $\mathcal{T}'$  is shown in Figure 2.5(e). By our GENERATESAMPLE algorithm, the kernel of  $\mathcal{S}$  would contain 6 items, one each for the leaves of  $\mathcal{T}$  in Figure 2.5(c). Even though  $\mathcal{T}(\mathcal{S})$  combines the kernel items corresponding to Squares and Rectangles into the Quadrilaterals cluster in Figure 2.5(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 2.5(e).

---

**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

```

---

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

**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 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  $O\left(m \left\lceil \frac{(n-f)}{(h-f)} \right\rceil\right)$ . The cost of clustering phase is *independent of the*

size of the dataset  $\mathcal{D}$ .

## 2.5 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 to carry out categorization. As we stated in Section 2.2, the reason we construct this hierarchy is to preserve information about the granularities at which workers cluster items. We now describe a procedure to determine the consensus clustering by finding the maximum likelihood complete frontier in  $\mathcal{T}$ . We now outline a procedure to find this frontier.

**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 concepts below  $v$  in their frontier. Let  $F$  be a frontier in  $\mathcal{T}$  and  $A$  be the set of ancestors of  $F$ , excluding  $R$ . Also define  $v(1), \dots, R$  to be the ancestors of  $v$  where  $v(1)$  is the parent of  $v$ ,  $v(2)$  is the parent of  $v(1)$ , etc. First observe that  $p(E_v | E_{v(1)}, \dots, E_R) = p(E_v | E_{v(1)})$  *i.e.*  $E_v$  is conditionally independent of the rest of its ancestors, given its parent. We now define the likelihood of  $F$  as,

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

Given a set of worker responses, 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  $\operatorname{argmax}_F \mathcal{L}(F)$  which can be computed using the recurrence relation  $D(v) = \max \left( p(\overline{E}_v | E_{v(1)}), p(E_v | E_{v(1)}) \prod_{u \in C(v)} D(u) \right)$ , 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 2.2 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  $O(\theta|\mathcal{D}|)$ , linear in the size of the dataset.

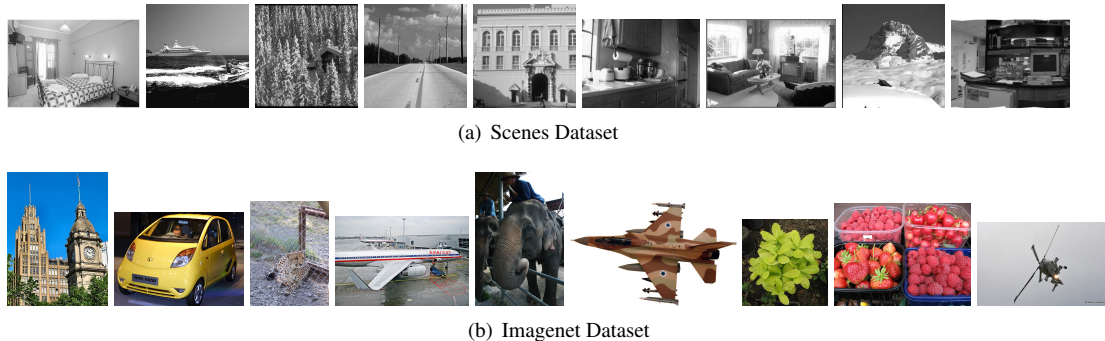


Figure 2.8: Sample Images from the Datasets

## 2.6 Experiments

In our experiments, our goals are to (i) compare ORCHESTRA to other crowd-powered clustering algorithms for organizing data, and (ii) examine the effects of ORCHESTRA’s stages and algorithms.

**Datasets.** We used three datasets in our experiments.

- Our first dataset, titled **shapes**, is a synthetic dataset consisting of 25 shapes, from Section 2.2. As described, each item has a random assignment of shape, size, and color. The images from this dataset include the ones displayed in Figure 2.2. Despite being stylized, we use this dataset since we can control the underlying organizational perspectives, 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 1025 images from 13 categories [23] in natural and man-made surroundings. This dataset has also been used in prior work on crowd clustering [18, 20].
- The third dataset, titled **imagenet**, contains images from ImageNet [24]. Images are sampled randomly from: {buildings, cars, parrots, vulture, fruit, flower, vegetable, fighter, commercial, helicopter, ship, seahorse, whale, cheetah, lion, elephant, tiger, jellyfish, sparrow, leaves}. For consistency with **scenes**, we sample 1025 images from these categories.

Sample images from scenes and imagenet are shown in Figure 2.8.

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

- **CrowdClust**: This is the algorithm from Gomes et al. [18].
- **MatComp**: This is the algorithm from Yi et al. [20].

Code for both these algorithms was provided by the authors; we faithfully set the parameters as described in the papers. Each clustering HIT for these algorithms was repeated by 5 workers – exactly as reported in the respective papers. For MatComp, we found that the recommended value of parameter  $C = \frac{1}{\sqrt{N}}$  fails to return clusterings because the item co-clustering matrix is extremely sparse at our level of sampling. In-

stead of not reporting results, we tuned this value until it reliably returned results. The value was  $\frac{1}{N^{\frac{1}{6}}}$ .

Both these algorithms require workers to cluster random samples of items repeatedly, while ensuring that no item is repeated in a task.

**Worker Responses.** We used Amazon’s Mechanical Turk to gather worker responses for all tasks. Each task asked the workers to organize a collection of at most 35 images. For ORCHESTRA, the categorization task used 10 images from each consensus cluster as *pivots* for categorization. Workers were paid 20 cents for a clustering task and 10 cents for a categorization task, since the categorization tasks are considerably simpler. (We consider the impact of this cost ratio subsequently.)

For each dataset, the clustering phase of ORCHESTRA organizes a fixed number of images  $n = 115$  clustering tasks. For the `scenes` and `imagenet`, each of these tasks was performed by 15 workers. As noted in Section 2.3, to study the effect of  $m$  on the clustering, `shapes` had 20 workers repeating each task. The kernel size is taken to be 15, as discussed in Section 2.4.2. Thus,  $\frac{115-35}{35-15} + 1 = 5$  clustering HITs were sufficient for this phase. In the next phase, ORCHESTRA organized the remaining  $1025 - 115 = 910$  items using categorization tasks. This required a total of  $\lceil \frac{910}{35} \rceil = 26$  categorization tasks, with each categorization task being repeated by 5 workers.

**Evaluated Aspects.** We divide the experimental evaluation into two parts (i) a head-to-head comparison of the various algorithms, and (ii) evaluation of internal components of ORCHESTRA.

#### Head-to-Head Comparison

- *Quality*: For a fixed total cost, how does the quality of clusterings provided by the different algorithms differ?
- *Cost*: How much cost do different algorithms need to reach a fixed level of quality?

#### Component Evaluation

- *Perspectives Disambiguation*: How well is ORCHESTRA’s HIERARCHYCONSTRUCTION component able to differentiate between different worker perspectives?
- *Sampling*: What is the benefit of GENERATESAMPLE’s intelligent sampling compared to random sampling?
- *Categorization*: What is the impact, on cost and quality, of the categorization interface?
- *Robustness*: How robust is ORCHESTRA with respect to variations in worker responses and item sampling?

**Metrics.** To quantitatively compare ORCHESTRA with prior work, we adopt five metrics. Note that prior work has considered only a subset of these metrics. We consider the additional metrics to be part of our contributions.

- *Precision (Pr)* and *Recall (Re)*. For every pair of distinct items that are clustered together by the algorithm, precision measures the fraction that are also clustered together in the ground truth, while recall measures the fraction of pairs that are clustered together by the algorithm, out of pairs of items that are clustered together

by the ground truth. Let  $\mathcal{G}$  be the set of item pairs that are clustered together by the ground truth, and let  $\mathcal{A}$  be the set of item pairs that were clustered together by the algorithm. Then

$$\text{Pr} = \frac{|\mathcal{G} \cap \mathcal{A}|}{|\mathcal{A}|} \quad \text{Re} = \frac{|\mathcal{G} \cap \mathcal{A}|}{|\mathcal{G}|} \quad (2.1)$$

Note that the metrics of precision and recall penalize any deviation from the ground truth, even if these deviations are meaningful. Therefore, an algorithm that returns a coarser granularity clustering (where the clusters correspond to unions of ground truth clusters) would have high recall but low precision. Further, recall is more critical in clustering than precision. If the recall of a coarse clustering is high, a finer granularity of clusters can be discovered in a hierarchical manner by asking the workers to cluster only the items belonging to a coarse cluster. These finer granularity clusters can further improve precision.

- *Accuracy (Ac)*. To avoid unnecessarily penalizing algorithms for identifying meaningful but coarser granularity clusters, we introduce another metric that measures the *accuracy* of an algorithm’s clustering with respect to the closest *coarse* clustering. Here, coarse clusterings are groupings of ground truth clusters. For example, given the ground truth clustering (*living room, bedroom, kitchen*), some possible coarse clusterings are (*living room, bedroom, kitchen*), (*living room, non-living room*), (*bedroom, non-bedroom*), (*kitchen, non-kitchen*) and (*home areas*). To find the closest coarse clustering, consider a ground truth clustering  $\mathcal{G}$  consisting of clusters  $G_i$  indexed by  $i$ . Let the clustering provided by the algorithm to be evaluated be  $\mathcal{A}$  comprising of clusters  $A_j$  indexed by  $j$ . Map every ground truth cluster  $G_i$  to some cluster  $A_{[i]}$  such that the number of items in  $G_i$  that are common with  $A_{[i]}$  is maximized. The ground truth clusters that get mapped to the same  $A_j$  are part of the same cluster in the closest coarse clustering. The mapping from clusters in  $\mathcal{G}$  to clusters in  $\mathcal{A}$  thus defines the closest coarse clustering. The accuracy is then defined as the fraction of items that are placed in the correct cluster based on this coarse clustering.

$$Ac = \frac{\sum_i \max_{[i]} |G_i \cap A_{[i]}|}{\sum_i |G_i|} \quad (2.2)$$

The above metrics are all defined on a scale of  $[0, 1]$ , where the ground truth clustering would have Pr, Re and Ac all equal to 1. These metrics are useful to compare algorithms against a known ground truth. However, for *shapes*, where no ground truth is available, we introduce a new metric:

- *Clustering Hierarchies*, the number of hierarchies that ‘explain’ the clustering returned by the algorithms. This is calculated as the minimum number of features (out of shape, size, color) required by a decision tree to place every item in the right cluster.

Since the algorithms are provided a fixed budget to cluster a given set of items, we have a penalty in each of the above metrics for algorithms that are unable to cluster

some of these items; each unclustered item is treated as its own independent cluster. This captures the fact that we have no information about these items and do not know any other items they are clustered together with. So, our final metric is the number of unclustered items after having consumed the entire budget.

- *Unclustered Items* ( $U_i$ ), the number of items not assigned by the algorithm to any cluster.

### 2.6.1 Head-to-Head Comparison

In this section, we compare all algorithms on (i) quality, keeping the cost fixed, and (ii) cost to achieve a fixed quality.

#### Quality Comparison

In this set of experiments, we ensure that all algorithms use the exact same cost. Since worker payments for categorization were half of that for clustering, ORCHESTRA uses an equivalent of  $5 \times 15 + \frac{1}{2} \times 26 \times 5 = 140$  clustering tasks. For a comparison on even footing, MatComp and CrowdClust are also provided a budget of 140 clustering tasks to organize the same set of items.

**Quantitative Comparison.** We compare the results of ORCHESTRA versus the other two algorithms on `scenes` and `imagenet`. The quantitative results are listed in Table 2.1. We find that ORCHESTRA outperforms both MatComp and CrowdClust on all metrics for the challenging `imagenet` dataset. For instance, ORCHESTRA’s recall of 0.800 is at least  $3\times$  that of MatComp and CrowdClust. Moreover, ORCHESTRA has an accuracy of 0.881 - this is  $1.9\times$  that of MatComp and CrowdClust, which means that ORCHESTRA clusters 90% more items correctly with respect to the closest granularity.

The scenario is similar even on the easier `scenes` dataset, with ORCHESTRA’s recall being as high as 0.985 indicating that ORCHESTRA *rarely assigns items from the same categories into different clusters*. In comparison, CrowdClust and MatComp have recall of only 0.161 and 0.089 respectively, making ORCHESTRA at least  $6\times$  better on recall. In terms of accuracy, ORCHESTRA’s accuracy is more than  $3\times$  of CrowdClust, and  $4.4\times$  of MatComp. The gains in recall and accuracy are not at the cost of precision—ORCHESTRA is close to the maximum precision of MatComp. The slightly higher precision of MatComp is because MatComp uses a much larger number of clusters and therefore has fewer opportunities to be incorrect. As we see in the qualitative comparison described next, MatComp’s organization into a higher number of clusters ends up assigning items from the same ground truth categories to different clusters.

Furthermore, looking at unclustered items in the table, we note that due to random sampling, both CrowdClust and MatComp are unable to cluster more than 37% of the dataset within the budget. In comparison, ORCHESTRA leaves no item unclustered on the `scenes` dataset, and only 1% items for the `imagenet` dataset. For ORCHESTRA, the

	scenes				imagenet			
	Pr	Re	Ac	Ui	Pr	Re	Ac	Ui
ORCHESTRA	0.230	<b>0.985</b>	<b>0.992</b>	<b>0</b>	<b>0.283</b>	<b>0.800</b>	<b>0.881</b>	<b>12</b>
CrowdClust	0.105	0.161	0.319	387	0.115	0.241	0.465	394
MatComp	<b>0.234</b>	0.089	0.226	387	0.072	0.268	0.457	394

Table 2.1: Quantitative Comparison of algorithms. The best values in every column have been highlighted in bold.

unclustered items are those that get assigned to nodes above the maximum likelihood frontier in the MERGINGHIERARCHIES step.

ORCHESTRA’s clustering is at least  $3\times$  better on recall and  $1.9\times$  better on accuracy while maintaining the best precision and organizing 60% more items in the same cost.

**Qualitative Comparison.** Next, we qualitatively examine the resulting clusters for the scenes and imagenet datasets via Figures 2.9 and 2.10 respectively. We depict a confusion matrix corresponding to each algorithm, where each ground-truth category corresponds to a row, while each cluster in the result corresponds to a column. Ideally, for each row we want a single column to have non-zero presence, indicating that the entire ground-truth category appears as a whole in some cluster. The items that are not clustered by the clustering algorithm are ignored for the computation of this confusion matrix. On examining Figure 2.9, it is clear that CrowdClust and MatComp have much worse matrices than ORCHESTRA. For instance, the only two rows in ORCHESTRA that have non-zero presence in more than one column are ‘opencountry’ and ‘highway’ (11/13 rows are *good*). Even for these categories, most of the items are assigned to one cluster. On the other hand, for CrowdClust, all rows have non-zero presence in multiple columns. Further, the items belonging to categories ‘livingroom’, ‘forest’, ‘opencountry’, ‘highway’ and ‘street’ are almost evenly split between two clusters (having similar color in two columns). Similarly, for MatComp, all rows have non-zero presence in multiple columns. Besides, it also gives clusters having items from the same categories (cluster #5 and #6), which should have been merged together. Thus, there is a clear benefit to ORCHESTRA in identifying organizational perspectives and treating worker responses accordingly, as opposed to operating on all of them at once.

Furthermore, the clusters provided by ORCHESTRA have clearly associable concepts — cluster 0 consists of indoor categories; cluster 2, of outdoor categories in natural setting; and cluster 1, of outdoor categories involving man-made structures; while there are no discernible concepts corresponding to the clustering of CrowdClust and MatComp. The hierarchy provided by ORCHESTRA is shown in Figure 2.11.

Similar results can be observed on the imagenet dataset, shown in Figure 2.10. Even though this dataset is significantly more challenging, ORCHESTRA is able to identify meaningful clusters – cluster 0 consists of all birds; cluster 2, of edible plant food; and cluster 1 of inedible plants. However, due to the challenging nature of this dataset, we see some rows where multiple columns are colored. Nonetheless, in such cases, ORCHESTRA still places a majority of items from a single category in a single cluster

— in all but 1 rows, there is only one column that is red/orange/yellow. It is also desirable that every cluster returned should have some underlying concept, i.e., at least one ground truth category for which a majority items appear in that cluster. Examining ORCHESTRA’s organization closely, we note that 8/9 clusters have clear representations from some ground truth categories. In contrast, 10/13 clusters provided by CrowdClust have no such representation. In addition, items from the same category appear in as many as 5 clusters. MatComp’s clustering is even worse – it returns just two clusters with 11/20 ground truth categories appearing almost evenly across both clusters (having similar color in both columns).

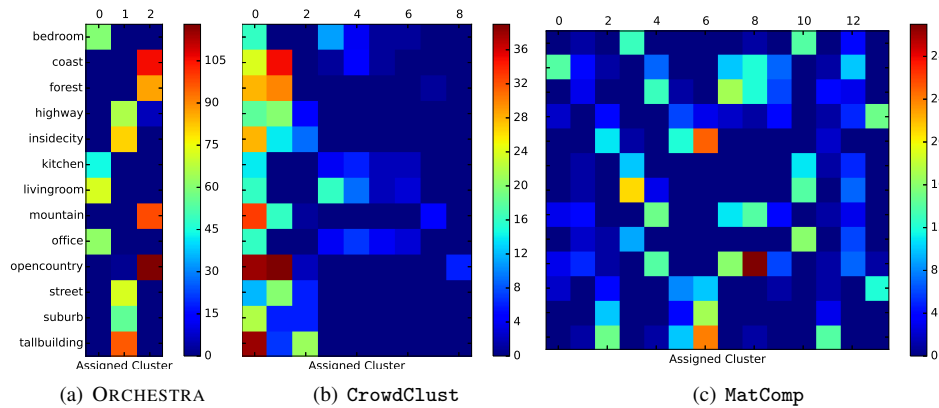


Figure 2.9: Qualitative Comparison of clusters provided by different algorithms on scenes

*ORCHESTRA’s clusters show clear underlying concepts making the whole organization understandable. The clusters are accurate with most items in every category appearing together in a cluster.*

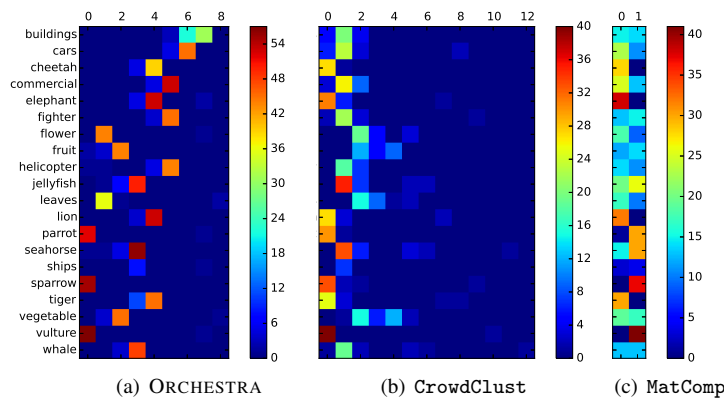


Figure 2.10: Qualitative Comparison of Clusters provided by different algorithms on imagenet

**Human Evaluation.** We also asked workers on Amazon Mechanical Turk to evaluate the clusterings derived from the three algorithms. In particular, the evaluation task showed clusterings from two algorithms on one of the datasets and asked workers to identify the one that they felt was more coherent — they were told that *coherent* clusterings had similarities between items in the same cluster, while being different



	ORCHESTRA vs. CrowdClust	ORCHESTRA vs. MatComp
scenes	15/20	16/20
imagenet	17/20	15/20

Table 2.2: Comparison of clusterings by workers on Amazon MTurk. The values are the number of workers (out of 20) who preferred ORCHESTRA’s clustering.

from the items in other clusters. Workers were also asked to explain their choice. The results of this evaluation are available in Tables 2.2. Overall, close to 80% of the workers preferred ORCHESTRA’s clustering over other clusterings. In their feedback, most workers stated that ORCHESTRA’s organization was easier to understand — items in a cluster are all alike and distinct from the items in other clusters. A full description of the workers’ feedback can be found in Table 2.4 and 2.5 at the end of this chapter. There is, thus, definite utility in identifying and separating different perspectives.

*Close to 80% of the evaluators prefer ORCHESTRA’s organization over prior work because it is more consistent and easier to understand.*

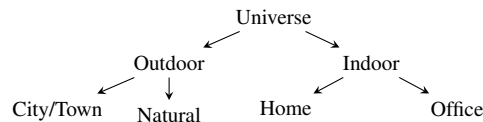


Figure 2.11: Hierarchy constructed by ORCHESTRA on the Scenes dataset

### Cost Comparison

In the previous experiments on quality, we saw that ORCHESTRA outperformed MatComp and CrowdClust. In this experiment, we examine the amount of monetary cost required by MatComp and CrowdClust to match ORCHESTRA on quality. To do this, we simulate clustering HITs beyond the 140 original ones for which worker responses were sought. These simulated HITs were assumed to have received perfect responses from each of the 5 workers repeating this task, *i.e.* each worker provides exactly the ground truth clustering ( $Pr = Re = Ac = 1$ ) on these HITs. These perfect worker responses represent the best-case future scenario for MatComp and CrowdClust and allows us to quantify the cost saving of ORCHESTRA in comparison with this best case for other algorithms.

The results of this simulation for `scenes` and `imagenet` are shown in Figure 2.12. The left, center and right figures show  $Pr$ ,  $Re$  and  $Ac$  respectively. The x-axis for each plot shows the total tasks (real + simulated) relative to the number of real tasks. The left most point on x-axis for each plot represents the performance of algorithms when no simulated data is added. From these plots, we can see that for `scenes`, even with perfect worker responses and  $3\times$  cost, CrowdClust and MatComp are unable to match ORCHESTRA on recall and accuracy. Even on precision, CrowdClust requires an additional  $1\times$  simulated perfect responses to cross this precision. Even after crossing ORCHESTRA’s precision, the clusterings do not show high quality — the recall and accuracy indicate that items in the same ground truth category are split across multiple clusters. We also note that adding worker responses leads to more improvement in

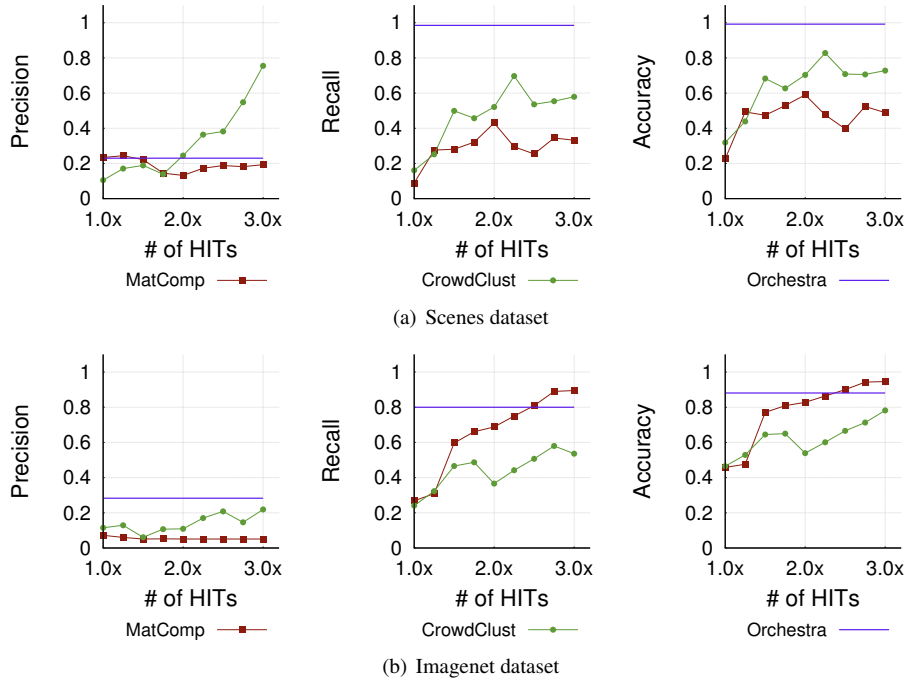


Figure 2.12: Performance of prior work with best-case future worker responses. HITs beyond  $1\times$  are simulated.

CrowdClust than MatComp.

On the challenging imagenet dataset, MatComp returned only one single cluster when the number of HITs went beyond  $1.5\times$ . This grouping of all items together leads to perfect Re and Ac, barring the penalty for unclustered items. However, this trivial clustering isn't informative from a requester's standpoint – it only indicates that all items share a similarity and does not delve into the differences between the items. On the other hand, CrowdClust is unable to match ORCHESTRA on any metric even with  $2\times$  additional HITs with perfect worker responses.

*Even with  $2\times$  additional perfect responses, prior work does not match ORCHESTRA on Re and Ac without returning a trivial organization (having only one cluster).*

Thus, ORCHESTRA is able to provide more accurate and relevant clusterings, while being at least 60% more efficient on cost, as compared to prior work.

## 2.6.2 Component Evaluation

In this part of the experimental evaluation, we examine the components of ORCHESTRA in detail (i) HIERARCHYCONSTRUCTION, (ii) GENERATESAMPLE, (iii) Categorization followed by (iv) an analysis of the ORCHESTRA workflow's robustness.

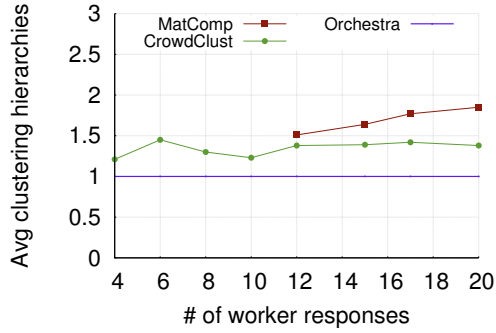


Figure 2.13: Perspective Disambiguation for the shapes dataset

### Perspective Disambiguation

To evaluate ORCHESTRA’s performance in handling multiple worker perspectives, we check the number of organizational perspectives required to explain the clustering provided by ORCHESTRA. We use 20 different worker responses for `shapes`. Recall that `shapes` is small enough to be grouped into one single clustering task. Thus, ORCHESTRA is restricted to the HIERARCHYCONSTRUCTION step, and all algorithms are run on the same data, by repeatedly taking subsets of worker responses.

We compute the number of clustering hierarchies that appear in the consensus clustering. Figure 2.13 shows the average number of organizing hierarchies vs. the number of worker responses ( $r$ ), repeated over 100 random samples of  $r$  worker responses. 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 eigen-components 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 the multiple hierarchies are equally probable *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.

*For all worker responses, ORCHESTRA is able to disambiguate the multiple worker perspectives and treats them accordingly to provide an organization on a single consensus perspective; other algorithms mix organizational perspectives.*

### Sampling

To evaluate the impact of our GENERATESAMPLE procedure, we run our HIERARCHYCONSTRUCTION and MERGINGHIERARCHIES algorithms on randomly sampled

clustering tasks for **scenes**. One such result is shown in Figure 2.14(a). Observe that clusters 1 and 4 have items from the same categories – (*bedroom, living room, kitchen*). This happens because the corresponding samples share have no items from these categories, 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.

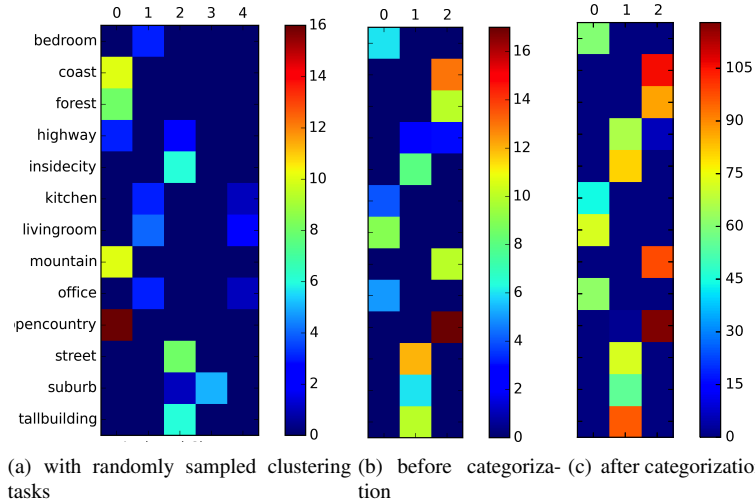
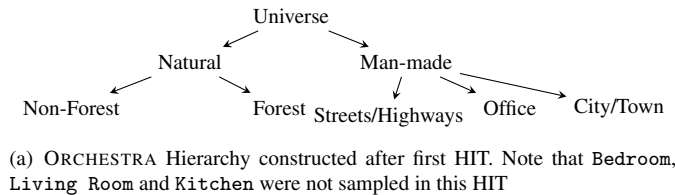
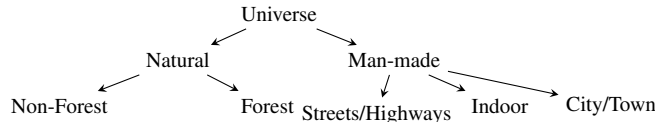


Figure 2.14: Clusters provided by ORCHESTRA on the Scenes dataset

In comparison, ORCHESTRA’s kernel-based sampling is able to handle undiscovered categories. As an example, for the scenes dataset, we take a case where the categories **Bedroom**, **Living Room** and **Kitchen** do not have any samples in the first HIT. Consequently the hierarchy constructed after the first HIT is shown in Figure 2.15(a). Subsequently, these categories are sampled in the second HIT. Because the kernel contains items from all leaf nodes of the current hierarchy, workers are able to group these new categories with **Office**, resulting in a hierarchy shown in Figure 2.15(b).



(a) ORCHESTRA Hierarchy constructed after first HIT. Note that **Bedroom**, **Living Room** and **Kitchen** were not sampled in this HIT



(b) ORCHESTRA Hierarchy constructed after second HIT. After **Bedroom**, **Living Room** and **Kitchen** are sampled, they are clustered together with **Office** to create a new **Indoor** node.

Figure 2.15: Advantage of using kernel-based GENERATESAMPLE

GENERATESAMPLE ensures sufficient overlap between samples to prevent multiple clusters from having the same underlying concept.

### Categorization

In this set of experiments, we examine (i) the effect of categorization phase on the quality of clusterings, (ii) worker agreement on categorization tasks and (iii) comparison with other algorithms as the cost of a categorization task is varied.

**Effect on Quality.** We compare the change in the quality of clustering through the categorization interface on `scenes` - Figures 2.14(b) and 2.14(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 quantitative metrics also reflect this preservation of quality. The change in `Pr` is from 0.224 before categorization to 0.230 after categorization. The corresponding changes in `Re` and `Ac` are  $0.988 \rightarrow 0.985$  and  $0.982 \rightarrow 0.992$  respectively.

*The categorization stage maintains the quality of organization - `Pr`, `Re` or `Ac` remain high even after categorization.*

**Worker agreement.** The categorization phase of ORCHESTRA had a very high agreement between workers. For `imagenet`, on average, 4.11 out of 5 workers agreed on the cluster to be assigned to an item in the categorization task. This average was computed over 910 items assigned to clusters using categorization. The corresponding agreement on `scenes` was 4.69. This high level of agreement in categorization is indicative of the easily discernible concepts underlying each cluster produced by ORCHESTRA and suggests that simple strategies (like majority vote) are sufficient for aggregating worker responses in this phase. Furthermore, exemplar based categorization where workers are shown examples for each category (instead of a description) is effectively able to convey these latent underlying concepts to the workers. Thus, it is even more important to separate worker responses by perspectives and present only one perspective for categorization — as is done by ORCHESTRA.

*The categorization phase has high worker agreement because of easily discernible concepts underlying each cluster. This high agreement ensures that simple strategies like majority vote suffice to aggregate responses from multiple workers.*

**Cost.** In the preceding section, the clustering algorithms were compared on fixed cost with the assumption that the cost of a categorization task is half the cost of a clustering task. However, if the cost of categorization was more, `MatComp` and `CrowdClust` would get more clustering `HITS` to organize the items in the same dollar budget as ORCHESTRA. We therefore examine the effects of varying this cost.

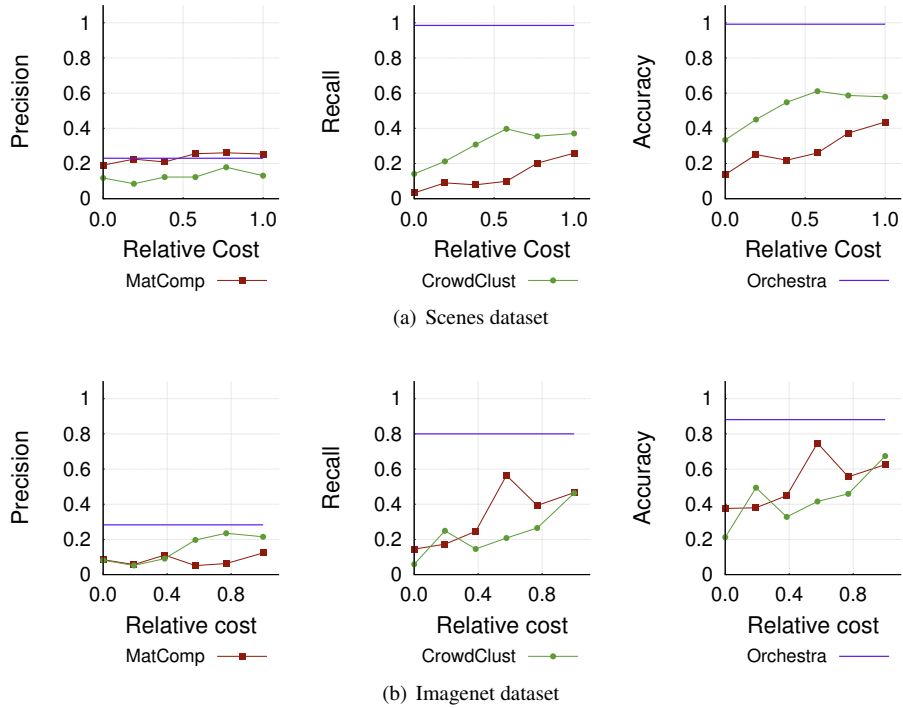


Figure 2.16: Performance of prior work with varying budget. The cost of categorization task (relative to clustering task) is varied. The performance of ORCHESTRA is shown for reference.

In Figure 2.16, we plot the performance of CrowdClust and MatComp while varying the number of clustering HITs for `scenes`. The number of clustering HITs determined by the cost of a categorization task (relative to clustering) is plotted on the x-axis. For a point  $x$  on the x-axis, CrowdClust and MatComp are given a budget equal to ORCHESTRA’s cost, *i.e.* an equivalent of  $15 \times 5 + x \times 26 \times 5$  clustering tasks. From the plots, it is clear that for all relative costs, the accuracy and recall of ORCHESTRA are more than that for CrowdClust and MatComp on both datasets with ORCHESTRA having either the highest or close to the highest precision.

As noted earlier, the cost of a categorization task was half of that for clustering. Combined with the fact that only 5 worker 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. The high level of agreement for categorization further indicates that even fewer worker answers can be sought. 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.

*Even when categorization is as costly as clustering, ORCHESTRA’s organizations are much better on recall and accuracy while being close to the best precision attained by prior work.*

Dataset	Precision		Recall		Accuracy	
	Mean	Variance	Mean	Variance	Mean	Variance
scenes	0.306	0.008	0.927	0.014	0.966	0.003
imagenet	0.319	0.007	0.953	0.004	0.98	0.0007

Table 2.3: The mean and variance of ORCHESTRA’s quality across 50 clustering HITs for each dataset

### Robustness

In general, the workers’ perspective will depend on the set of items to be organized. On seeing only triangles, they may organize them by *color* even though the dominant organization is *shape*. In addition, the final organization outcome of any algorithm may also vary with the number of workers who perform a task. As we showed in Section 2.3, dominant perspectives are more probable when a sufficient number of workers perform every task. Thus, our goal in this set of experiments is to examine the robustness of ORCHESTRA to variations in (i) item sampling and (ii) number of responses.

**Variation in item samples.** To examine the effect of item sampling, we created 50 different random samples of 35 items each from *scenes* and *imagenet*. These samples were then arranged into one clustering HIT. Each such HIT was repeated by 15 workers as earlier. We run ORCHESTRA’s HIERARCHYCONSTRUCTION step on each of these 100 HITs and show the scatter plots of *Pr*, *Re* and *Ac* in Figures 2.17(a), 2.17(b) and 2.17(c) respectively. The x-axis shows the number of the HIT and the y-axis shows the corresponding metric. For both datasets, the metrics are well concentrated, with only 5% of the HITs having  $Pr < 0.2$  or  $Re < 0.8$  or  $Ac < 0.9$ . As we have already seen in Table 2.1 and in Section 2.6.1, clusterings having metrics close to these values are of high quality. This clearly indicates that ORCHESTRA provides high clusterings irrespective of the item set. Additionally, we also show the mean and responses of all metrics for both datasets in Table 2.3. The low values of variance indicate that the mean quality is close to the values reported in Table 2.1 with high confidence.

In addition, in Figure 2.17(d), we also plot a histogram of the size of the largest clique sizes in the clustering graph. The x-axis shows this size whereas the y-axis shows the number of HITs where the largest clique had that size. From this plot, we note that more than 60% of the HITs had a clique size of at least 5. In addition, the average clique size across both datasets was 5.4, *i.e.* the maximum clique was able to *explain* 5 workers responses (out of 15). This suggests that irrespective of item sampling, ORCHESTRA is able to identify the dominant perspectives, providing relevant and accurate clusterings.

ORCHESTRA handles variations in item sampling while maintaining high quality - 95% of the HITs have  $Pr \geq 0.2$  and  $Re \geq 0.8$  and  $Ac \geq 0.9$ .

**Variation in number of responses.** As noted in Section 2.3, the chances of recovering the dominant perspective are higher when a larger number of workers perform each task. For *shapes*, we saw that  $m = 15$  was conservative estimate for the number of workers to guarantee that the probability of getting an organization on SHAPES was 1.

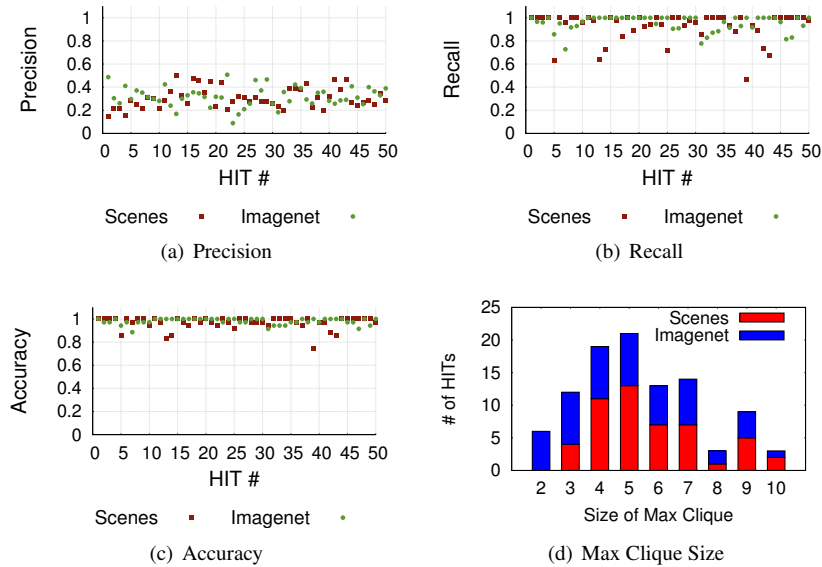


Figure 2.17: ORCHESTRA’s robustness to variations in item sampling. Each point in the scatter plots represents a clustering HIT.

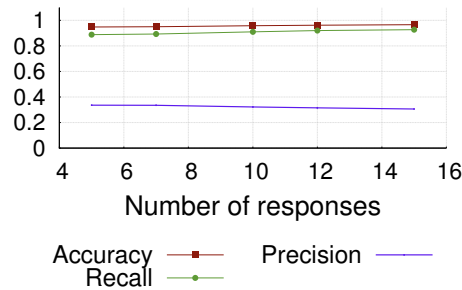


Figure 2.18: ORCHESTRA’s robustness to variations in number of worker responses for the scenes dataset

In this experiment, we see the effect of varying  $m$  for the more challenging scenes and imagenet datasets.

We use the same data as the previous experiment - 50 clustering HITs from scenes. For every HIT, we take a subset of worker responses and run ORCHESTRA on this subset. Figure 2.18 shows the quality of the eventual organizations as the size of this subset is varied. These metrics are averaged over the 50 HITs and 200 random subsets for each subset size. The plots for Pr, Re and Ac show that the quality of the eventual organization is not sensitive to the number of workers providing the clustering. This suggests that our estimate of  $m = 15$  is conservative and lower values can be used without any loss in quality. This robustness to worker responses is built into ORCHESTRA since it identifies a group (clique) of workers who are consistent with each other, thereby providing validation for the group’s responses.

ORCHESTRA’s clustering maintains high Pr, Re and Ac even with fewer worker responses by still identifying a core group of workers who organize items consistently.

The above experiments show that ORCHESTRA produces *high-quality, meaningful or-*



ganizations in a *cost-efficient* and *robust* manner.

### 2.6.3 Discussion

As the above results show, our approach of choosing the maximum likelihood frontier in the maximum likelihood hierarchy ensures that the chosen frontier represents a high-quality clustering. However, it does not provide control over number of clusters – it varies from 3 for `scenes` to 9 for `imagenet`.

To tune the number of clusters ( $z$ ), the strategy of choosing the maximum likelihood frontier can be easily modified to incorporate constraints. One way to satisfy constraints of the form  $z = z_0$  or  $z \geq z_0$  or  $z \leq z_0$  is to look at frontiers that satisfy these constraints and return the frontier with the highest likelihood. While this may not be the maximum-likelihood frontier overall, this is the best *viable* frontier satisfying the constraints. Another strategy could be to *drill-down* on course granularity nodes in the maximum-likelihood frontier. Asking workers to organize only `Quadrilaterals` – a cluster in the maximum-likelihood frontier – can help in discovering finer granularities (`Rectangles`, `Squares`) with high confidence. This drill-down and associated discovery of finer granularity clusters can further improve the precision from the values reported in Table 2.1.

To examine this experimentally, we asked workers to organize some items belonging only to the `indoor` category (cluster # 0) in `scenes`, *i.e.*, the ORCHESTRA workflow was run on a sub-category (`indoor`) of items. The qualitative results of this drill down can be seen in Figure 2.19. The four categories making up the `indoor` cluster are all discovered in the drill-down. These categories are recovered with `Pr`, `Re` and `Ac` of 0.84, 0.94 and 0.94 respectively. Furthermore, the size of the max-clique in the clustering graph was 13 (out of 15). On closer examination, the two images of bedroom category that were placed in cluster #3 are ambiguous and do infact resemble `living rooms`.

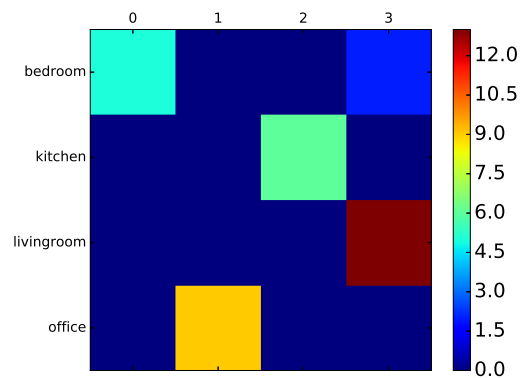


Figure 2.19: Effect of drill-down

Dataset	Sentiment	ORCHESTRA	CrowdClust	MatComp
scenes	+	<p>“... seems to follow a more cohesive pattern.”</p> <p>“The flow of the images are more enjoyable on the left hand side.”</p> <p>“... more coherent as it contained more of the same type of images ...”</p> <p>“It’s a more appropriate, identifiable way to group pictures.”</p> <p>“Each row represented a certain type of image clearly.”</p> <p>“... is predictable.”</p> <p>“In vs. out seems a more contrasting comparison”</p> <p>“... is more coherent and in three distinct groups ...”</p> <p>“It would be the easiest way to sort through and find particular items.”</p> <p>“It would be easy to determine which group a new picture should go in.”</p> <p>“The images seem more ordered and similar ...”</p> <p>“... groups... have a clear distinction. They are each grouped in a clear, distinguishable group with no overlap”</p> <p>“... makes a clear division between the three characteristics (structure internals, structure externals, structureless)”</p> <p>“The picture groupings are more obvious and definite in nature”</p> <p>“It is more understandable and interesting as an organizing idea to me.”</p>	<p>“looks more robust”</p> <p>“more interesting photos”</p> <p>“... looks like a story of building and the beach... seems to tell a story.”</p> <p>“... it had a better mix and was more visually appealing.”</p> <p>“... more coherent ... more content flows from a small room space to larger outdoor spaces.”</p> <p>“It just seemed to flow better.”</p> <p>“makes sense to me as a scroll from left to right (typical reading style)”</p> <p>“... seemed to have different columns of sort of the same pictures”</p>	<p>“they all share the black and white theme”</p> <p>“It has more images that match most of them are of a outdoor scene”</p> <p>“... have more in a common like many are scenic pictures ...”</p> <p>“is a better setup and format for organizing”</p> <p>“It has greater depth and gives a much fuller picture due to there being more images.”</p>
	-		<p>“I cannot tell what each column is trying to group together.”</p> <p>“... shows so many pictures in an almost random order”</p> <p>“... have no clear grouping method”</p> <p>“... images do not always follow a clear pattern”</p> <p>“... too much variation ..., eg pictures of nature jumbled with pictures of buildings”</p> <p>“There are more categories ... than make sense.”</p> <p>“... is confusing and there are too many columns to see it all clearly”</p>	<p>“I dont get the right (one) at all”</p> <p>“has a lot of similar pictures”</p> <p>“some of the pictures didn’t fit their categories”</p>

Table 2.4: Human Evaluation of organizations provided by algorithms for scenes dataset

Dataset	Sentiment	ORCHESTRA	CrowdClust	MatComp
imagenet	+	<p>“The theme makes more sense”</p> <p>“This one was best organized.”</p> <p>“This is the only organizational option that made sense to me.”</p> <p>“Just seems more understandable”</p> <p>“The subjects are more similar to each other with less outliers”</p> <p>“.. is more coherent because the images are similar and you can follow along with the theme”</p> <p>“... flows nicely and makes at least a bit of sense.”</p> <p>“Just seems more efficient”</p> <p>“It makes more sense and everything is in the right place.”</p> <p>“... most of the images are put into a sensible category.”</p> <p>“... more reproducible ... ”</p> <p>“There’s a theme and overall idea to what I’m looking at”</p> <p>“it is more organized. each row consist of mostly the same images”</p> <p>“Things were lumped together in a way I could see the connections.”</p>	<p>“Habitat makes more sense than if the item is edible or not.”</p> <p>“It seems to make more sense to show the animals first in 1 collumn”</p> <p>“... each group that was clear was very thorough and did not have any outliers.”</p>	<p>“This one is better organized.”</p> <p>“These images look like they had slightly more thought put into choosing them.”</p> <p>“A little more eye pleasing in general to me.”</p> <p>“Only 2 columns and easier to view”</p> <p>“Images are lined up better.”</p> <p>“Easier to look at”</p>
	—	<p>“I could find NO single theme.”</p> <p>“... there’s some overlap ...”</p> <p>“Living versus nonliving is a very basic categorization scheme.”</p>	<p>“I didn’t understand why nonliving and living things were grouped together ...”</p> <p>“I could not even tell what the organizing principle in the right hand box was supposed to be!”</p> <p>“... is a little more abstract.”</p> <p>“... had a couple groups that were not coherent at all”</p> <p>“has some pics that are not related”</p>	<p>“Couldn’t find any coherency”</p> <p>“... it’s almost completely random.”</p> <p>“... seems to be have to much variety and I can’t really get a feel for what it is aiming”</p> <p>“ ... just random photos grouped together, nothing coherent. It’s hard to follow any line of thought ...”</p> <p>“I don’t understand the timeline or connections”</p>

Table 2.5: Human Evaluation of organizations provided by algorithms for imagenet dataset

The above discussion shows that different strategies can help in controlling the organization granularity of ORCHESTRA. One such strategy, *drill-down*, was observed to be effective for finding finer granularity clusters. A detailed examination and comparison of such strategies requires further investigation and we defer that to future work.

## 2.7 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 [18, 19, 20]. 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 save cost.

There has been other work on variants of clustering: Heikinheimo and Ukkonen [25] 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 [26] 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. [27] 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. [28, 29] use text labels and filtering on the labels to create a hierarchy while Sun et al. and Karampinas et al. [30, 31] 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. [32] obtain hard pairwise constraints in a crowdsourced setting by asking targeted questions related to an item pair. Lad et al. [33] 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 an 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. [34] create concept embeddings by combining human experts with automation. Tamuz et al. [35] 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 [36, 37] 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. [38, 39, 40, 41, 3] are all examples of work that rely on human judgments to carry out ER, all using pairwise comparisons.

## 2.8 Summary

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  $4\times$  better on accuracy) and are more cost effective (reduction of at least 60%) than other schemes.

We believe our work 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?

## CHAPTER 3

# JELLYBEAN: Crowd-Vision-Hybrid Counting Algorithms

### 3.1 Introduction

The field of computer vision [42, 43] concerns itself with the understanding and interpretation of the contents of images or videos. Many of the fundamental problems in this field are far from solved, with even the state-of-the-art techniques achieving poor results on benchmark datasets. For example, the recent techniques for image categorization achieve average precision ranging from 19.5% (for the `chair` class) to 65% (for the `airplane` class) on a canonical benchmark [44]. Another study [45] has shown that even the state-of-the-art deep-network methods are susceptible to high-confidence erroneous labels, recognizing objects within images containing no objects visible to the human eye.

*Counting* is one such fundamental image understanding problem, and refers to the task of counting the number of items of a particular type within an image or video.

*Counting is important.* Counting objects in images or videos is a ubiquitous problem with many applications. For instance, biologists are often interested in counting the number of cell colonies in periodically captured photographs of petri dishes; counting the number of individuals at concerts or demonstrations is often essential for surveillance and security [9]; counting is often necessary in military applications; counting nerve cells or tumors is standard practice in medical applications [10]; and counting the number of animals in photographs of ponds or wildlife sanctuaries is often essential for animal conservation [11]. In many of these scenarios, making errors in counting can have unfavorable consequences. Furthermore, counting is a prerequisite to other, more complex computer vision problems requiring a deeper, more complete understanding of images.

*Counting is hard for computers.* Unfortunately, current supervised computer vision techniques are typically very poor at counting for all but the most stylized settings, and cannot be relied upon for making strategic decisions. In our conversations with microbiologists, they report that they do not rely on automated counts [46], and will frequently spend hours painstakingly counting and cataloging photographs of cell colonies. The computer vision techniques primarily have problems with *occlusion*, i.e., identifying objects that are partially hidden behind other objects. As an example, consider Figure 3.1, depicting the performance of a recent pre-trained face detection algorithm [47]. The algorithm performs poorly for occluded faces, detecting only 35 out of 59 (59.3%)

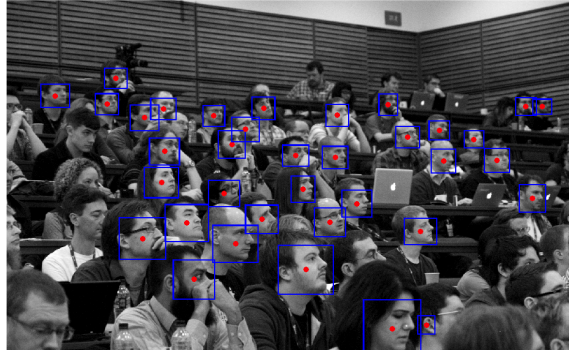


Figure 3.1: Counting: Challenging image for Machine Learning

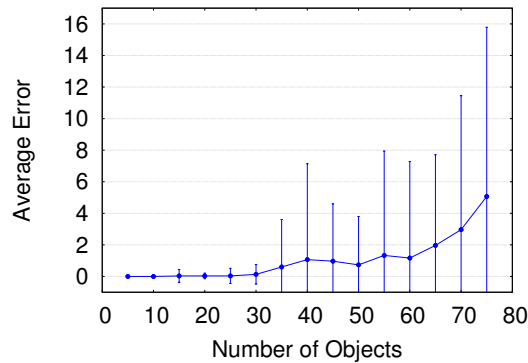


Figure 3.2: Counting: Worker Error

faces. The average precision for the state-of-the-art person detector is only 46% [44]. Furthermore, these techniques are not generalizable; separate models are needed for each new application. For instance, if instead of wanting to count the number of faces in a photograph, we needed to count the number of women, or the number of people wearing hoodies, we would need to start afresh by training an entirely new model.

*Even humans have trouble counting.* While humans are much better at counting than automated techniques, and are good at detecting occluded (hidden) objects, and not missing them while counting, as the number of objects in the image increases, they start making mistakes. Psychology studies have estimated that human beings have a memory span of 7 *chunks* of information at a time [48]. While counting objects is a simpler problem than remembering objects, we expect a similar (but possibly higher) threshold for counting—this is because workers may end up counting the same object multiple times or not count them at all. To observe this behavior experimentally, we had workers count the number of cell colonies in simulated fluorescence microscope images with a wide range of counts. We plot the results in Figure 3.2, displaying the average error in count (on the y-axis) versus the actual count (on the x-axis). As can be seen in the figure, crowd workers make few mistakes until the number of cells hit 20 or 25, after which the average error increases. In fact, when the number of cells reaches 75, the average error in count is as much as 5. (There are in fact many images with even higher errors.) Therefore, simply showing each image to one or more workers and using those counts is not useful if accurate counts are desired (as we will show later).

*The need for a hybrid approach.* Thus, since both humans and computers have trouble



with counting, there is a need for an approach that best *combines human and computer capabilities to count accurately while minimizing cost*. These techniques would certainly help with the counting problem instance at hand—the alternative of having a biology, security, military, medical, or wildlife expert count objects in each image can be error-prone, tedious, and costly. At the same time, these techniques would also *enable the collection of training data at scale*, and thereby spur the generation of even more capable computer vision algorithms. To the best of our knowledge, we are the first to articulate and make concrete steps towards solving this important, fundamental vision problem.

*Key idea: judicious decomposition.* Our approach, inspired by Figure 3.2, is to judiciously decompose an image into smaller ones, focusing worker attention on the areas that require more careful counting. Since workers have been observed to be more accurate on images with fewer objects, the key idea is to obtain reliable counts on smaller, targeted sub-images, and use them to infer counts for the original image. However, it is not clear when or how we should divide an image, or where to focus our attention by assigning more workers. For example, we cannot tell a-priori if all the cell colonies are concentrated in the upper left corner of the image. Figuring out the granularity of this sub-division scheme is another problem. Once it has been decided that a particular image region needs to be divided, how many portions should it be divided into? Another challenge is to divide an image while being cognizant of the fact that you may cut across objects during the division. This could result in double-counting some objects across different sub-images. Lastly, given that there may be computer vision algorithms that can provide us a good starting point for deciding where to focus our attention, yet another challenge is to exploit this information in the best way possible.

*Adaptivity to two modes.* In the spirit of combining the best of human worker and computer expertise, when available, we develop algorithms that are near-optimal for two separate regimes or modes:

- First, assuming we have no computer vision assistance (i.e., no prior computer vision algorithm that could guide us to where the objects are in the image), we design an algorithm that will allow us to *narrow our focus* to the right portions of the image requiring special attention. The algorithm, while intuitively simple to describe, is *theoretically optimal in that it achieves the best possible competitive ratio*, under certain assumptions. At the same time, in practice, on a real crowd-counting dataset, the cost of our algorithm is within  $2.75\times$  of the optimal “oracle” algorithm that has perfect information, while still maintaining very high accuracy.
- Second, if we have primitive or preliminary computer vision algorithms that provide segmentation and prior count information, we design algorithms that can use this knowledge to once again identify the regions of the image to focus our resources on, by “fast-forwarding” to the right areas. We formulate the problem as a *graph binning problem, known to be NP-COMplete and provide an efficient articulation-point based heuristic* for this problem. We show that in practice, on

a real biological cell dataset, our algorithm has a very high accuracy, and only incurs  $1.3\times$  the cost of the optimal, perfect information “oracle” algorithm.

We dub our algorithms for these two regimes as the *JellyBean* algorithm suite, as a homage to one of the early applications of crowd counting<sup>1</sup>.

## 3.2 Preliminaries

In this section, we describe our data model for the input images and our interaction model for worker responses. We provide a list of notations in Table 3.1 for easy reference.

### 3.2.1 Data Model

Given an image with a large number of (possibly heterogenous) objects, our goal is to estimate, with high accuracy, the number of objects present. As noted above in Figure 3.2, humans can accurately count up to a small number of objects, but make significant errors on images with larger numbers of objects. To reduce human error, we split the image into smaller portions, or *segments*, and ask workers to estimate the number of objects in each segment. Naturally, there are many ways we may split an image. We discuss our precise algorithms for splitting an image into segments subsequently. For now, we assume that the segmentation is fixed.

We represent a given image and all its segments in the form of a directed tree  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ , called a *segmentation tree*. The original image is the root node,  $V_0$ , of the tree. Each node  $V_i \in \mathbf{V}, i \in \{0, 1, 2, \dots\}$  corresponds to a sub-image, denoted by  $\text{Image}(V_i)$ . We call a node  $V_j$  a *segment* of  $V_i$  if  $\text{Image}(V_j)$  is contained in  $\text{Image}(V_i)$ . A directed edge exists between nodes  $V_i$  and  $V_j : (V_i, V_j) \in \mathbf{E}$ , if and only if  $V_i$  is the lowest node in the tree, i.e. smallest image, such that  $V_j$  is a segment of  $V_i$ . Note that not all segments need to be materialized to actual images — the tree is conceptual and can be materialized on demand. For brevity, we refer to the set of children of node  $V_i$  (denoted as  $\mathbf{C}_i$ ) as the split of  $V_i$ . If  $\mathbf{C}_i = \{V_1, \dots, V_s\}$ , we have  $\text{Image}(V_i) = \bigcup_{j \in \{1, \dots, s\}} \text{Image}(V_j)$ .

For example, consider the segmentation tree in Figure 3.3. The original image,  $V_0$ , can be split into the two segments  $\{V_1, V_2\}$ .  $V_1$ , in turn, can be split into segments  $\{V_3, V_4\}$ , and  $V_2$  into  $\{V_5, V_6, V_7\}$ . Intuitively, the root image can be thought of as physically segmented into the five leaf nodes  $\{V_3, V_4, V_5, V_6, V_7\}$ .

We assume that all segments of a node are non-overlapping. That is, given any node  $V_i$  and its immediate set of children  $\mathbf{C}_i$ , we have (1)  $\text{Image}(V_i) = \bigcup_{V_j \in \mathbf{C}_i} \text{Image}(V_j)$  and (2)  $\text{Image}(V_j) \cap \text{Image}(V_k) = \phi \forall V_j, V_k \in \mathbf{C}_i$ . We denote the actual number of objects in a segment,  $\text{Image}(V_i)$ , by  $\text{TrueCount}(V_i)$ . A significant challenge is to ensure that each object appears in exactly one segment in  $\mathbf{C}_i$ . We will describe how

<sup>1</sup> Counting or estimating the number of jellybeans in a jar has been a popular activity in fairs since 1900s, while also serving as unfortunate vehicle for disenfranchisement [49].

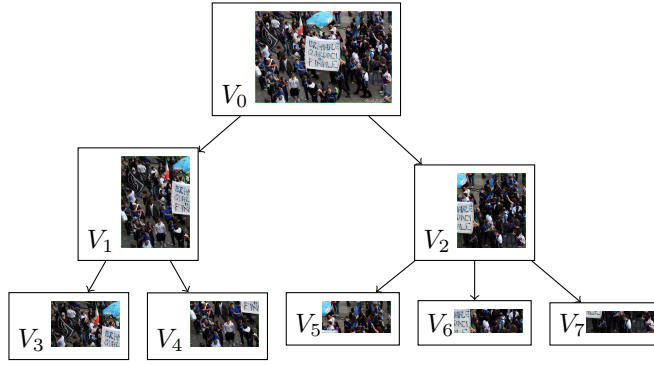


Figure 3.3: Segmentation Tree

we handle this in Section 3.3.4. Our assumption of non-overlapping splits ensures that  $\text{TrueCount}(V_i) = \sum_{V_j \in \mathcal{C}_i} \text{TrueCount}(V_j)$ .

Our first constraint (1) above ensures that each object is counted at least once, while the second constraint (2) ensures that none of the objects are counted more than once.

One of the major challenges of the counting problem is to estimate these TrueCount values with high accuracy, by using elicited worker responses. Given the segmentation tree  $\mathbf{G}$  for image  $V_0$ , we can ask workers to count, possibly multiple times, the number of objects in any of the segments. For example, in Figure 3.3, we can ask workers to count the number of objects in the segments  $(V_3)$ ,  $(V_4)$ ,  $(V_5)$ ,  $(V_6)$ ,  $(V_7)$ ,  $(V_1 = V_3 \cup V_4)$ ,  $(V_2 = V_5 \cup V_6 \cup V_7)$ ,  $(V_0 = V_3 \cup V_4 \cup V_5 \cup V_6 \cup V_7)$ . While we can obtain counts for different nodes in the segmentation tree, we need to consolidate these counts to a final estimate for  $V_0$ . To help with this, we introduce the idea of a *frontier*, which is central to all our algorithms. Intuitively, a frontier  $F$  is a set of nodes whose corresponding segments do not overlap, and cover the entire original image,  $\text{Image}(V_0)$  on merging. We formally define this notion below.

**Definition 12 (Frontier).** Let  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$  be a segmentation tree with root node  $V_0$ . A set of  $k$  nodes given by  $F = \{V_1, V_2, \dots, V_k\}$ , where  $V_i \in \mathbf{V} \forall i \in \{1, \dots, k\}$  is a frontier of size  $k$  if  $\text{Image}(V_0) = \bigcup_{V_i \in F} \text{Image}(V_i)$ , and  $\text{Image}(V_i) \cap \text{Image}(V_j) = \emptyset \forall V_i, V_j \in F$

A frontier  $F$  is now a set of nodes in the segmentation tree such that taking the sum of  $\text{TrueCount}(\cdot)$  over these nodes returns the desired count estimate  $\text{TrueCount}(V_0)$ . Note that the set containing just the root node,  $\{V_0\}$  is a trivial frontier. Continuing with our example in Figure 3.3, we have the following five possible frontiers:  $\{V_0\}$ ,  $\{V_1, V_2\}$ ,  $\{V_1, V_5, V_6, V_7\}$ ,  $\{V_2, V_3, V_4\}$ , and  $\{V_3, V_4, V_5, V_6, V_7\}$ .

### 3.2.2 Worker Behavior Model

Intuitively, workers estimate the number of objects in an image correctly if the image has a small number of objects. As the number of objects increases, it becomes difficult for humans to keep track of which objects have been counted. As a result, some objects may be counted multiple times, whereas others may not be counted at all. Based on the

$\mathbf{G} = (\mathbf{V}, \mathbf{E})$	Segmentation Tree
$V_0$	Root Node
$\text{Image}(V_i)$	Image/Segment corresponding to node $V_i$
$F$	Frontier
$\text{TrueCount}(V_i)$	Actual number of objects in $\text{Image}(V_i)$
$\text{WorkerCount}(V_i)$	Worker estimate of number of objects in $\text{Image}(V_i)$
$\mathbf{C}_i$	Children of node $V_i$
$b$	Fanout of the segmentation tree
$d^*$	Counting threshold for worker errors
$G_P = (V_P, E_P)$	Partition Graph

Table 3.1: Notations

experimental evidence in Figure 3.2, we hypothesize that there is a threshold number of objects, above which workers start to make errors, and below which their count estimates are accurate. Let this threshold be  $d^*$ . So, in our interface, we ask the workers to count the number of objects in the query image. If their estimate,  $d$ , is less than  $d^*$ , they provide that estimate. If not, they simply inform us that the number of objects is greater than  $d^*$ . This allows us to cap the amount of work done by the workers and ensure that we pay them a minimum wage—the workers can count as far as they are willing to correctly, and if the number of objects is, say, in the thousands, they may just inform us that this is greater than  $d^*$  without expending too much effort. We denote the worker’s estimate of  $\text{TrueCount}(V)$  by  $\text{WorkerCount}(V)$ .

$$\text{WorkerCount}(V) = \begin{cases} \text{TrueCount}(V) & : \text{TrueCount}(V) \leq d^* \\ > d^* & : \text{TrueCount}(V) > d^* \end{cases}$$

Based on Figure 3.2, the threshold  $d^*$  is 20. We provide further experimental verification for this error model in Appendix A. While we could choose to use more complex error models, we find that the above model is easy to analyze and experimentally valid, and therefore suffices for our purposes.

### 3.3 Crowdsourcing-Only Solution

In this section, we consider the case when we do not have a computer vision algorithm at our disposal. Thus, we must use only crowdsourcing to estimate image counts. Since it is often hard to train computer vision algorithms for every new type of object, this is a scenario that often occurs in practice.

As hinted at in Section 3.2, the idea behind our algorithms is simple: we ask workers to estimate the count at nodes of the segmentation tree in a top-down expansion, until we reach a frontier such that we have a high confidence in the worker estimates for all nodes in that frontier.

### 3.3.1 Problem Setup

We are given a fixed  $b$ -ary segmentation tree i.e. a tree with each non-leaf node having exactly  $b$  children. We also assume that each object is present in exactly one segment across siblings of a node, and that workers follow the behavior model from Section 3.2.2. Some of these assumptions may not always hold in practice, and we discuss their relaxations later in Section 3.3.4.

For brevity, we will refer to displaying an image segment (node in the segmentation tree) and asking a worker to estimate the number of objects, as *querying the node*. Our problem can be therefore restated as that of *finding the exact number of objects in an image by querying as few nodes of the segmentation tree as possible*. Next, we describe our algorithm on this setting in Section 3.3.2, and give complexity and optimality guarantees in Section 3.3.3.

### 3.3.2 The FrontierSeeking Algorithm

Our algorithm is based on the simple idea that to estimate the number of objects in the root node, we need to find a frontier with all nodes having fewer than  $d^*$  objects. This is because the elicited `WorkerCounts` are trustworthy only at the nodes that meet this criteria. We call such a frontier a *terminating frontier*. If we query all nodes in such a terminating frontier, then the sum of the worker estimates on those nodes is in fact the correct number of objects in the root node, given our model of worker behavior. Note that terminating frontiers are not necessarily unique for any given image and segmentation tree.

**Definition 13.** A frontier  $F$  is said to be terminating if  $\forall V \in F, \text{TrueCount}(V) \leq d^*$

If a node  $V$  has greater than  $d^*$  objects, then we cannot estimate the number of objects in its parent node, and consequently the root node, without querying  $V$ 's children. Our Algorithm 4, `FrontierSeeking(G)`, depends on this observation for finding a terminating frontier efficiently, and correspondingly obtaining a count for the root node,  $V_0$ . The algorithm simply queries nodes in a top-down expansion of the segmentation tree, for example, with a breadth-first or depth-first search. For each node, we query its children if and only if workers report its count as being higher than the threshold  $d^*$ . We continue querying nodes in the tree, only stopping our expansion at nodes whose counts are reported as smaller than  $d^*$ , until we have queried all nodes in a terminating frontier. We return the sum of the reported counts of nodes in this terminating frontier as our final estimate.

### 3.3.3 Guarantees

We now prove the optimality that our Algorithm 4 provides under our proposed model. Given an image and its segmentation tree, let  $F^*$  be a terminating frontier of the smallest possible size, having  $k$  nodes. Although there are many terminating frontiers, there

---

**Algorithm 4** FrontierSeeking( $\mathbf{G}$ )

---

**Require:** Segmentation Tree  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$   
**Ensure:** Terminating Frontier  $F$ , Actual count count

```
 $F \leftarrow \phi$   
count  $\leftarrow 0$   
create a queue  $Q$   
 $Q$ .enqueue( $V_0$ ) {Add root  $V_0$  to  $Q$ }  
while  $Q$  is not empty do  
   $V \leftarrow Q$ .dequeue()  
  Query( $V$ )  
  if WorkerCount( $V$ )  $> d^*$  then  
     $Q$ .enqueue(children( $V$ ))  
  else  
     $F \leftarrow F \cup \{V\}$   
  end if  
end while
```

---

exists a unique minimal terminating frontier with  $k$  nodes, such that all other terminating frontiers,  $F$ , have  $|F| > k$ . Our goal is to find the minimal terminating frontier with as few queried nodes as possible.

First, we describe the complexity of a baseline optimal *oracle* algorithm using the following lemma, whose proof follows trivially from the observation that we need to query at least one complete terminating frontier to obtain a count for the root node of the tree.

**Lemma 7.** *The number of questions required to get a true count of the number of objects in the given image is  $\geq k$ .*

Thus  $k$  is the minimum number of questions that an algorithm with prior knowledge of a minimal terminating frontier requires to determine the true count, if it were only allowed to ask questions at nodes of the tree. We use this as the baseline to compare against online algorithms, that is, algorithms which have no prior knowledge and discover counts of segments only through the process of querying them. To measure the performance of an algorithm against this optimal baseline, we use a *competitive ratio* analysis [50]. Let  $A_{FS}(\mathbf{G})$  denote the sequence of questions asked, or nodes queried, by an online deterministic algorithm  $A$  on segmentation graph  $\mathbf{G}$ . Let  $|A_{FS}(\mathbf{G})|$  be the corresponding number of questions asked. For the optimal oracle algorithm  $OPT$ ,  $|OPT(\mathbf{G})| = k$  where  $k$  is the size of the minimal terminating frontier of  $\mathbf{G}$ . For brevity, we also denote FrontierSeeking algorithm by  $A_{FS}$ .

**Definition 14** (Competitive ratio). *Let  $\mathbb{G}$  be the set of all segmentation trees with fanout  $b$ . The competitive ratio of an algorithm  $A$  is given by  $CR(A) = \max_{\mathbf{G} \in \mathbb{G}} \frac{|A(\mathbf{G})|}{|OPT(\mathbf{G})|}$ .*

Intuitively, the competitive ratio of an algorithm is a measure of its worst case performance against the optimal oracle algorithm. Note that since our segmentation graphs represent real segmentations on images, the true counts on nodes are constrained by  $\text{TrueCount}(\text{parent node}) = \sum \text{TrueCount}(\text{children nodes})$ . This means that when querying nodes in segmentation trees, only answers that are consistent with our worker

behavior model over some consistent assignment of `TrueCounts` will be observed. We now show that our Algorithm 4 has a constant competitive ratio that depends only on the fanout of the given segmentation tree.

**Theorem 7.** *Let  $\mathbb{G}$  be the set of all segmentation trees with fanout  $b$ . We have,  $CR(A_{FS}) = \frac{b}{b-1}$  over  $\mathbb{G}$ .*

*Proof.* Let  $\mathbf{G}$  be any given segmentation tree with fanout  $b$  with minimal terminating frontier  $F^*$ . Let  $\mathbf{G}^*$  be the subtree of  $\mathbf{G}$  above and including  $F^*$ . Now, our algorithm asks nodes in a top-down search expansion, querying a node's children only if it is not a terminating node. Therefore, the set of nodes queried by our algorithm is *upper bounded* by  $\mathbf{G}^*$ . This follows from the fact that to ask any node below  $F^*$ , our algorithm would have to first ask a node in  $F^*$ , at which point, our algorithm will receive a count report of less than  $d^*$ , and choose not to continue its expansion in that subtree. We also observe that our algorithm has to query *at least all* nodes in  $\mathbf{G}^*$ . This follows from the fact that  $F^*$  is a *minimal* terminating frontier which means that every node above  $F^*$  has count greater than  $d^*$ . Combining these two bounds, we have  $|Alg\ 4(\mathbf{G})| = |\mathbf{G}^*|$ . Therefore,  $CR(Alg\ 4) = \max_{\mathbf{G} \in \mathbb{G}} \frac{|\mathbf{G}^*|}{|F^*|}$ . We show below that this value converges to  $\frac{b}{b-1}$ .

**Definition 15** (Compact Frontier). *Frontier  $F_C$  is compact if the difference in depths of any two nodes in  $F_C$  is  $\leq 1$ .*

**Lemma 8.** *Consider a segmentation tree  $\mathbf{G}$  with minimal terminating frontier  $F^*$  of size  $k$ . Let  $\mathbf{G}^*$  be the subtree of  $\mathbf{G}$  above and including  $F^*$ . Then, there exists a compact frontier  $F_C$  of size  $k$  in  $\mathbf{G}$  such that  $|\mathbf{G}^*| = |\mathbf{G}_C|$ , where  $\mathbf{G}_C$  is the subtree of  $\mathbf{G}$  above and including  $F_C$  and  $|\mathbf{G}| =$  number of nodes in  $\mathbf{G}$ .*

*Proof.* If  $F^*$  is compact, then  $F_C = F^*$  and the proof follows trivially. Suppose  $F^*$  is not in the above compact form, that is, it has two nodes whose depths differ by more than 1. Let  $V_i$  and  $V_j$  be two nodes such that  $\text{depth}(V_i) - \text{depth}(V_j)$  is maximum. We transform  $F^*$  as follows: pick  $V_j$  in  $F^*$  (topmost layer) and replace it by its  $b$  children. Simultaneously, pick  $b$  siblings that include  $V_i$  in  $F^*$  ( $F^*$  necessarily has nodes occurring with all siblings), and replace them by their parent. Let the new set of nodes be  $F'$ . It is easy to prove that  $F'$  is a frontier of size  $k$  such that  $|\mathbf{G}^*| = |\mathbf{G}'|$ . Here,  $\mathbf{G}'$  is the subtree of  $\mathbf{G}$  above and including  $F'$ . We observe that  $F'$  is “more compact” than  $F^*$  in that it reduces one node from the top most layer and  $b$  nodes from the bottom most and replaces them with nodes in between. It is easy to see that repeating this transformation iteratively converges to a compact frontier  $F_C$  with all nodes lying in two consecutive depths, say  $d$  and  $d + 1$ .  $\square$

Now, we calculate  $\frac{|\mathbf{G}^*|}{|F^*|} = \frac{|\mathbf{G}_C|}{|F_C|}$ . Let the number of nodes from  $F_C$  at depth  $d$  be  $n_1$ . Let  $n_2 = b^{d-1} - n_1$  be the remaining nodes in depth  $d$ . Then, we know that the nodes from  $F_C$  at depth  $d + 1$  is the set of children corresponding to the  $n_2$  remaining nodes. Therefore,  $|F_C| = n_1 + bn_2$ . We have  $|\mathbf{G}_C| = (1 + b + \dots + b^{d-2}) + n_2 + |F_C|$

and  $|F_C| = n_1 + bn_2$ . Substituting, and simplifying we obtain  $\frac{|\mathbf{G}_C|}{|F_C|} = 1 + \frac{1}{b-1} \left[1 - \frac{1}{b^{d-1} + (b-1)n_2}\right]$ . Note that this is an increasing function of  $n_2$  that converges to  $1 + \frac{1}{b-1}$  as  $n_2 \rightarrow \infty$ . Therefore  $\frac{|\mathbf{G}_C|}{|F_C|} \leq \frac{b}{b-1}$ , and  $\text{CR}(\text{Alg 4}) = \frac{b}{b-1}$ .  $\square$

The following theorem (combined with the previous one) states that our algorithm achieves the best possible competitive ratio across all online deterministic algorithms.

**Theorem 8.** *Let  $A$  be any online deterministic algorithm that computes the correct count for every given input segmentation tree  $G$  with fanout  $b$ . Then,  $\text{CR}(A) \geq (\frac{b}{b-1})$ .*

*Proof.* We divide our proof into two parts. First, we construct a segmentation tree  $\mathbf{G}^*$  for which our algorithm 4 queries fewer nodes than  $A$  to arrive at the correct answer. That is,  $|A(\mathbf{G}^*)| > \text{Alg 4}(\mathbf{G}^*)$ . Next, we show that for  $\mathbf{G}^*$ ,  $|A(\mathbf{G}^*)| \geq (\frac{b}{b-1})|\text{OPT}(\mathbf{G}^*)|$ , thereby proving that  $\text{CR}(A) \geq (\frac{b}{b-1})$ .

Let  $\mathbf{G}$  be any segmentation tree such that  $|A(\mathbf{G})| < \text{Alg 4}(\mathbf{G})$ . Let  $A(\mathbf{G}) = \langle V_1, V_2, \dots, V_n \rangle$  be the sequence of nodes queried by  $A$ . Then, either  $V_1$  is not the root node of  $\mathbf{G}$ , or there exists  $i$  such that  $\text{parent}(V_i) \notin V_1, V_2, \dots, V_{i-1}$ . Intuitively this claim is equivalent to saying that at some point  $A$  queries a node  $V_i$  without having first queried its parent, and can be proven by contradiction. If our claim is not true, then the set of nodes  $A(\mathbf{G})$  is in fact the same set of nodes queried by our algorithm 4 in its top-down traversal. Since we assume  $|A(\mathbf{G})| < \text{Alg 4}(\mathbf{G})$ , we have a contradiction. Now, pick the first  $i$  such that  $\text{parent}(V_i) \notin V_1, V_2, \dots, V_{i-1}$  (similar proof when  $V_1$  is not the root of  $\mathbf{G}$ ). We can also assume that if a node  $V_j$  having count less than  $d^*$  is queried at some point, then algorithm  $A$  does not subsequently query a descendant of  $V_j$  (otherwise we can replace  $A$  by a strictly better deterministic algorithm and repeat our analysis with that).

Let  $F' = \{V_j \in A(\mathbf{G}) \mid j < i \wedge \text{TrueCount}(V_j) \leq d^*\}$  be the set of terminating frontier nodes queried by  $A$  before  $V_i$ . Now, consider the set of nodes  $V \setminus \{V_1, V_2, \dots, V_{i-1}\}$ . For any other image segmentation tree having identical `TrueCounts` on  $\{V_1, V_2, \dots, V_{i-1}\}$ , and arbitrary `TrueCounts` on the nodes  $V \setminus \{V_1, V_2, \dots, V_{i-1}\}$ , deterministic algorithm  $A$  queries the same sequence of nodes,  $\{V_1, V_2, \dots, V_{i-1}\}$ . It is easy to see that all possible segmentation trees (with `TrueCounts` on  $\{V_1, V_2, \dots, V_{i-1}\}$  fixed) contain  $F'$  in their minimal terminating frontiers. Consider the new segmentation tree  $\mathbf{G}^*$  which has identical `TrueCounts` on  $\{V_1, V_2, \dots, V_{i-1}\}$  and the smallest set of nodes  $F''$  (across all the possible segmentation trees described), such that  $F = F' \cup F''$  is the minimal terminating frontier. Intuitively, this new segmentation tree will have a `TrueCount` of less than  $d^*$  on every node just after  $\{V_1, V_2, \dots, V_{i-1}\}$  in the top-down expansion. We observe that our algorithm 4 matches  $A$  up to  $\{V_1, V_2, \dots, V_{i-1}\}$ , and then asks the minimal possible number of questions to discover  $F''$ . It follows that  $|A(\mathbf{G}^*)| > \text{Alg 4}(\mathbf{G}^*)$ .

We have shown above that there exists at least one segmentation tree  $\mathbf{G}^*$  such that  $|A(\mathbf{G}^*)| > \text{Alg 4}(\mathbf{G}^*)$ . Recall from Theorem 3.3.3 that  $\frac{|\text{Alg 4}(\mathbf{G}^*)|}{|\text{OPT}(\mathbf{G}^*)|} = 1 + \frac{1}{b-1} \left[1 - \frac{1}{b^{d-1} + (b-1)n_2}\right]$ . Using  $|A(\mathbf{G}^*)| \geq \text{Alg 4}(\mathbf{G}^*) + 1$ , we can show that  $|A(\mathbf{G}^*)| \geq (\frac{b}{b-1})|\text{OPT}(\mathbf{G}^*)|$ .



This completes our proof.  $\square$

### 3.3.4 Practical Setup

In this section we discuss some of the practical design challenges faced by our algorithm and give a brief overview of our current mechanisms for addressing these challenges. While our current mechanisms suffice for deriving accurate results, it remains to be seen if further exploration into these choices would give greater benefits—this forms an important direction for our future work in this space. Further details can be found in Section 3.5.

**Worker error.** So far, we have assumed that human worker counts are accurate for nodes with fewer than  $d^*$  objects permitting us to query each node just a single time. However, this is not always the case in practice. In reality, workers may make mistakes while counting images with any number of objects (and we see this manifest in our experiments as well). So, in our algorithms, we show each image or node to multiple (five in our experiments) workers and aggregate their answers via `median` to obtain a count estimate for that node. We observe that although individual workers can make mistakes, our aggregated answers satisfy our assumptions in general (e.g., that the aggregate is always accurate when the count is less than  $d^*$ ). While we use a primitive aggregation scheme in this work, it remains to be seen if more advanced aggregation schemes, such as those in [51, 4, 52] would lead to better results; we plan to explore these schemes in future work.

**Segmentation tree.** So far, we have also assumed that a segmentation tree with fanout  $b$  is already given to us. In practice, we are often only given the whole image, and have to create the segmentation tree ourselves. In our setup, we create a binary segmentation tree ( $b = 2$ ) where the children of any node are created by splitting the parent into two halves along its longer dimension. As we will see later on, this choice leads to accurate results. While our algorithms also apply to segmentation trees of any fanout; further investigation is needed to study the effect of  $b$  on the cost and accuracy of the results.

**Segment boundaries.** We have assumed that objects do not cross segmentation boundaries, i.e., each object is present in exactly one leaf node, and cannot be partially present in multiple siblings. Our segmentation does not always guarantee this. To handle this corner case, in our experiments we specify the notion of a “majority” object to workers with the help of an example image, and ask them to only count an object for an image segment *if the majority of it is present in that segment*. Once again, we find that this leads to accurate results in our present experiments.

Note that this notion of “majority object” is harder to define for segmentation trees with fanout higher than 2, as the same object could be present in 1, 2, 3 or 4 image segments. To check for “majority”, the worker would have to look at all segments containing a portion of the object simultaneously and gauge which contains the largest chunk. An alternate way to handle this could be to instead just ask the worker to populate a table for each displayed image containing two columns: (a) Number of

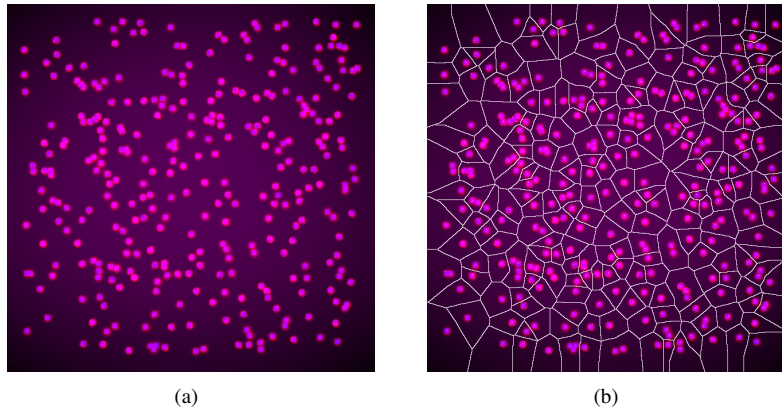


Figure 3.4: Biological image (a) before and (b) after partitioning

boundaries crossed, and (b) Number of objects crossing these many boundaries. This table could then be used to avoid the double-counting of objects when aggregating counts across different image segments.

As mentioned earlier, while our present design decisions are adequate for high quality results, a complete exploration of the effect of aggregations schemes, fanout, and objects crossing segment boundaries is left for future work. We revisit these design decisions in Section 3.5.

## 3.4 Incorporating Computer Vision

Unlike the previous section, where we assumed a fixed segmentation tree, here, we use computer vision techniques (when easily available) to help build the segmentation tree, and use crowds to subsequently count segments in this tree. For certain types of images, existing machine learning techniques give two things: (1) a partitioning of the given image such that no object is present in multiple partitions, and (2) a prior count estimate of the number of objects in each partition. While these prior counts are not always accurate and still need to be verified by human workers, they allow us to skip some nodes in the implicit segmentation tree and “fast-forward” to querying lower nodes, thereby requiring fewer human tasks. Note that this is a departure from our top-down search based approach discussed in Section 3.3. We formulate our idea of fast-forwarding through the segmentation tree as a merging problem and discuss it in greater detail in Section 3.4.2.

### 3.4.1 Partitioning

As a running example, we consider the application of counting cells in biological images. Figure 3.4(a) shows one such image, generated using SIMCEP, a tool for simulating fluorescence microscopy images of cell populations [53]. SIMCEP is the gold standard for testing algorithms in medical imaging, providing many tunable parameters that can simulate realworld conditions. We implement one simple partitioning scheme

that splits any given such cell population image into many small, disjoint partitions. Applying this partitioning scheme to the image in Figure 3.4(a) yields Figure 3.4(b). Combined with the partitioning scheme above, we can leverage existing machine learning (ML) techniques to estimate the number of objects in each of the partitions. We denote these ML-estimated counts on each partition,  $u$ , as *prior counts* or simply priors,  $d^u$ . Note that these priors are only approximate estimates, and still need to be verified by workers. We discuss details of our partitioning algorithm, prior count estimation, and other implementation details later in Section 3.4.3.

We use these generated partitions and prior counts to define a *partition graph* as follows:

**Definition 16** (Partition Graph). *Given an image split into the set of partitions,  $V_P$ , we define its partition graph,  $G_P = (V_P, E_P)$ , as follows. Each partition,  $u \in V_P$ , is a node in the graph and has a weight associated with it equal to the prior,  $w(u) = d^u$ . Furthermore, an undirected edge exists between two nodes,  $(u, v) \in E_P$ , in the graph if and only if the corresponding partitions,  $u, v$ , are adjacent in the original image.*

Notice that while we have used one partitioning scheme and one prior count estimation technique for our example here, other machine learning or vision algorithms for this, as well as other settings provide similar information that will allow us to generate similar partition graphs. Thus, the setting where we begin with a partition graph is *general*, and applies to other scenarios.

Now, given a partition graph, one approach to counting the number of objects in the corresponding image could be to have workers count each partition individually. The number of partitions in a partition graph is, however, typically very large, making this approach impractical. For instance, most of the 5–6 partitions close to the lower right hand corner of the image above have precisely one cell, and it would be wasteful and expensive to ask a human to count each one individually. Next, we discuss an algorithm to merge these partitions into a smaller number, to minimize the number of human tasks.

### 3.4.2 Merging Partitions

Given a partition graph corresponding to an image, we leverage the prior counts on partitions to avoid the top-down expansion of segmentation trees described in Section 3.3. Instead, we infer the count of the image by merging its partitions together into a small number of *bins*, each of which can be reasonably counted by workers, and aggregating the counts across bins.

**Merging problem.** Intuitively, the problem of merging partitions is equivalent to identifying connected components (or bins) of the partition graph, with total weight (or count) at most  $d^*$ . Since workers are accurate on images with size up to  $d^*$ , we can then elicit worker counts for our merged components and aggregate them to find the count of the whole image. Overall, we have the following problem:

**Problem 4 (Merging Partitions).** *Given a partition graph  $G_P = (V_P, E_P)$  of an image, partition the graph into  $k$  disjoint connected components in  $G_P$ , such that the sum of node weights in each component is less than or equal to  $d^*$ , and  $k$  is as small as possible.*

In other words, given a partition graph  $G_P = (V_P, E_P)$  of an image, we wish to find  $m$  disjoint connected components in  $G_P$ , such that the sum of node weights in each component is close to  $d^*$ . Enforcing disjoint components ensures that no components overlap over a common object, thereby avoiding double-counting. Furthermore, restricting our search to connected components ensures that our displayed images are contiguous — this is a desirable property for images displayed to workers over most applications, because it provides useful, necessary context to understand the image.

**Hardness and reformulation.** The solution to the above problem would give us the required merging. However, the problem described above can be shown to be NP-Complete, using a reduction from the NP-Complete problem of partitioning planar bipartite graphs [54]; our setting uses arbitrary planar graphs, and so our problem is more general. Thus, we have:

**Theorem 9 (Hardness).** *Problem 4 is NP-COMPLETE.*

We consider an alternative formulation for the above balanced partitioning problem based on the idea that given an upper bound on the size of each component, balancing component sizes is intuitively similar to finding the minimum number of covering components satisfying the size constraint. That is, given total weight on the graph of say  $D$ , and upper bound on component weight  $d^*$ , the problem of partitioning the graph into roughly even sized components with weights as close to (but not exceeding)  $d^*$  will yield roughly  $D/d^*$  components. Conversely, the problem of partitioning the segmentation graph into the smallest set of components, each upper bounded by size  $d^*$ , will yield roughly  $D/d^*$  even sized components. We formalize this problem of partitioning the segmentation graph into the number of disjoint connected components bounded by size  $d^*$  below. Note that while this problem is still NP-COMPLETE, as we see below, it is more convenient to design heuristic algorithms for it.

**Problem 5 (Modified Merging Problem).** *Let  $d_{max} = \max_u d^u$ ,  $u \in V_P$  be the maximum partition weight in the partition graph  $G_P = (V_P, E_P)$ . Split  $G_P$  into the smallest number of disjoint, connected components such that for each component, the sum of its partition weights is at most  $k \times d_{max}$ .*

By setting  $k \leq d^*/d_{max}$  in the above problem, we can find connected components whose prior counts are estimated to be at most  $d^*$ . Observe that here, although we do not start out with a segmentation tree, the partitions provided by the partitioning algorithm can be thought of as leaf nodes of a segmentation tree and our merged components form parents, or ancestors of the leaf nodes. We comment on the relationship between the partition graphs (discussed in this section) and the segmentation graphs (discussed in Section 3.3) in greater detail below.

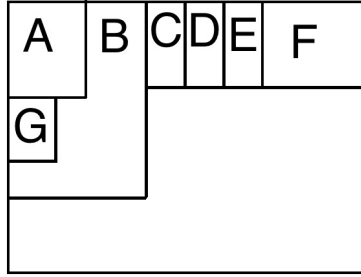


Figure 3.5: Articulation Point

Each component produced via a solution of Problem 5 also corresponds to an actual image segment formed by merging its member partitions: if the prior counts are accurate, these image segments together comprise a minimal terminating frontier for some segmentation tree. While in practice, they need not necessarily form a minimal terminating frontier, or indeed even a terminating frontier, we observe that they provide very good approximations for one.

**Relationship between Partition graphs and Segmentation trees.** So far we have looked at the problem of counting given a partition graph as an independent merging problem. In this section, we discuss how this problem is related to that of counting on a segmentation tree described in Section 3.3. First, we note that since partitions are atomic segments that we do not split further, they can be thought of as the leaf nodes of a segmentation tree for a given image. Observe that fixing the partition graph does not necessarily fix a segmentation tree – there can be many segmentation trees with the same set of leaf nodes (partitions). Intuitively, each intermediate (non-leaf) node in any such segmentation tree is the union of a group of *adjacent* partitions (or leaf nodes), and its prior count estimate is the sum of the priors of its constituent partitions. Note that by only taking the union of adjacent partitions to form intermediate nodes of the segmentation tree, we restrict ourselves to trees where each node, or image, is *contiguous*.

If the prior count on an intermediate node is greater than  $d^*$ , we can now directly query a lower node in the segmentation tree, without verifying this image against the crowd. Intuitively, this is exactly what we do when we display our merged components to workers — we are skipping the nodes for which querying them would anyway yield incorrect answers resulting in image splitting. This is equivalent to jumping ahead to a lower layer of the segmentation tree, one that is potentially very close to a terminating frontier. Also note that jumping too far ahead (segments having counts  $\ll d^*$ ), while yielding accurate counts on each node, will result in the counting of a large number of nodes, which will correspondingly increase the cost of counting. As a compromise, we wish to jump ahead to nodes that ideally each have prior counts just smaller than or equal to  $d^*$ . This motivates our merging problem 5.

Given the hardness of this modified merging problem, we now discuss good heuristics for it, and provide theoretical and experimental evidence in support of our algorithms. Note that while we discuss the complexity of our merging algorithms, in

practice their running time is small compared to the latency of human workers. It is still impractical to run a brute force algorithm to solve the merging problem optimally – fortunately, our second, relatively sophisticated heuristic, *ArticulationAvoidance*, described below achieves the theoretical optimum for most problem instances and is very close to it for the rest.

**FirstCut** Algorithm.

One simple, natural approach to Problem 5, motivated by the first fit approximation to the Bin-Packing problem [55], is to start a component with one partition, and incrementally add neighboring partitions one-by-one until no more partitions can be added without violating the upper bound,  $k \times d_{max}$  on the sum of vertex weights. We refer to this as the **FirstCut** algorithm, and detail it in Algorithm 5.

---

**Algorithm 5** **FirstCut**( $\mathbf{G}_P$ )

---

**Require:** Partition Graph  $\mathbf{G}_P = (\mathbf{V}_P, \mathbf{E}_P)$

**Ensure:** Set  $\mathbf{S}_P$  of components

$\mathbf{V}_{copy} \leftarrow \mathbf{V}_P$

$\mathbf{S}_P \leftarrow \phi$

**while**  $\mathbf{V}_{copy}$  is not empty **do**

$newComponent \leftarrow \phi$

    Choose  $V \in \mathbf{V}_{copy}$

$newComponent \leftarrow newComponent \cup V$

$\mathbf{N} \leftarrow \mathbf{V}_{copy}.neighbors(V)$

$\mathbf{V}_{copy}.removeNode(V)$

**while** Sum of vertex weights in  $newComponent < k \times d_{max}$  **do**

        Choose  $V \in \mathbf{N}$

$newComponent \leftarrow newComponent \cup V$

$\mathbf{N} \leftarrow \mathbf{N} \cup \mathbf{V}_{copy}.neighbors(V)$

$\mathbf{N}.removeNode(V)$

$\mathbf{V}_{copy}.removeNode(V)$

**end while**

$\mathbf{S}_P \leftarrow \mathbf{S}_P \cup \{newComponent\}$

**end while**

---

Consider running this algorithm on a complete partition graph (every partition touches every other partition). By repeating this procedure to termination, (1) each of the merged components should have between  $(k - 1) \times d_{max}$  and  $k \times d_{max}$  objects, and (2) the number of components obtained will be at most  $\frac{k}{k-1} \times OPT$ , where  $OPT$  is a theoretical lower bound to the minimum number of components possible in Problem 5. In practice, however, we find that for several graphs, certain partitions and components get disconnected by the formation of other components during this process, resulting in a sub-optimal merging. For example, consider the partitioning shown in Figure 3.5.

Suppose partitions  $A, \dots, G$  contain 100 objects each and parameter  $k = 6$ . The maximum allowed size for a merged component is  $6 \times d_{max} \geq 6 \times 100$ . Supposing we start a component with  $A$ , and incrementally merge in partitions  $B, \dots, F$ , we end up isolating  $G$  as an independent merged component. This causes a significant

departure from our first bound on component size described above ( $500 \leq (k - 1) \times d_{max} \leq$  component containing  $G$ ), which in turn will result in a higher final number of merged components than optimal. Note that if we relaxed our requirement of only merging adjacent partitions and allow arbitrary non-connected components, the above guarantees would hold for our `FirstCut` algorithm over arbitrary partition graphs.

**Theorem 10** (Complexity). *The worst case complexity of the `FirstCut` algorithm is  $\mathcal{O}(n^2)$  where  $n$  is the number of nodes in the partition graph.*

*Proof.* First, we observe that for each growing component, each partition is examined at most once - if it is not added to the component at that time (because adding it will cause the component to exceed the threshold size), then it need never be looked at for that component again. So each component takes at most  $\mathcal{O}(n)$  time to form, where  $n$  is the total number of partitions. In the worst case, when every component is an individual node (for example if the partition graph was a star with every node connected to just one common large central node), the number of components formed could at most be  $\mathcal{O}(n)$ , resulting in an overall complexity of  $\mathcal{O}(n^2)$  for the `FirstCut` algorithm.  $\square$

While the number of components formed could at most be  $\mathcal{O}(n)$ , in practice, given an upper bound on component size of  $kd_{max}$ , we observe that this number is close to  $n/k$  resulting in an effective run time of  $\mathcal{O}(n^2/k)$ . Also, if we relax the requirement of only merging adjacent partitions, then the worst case running time of our algorithm is  $\mathcal{O}(n^2/k)$  since (as discussed above) every component will have size lying between  $(k - 1)d_{max}$  and  $kd_{max}$  resulting in  $\mathcal{O}(n/k)$  components.

Next, we describe an algorithm that builds on the binning ideas used in Algorithm 5, while also efficiently handling cases like the one shown in Figure 3.5.

#### ArticulationAvoidance Algorithm.

Recall that applying our first cut procedure to Figure 3.5 results in poor quality components if we merge partitions  $B \dots F$  to  $A$  before  $G$ . Intuitively, when adding  $B$  to the component containing  $A$ , the partition graph is split into two disconnected components: one containing  $G$ , and another containing  $C \dots F$ . Given our constraint requiring connected components (contiguous images), this means that partition  $G$  can never be part of a reasonably sized component. This indicates that merging *articulation partitions* like  $B$ , i.e., nodes or partitions whose removal from the partition graph splits the graph into disconnected components, potentially results in imbalanced final merged components:

**Definition 17** (Articulation Partition). *A node  $v$  in graph  $G_P$  is defined to be an articulation partition iff removing it (along with the edges through it) disconnects the graph  $G_P$ .*

Since adding articulation partitions early results in the formation of disconnected components or imbalanced islands, we implement our `ArticulationAvoidance` algorithm detailed in Algorithm 6 that tries to merge them to growing components as

late as possible. We merge partitions as before, growing one component at a time up to an upper bound size of  $k \times d_{max}$ , but we prioritize the adding of non-articulation partitions first. With each new partition,  $u$ , added to a growing component, we update the graph of the remaining unmerged partitions by removing the node corresponding to  $u$  from the remaining partition graph—we also update our list of articulation partitions for the new graph and repeat this process until all partitions have been merged into existing components.

---

**Algorithm 6** ArticulationAvoidance( $\mathbf{G}_P$ )

---

**Require:** Segmentation Graph  $\mathbf{G}_P = (\mathbf{V}_P, \mathbf{E}_P)$

**Ensure:** Set  $\mathbf{S}_P$  of components

```

 $\mathbf{V}_{copy} \leftarrow \mathbf{V}_P$ 
 $\mathbf{S}_C \leftarrow \phi$ 
while  $\mathbf{V}_{copy}$  is not empty do
    newComponent  $\leftarrow \phi$ 
    Choose  $V \in \mathbf{V}_{copy}$  with priority to non-articulation points
    newComponent  $\leftarrow$  newComponent  $\cup V$ 
     $\mathbf{N} \leftarrow \mathbf{V}_{copy}.neighbors(V)$ 
     $\mathbf{V}_{copy}.removeNode(V)$ 
    Update articulation points of  $\mathbf{V}_{copy}$ 
    while Sum of vertex weights in newComponent  $< k \times d_{max}$  do
        Choose  $V \in \mathbf{N}$  with priority to non-articulation points
        newComponent  $\leftarrow$  newComponent  $\cup V$ 
         $\mathbf{N} \leftarrow \mathbf{N} \cup \mathbf{V}_{copy}.neighbors(V)$ 
         $\mathbf{N}.removeNode(V)$ 
         $\mathbf{V}_{copy}.removeNode(V)$ 
        Update articulation points of  $\mathbf{V}_{copy}$ 
    end while
     $\mathbf{S}_P \leftarrow \mathbf{S}_P \cup \{\text{newComponent}\}$ 
end while

```

---

Again, we state and prove the worst-case complexity of our algorithm below.

**Theorem 11.** *The worst case complexity of the ArticulationAvoidance algorithm is  $\mathcal{O}(n^2(n+m))$  where  $n$  is the number of nodes and  $m$  is the number of edges in the partition graph.*

*Proof.* Each time a partition is added to an existing component, we recompute the set of articulation points. Since the maximum size of a component is  $kd_{max}$ , we update the set of articulation points at most  $\min(n, kd_{max})$  times for each component (if each new partition added is of size 1). We use a standard library to compute the set of articulation points whose running time is  $\mathcal{O}(n+m)$  where  $n$  is the number of partitions, and  $m$  is the number of edges between partitions in the partition graph. Here, we use the `articulation_points` function of Python’s NetworkX library [56] which is implemented using a non-recursive depth-first-search (DFS) that keeps track of the highest level that back edges reach in the DFS tree.

Similar to the FirstCut algorithm, for each growing component, each partition is examined at most once. So each component takes at most  $\mathcal{O}(n+(n+m) \min(n, kd_{max}))$ , or  $\mathcal{O}(n(n+m))$  time to form. Again, in the worst case, the number of components



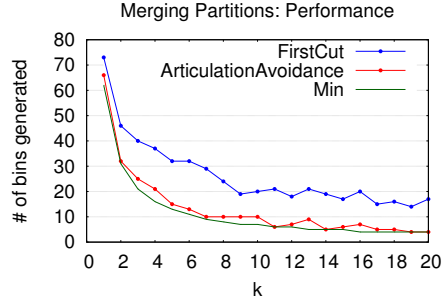


Figure 3.6: Performance of algorithms for merging partitions. `Min` corresponds to the absolute minimum number of segments required such that each segment has less than  $k \times d_{max}$  objects

formed could at most be  $\mathcal{O}(n)$ , resulting in an overall complexity of  $\mathcal{O}(n^2(n+m))$  for the `ArticulationAvoidance` algorithm.  $\square$

While the worst case complexity is  $\mathcal{O}(n^2(n+m))$ , in practice, we observe the number of components formed using this algorithm is closer to  $n/k$  resulting in an effective run time of  $\mathcal{O}(n^2(n+m)/k)$ .

**Evaluation.** We perform extensive evaluation of our algorithms on synthetic and real partition graphs on the metric of number of components generated. Since the optimal, minimum number of bins is not known to us, we compare our algorithms against the theoretical minimum. That is, given a partition graph with sum of partition weights  $N$ , highest partition weight  $d_{max}$ , and upper bound on component size  $kd_{max}$ , the theoretically minimum number of components possible is  $\frac{N}{kd_{max}}$ . Note that this is a lower bound to the true optimal number of components. We show one representative plot for the variation in number of components created (y-axis) on the partitioned image shown in Figure 3.4(b), against the maximum component size determined by  $k$  (x-axis) in Figure 3.6. The image has 315 objects with  $d_{max} = 5$  and the plot is representative of the following general trend—we observe that `ArticulationAvoidance` performs close to the theoretical optimum; `FirstCut`, on the other hand, often gets stuck at articulation partitions, unable to meet the theoretical optimum.

### 3.4.3 Practical Setup

In this section we discuss some of the implementation details of and challenges faced by our algorithms in practice. Many of the challenges faced in Section 3.3.4 apply here as well.

**Partitioning.** The first step of our algorithm is to partition the image into small, non-overlapping partitions. To do this, we use the marker-controlled watershed algorithm [57]. The foreground markers are obtained by background subtraction using morphological opening [57] with a circular disk.

**Prior counts.** In the example of Figure 3.4, we learn a model for the biological cells using a simple Support Vector Machine classifier, trained on positive and negative ex-

amples extracted from 25 such images. For a test image, every  $15 \times 15$  pixel window in the image is classified as ‘cell’ or ‘not cell’ using the learned model to obtain a confidence map of the original image. The confidence value associated with each window is a measure of probability that the window contains a cell. Simple local maxima search and thresholding of this confidence map provides the locations of the cells. For more details of this approach, we refer the reader to [58]. Empirically we observed that our choice of threshold = 0.5 gave closest estimates to the true counts. Note that this procedure *always undercounted*, that is, the prior count estimate obtained for any partition was always smaller than the true number of objects in that partition. Setting a lower threshold value can cause overcounting, and we found these estimates to be more erroneous than the ones obtained by higher thresholds. Additionally, if the machine learning algorithm overcounts, then each component will have  $\text{TrueCount} < d^*$ . This implies that the corresponding nodes will lie below the actual minimal terminating frontier in the segmentation tree, and the size of our guess of the frontier will be higher than the minimal frontier. It is thus more costly (in terms of number of nodes queried) to overcount.

**Traversing the Segmentation Tree.** While Section 3.4.2 gives us a set of merged components, or unified image segments with prior count estimates, we still need to show these images to human workers to verify the counts. As mentioned earlier in the same section, by setting  $k = \lfloor \frac{d^*}{d_{max}} \rfloor$ , we find connected components whose prior count is estimated to be at most  $d^*$ . Since the prior count estimates are approximate and lower than the true counts, we expect these merged image segments constructed to have true counts close to, and potentially higher  $d^*$ . One option is to have (multiple) workers simply count each of these image components and aggregate the counts to get an estimate for the whole image. Since some of these image components may have counts higher than our set worker threshold of  $d^*$ , our model tells us that worker answers on the larger components could be inaccurate. So, another option is to use these images as a starting point for an expansion down the segmentation tree, and perform a `FrontierSeeking` search similar to that in Section 3.3 by splitting these segments until we reach segments whose counts are all under  $d^*$ . We compare these two counting algorithms (termed `AA` and `AA-Split`) further in Section 3.5.

## 3.5 Experimental Study

We deployed our crowdsourcing solution for counting on two image datasets that are representative of the many applications of our work. We examine the following questions:

- How do the JellyBean algorithms compare with the theoretically best possible “oracle” algorithms on cost?
- How accurate are the JellyBean algorithms relative to machine learning baselines?



Figure 3.7: Images in the crowd dataset

- What are the monetary costs of our algorithms, and how do they scale with the number of objects?
- How accurate are the JellyBean algorithms relative to directly asking workers to count on the entire image?
- Do we get any additional benefit from counting nodes below the initially determined terminating frontier (when we work in concert with an ML algorithm)?

Later, in Section 3.5.4, we provide details of experiments evaluating various answer aggregation schemes beyond median. In our appendix, we experimentally study worker behavior, including the frequency of errors and level of confidence in answers.

### 3.5.1 Datasets

**Dataset Description.** Our first dataset is a collection of 12 images from Flickr. These images, shown in Figure 3.7, depict people in various settings, with the number of people (counts) ranging from 41 to 209. This dataset is representative of the applications in surveillance. This is a challenging dataset, with people looking very different across images—ranging from partially to completely visible, aligned at different angles and sizes, and varying backgrounds. Furthermore, no priors or partitions are available for these images—so we evaluate our solutions from Section 3.3 on this dataset. For the rest of this section, we refer to this as the *crowd dataset*.

The second dataset consists of 20 simulated images showing biological cells, generated using SIMCEP [53]. The number of objects in each image was sampled uniformly from the range 150 to 350. The minimum number of objects in an image in our (randomly) generated dataset was 151, and the maximum was 328. The computer vision techniques detailed in Section 3.4 are applied on each of these images to get prior counts and partitions. For the remainder of this section, we refer to this as the *biological dataset*.

**Segmentation Tree Construction.** For the crowd dataset, the segmentation tree was

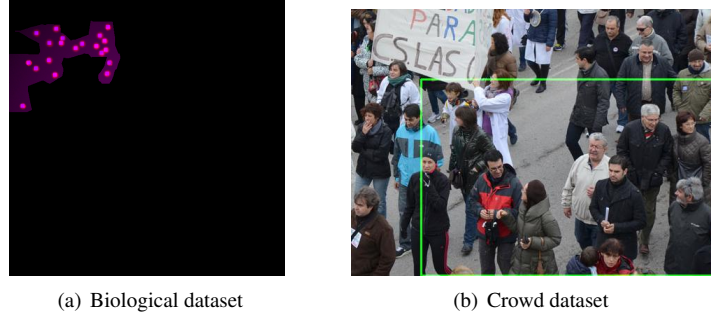


Figure 3.8: Counting Interface: Sample Image shown to workers

constructed with fanout  $b = 2$  until a depth of 5, for a total of 31 nodes per image. At each stage, the image was split into two equal halves along the longer dimension. This ensures that the aspect ratios of all segments are close to the aspect ratio of the original image. Given a segment, workers were asked to count the number of ‘majority’ heads (as described in Section 3.3.4)—if a head crossed the image boundary, it was to be counted only if the worker felt that majority of it was visible. To aid the worker in judging whether a majority of an object lies within the image, the surrounding region was also shown demarcated by clear lines. One such image is shown in Figure 3.8(b).


For the biological dataset, bins were generated by our `ArticulationAvoidance` algorithm. As detailed in Section 3.4, we start our search for the terminating frontier from these nodes. Starting from these bins, subsequent layers of the segmentation tree are constructed by applying `ArticulationAvoidance` recursively, but with a reduced bin size threshold. The threshold  $d^*$  in the algorithm is reduced by a factor of `split_ratio` for subsequent layers. Although any `split_ratio`  $> 1$  suffices, choosing higher values generates more segments (children), leading to more nodes being queried. Smaller values generate less segments (children), but the number of objects in each segment is possibly higher. If the `WorkerCount` on these nodes is still more than  $d^*$ , going deeper into the segmentation tree becomes necessary, thereby increasing the number of round-trips to crowd market. Additionally, the number of segments generated by `ArticulationAvoidance` is more than  $\frac{\text{TrueCount}}{\left(\frac{d^*}{\text{split\_ratio}}\right)}$ . Therefore, values smaller than 2 may lead to fewer nodes being queried if the prior counts are reasonable. We chose `split_ratio` = 1.75 and find that our algorithm never needs to go more than one layer below the created bins.

**Task Generation.** The segments/bins, generated as above, were organized randomly into Mechanical Turk HITs (Human Intelligence Tasks) having 15 images each. The workers were paid 30¢ for each HIT. Additionally, workers had the option to check ‘Too many to be counted precisely’. However, they were still expected to provide a rough estimate. Figure 3.9 shows the instructions given to workers for both datasets. Across both datasets, workers provided counts for 2250 segments. Each HIT was answered by 5 workers and then take the median of their responses as the `WorkerCount`. Getting responses from multiple workers is necessary as workers often unwittingly an-

**Instructions**

**Count the number of cells that are visible in the 15 test images displayed below.**

- The cells that need to be counted may not be completely visible in the image - they may be covered by other cells in front of them. You are still expected to count these cells.
- If the number of cells is too large to be counted precisely or if you're having trouble counting the number of cells, check the box "Too many to be counted precisely". In such cases, **you are still expected to provide an approximate/estimate count.**
- Clicking on the image will open it in a new tab/window. You can zoom in and out using your browser's controls.
- Count carefully** - Good performers will be rewarded with **bonus**.
- As an example, the following image has 5 cells.

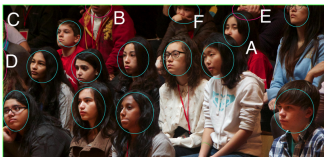


(a) Biological dataset

**Instructions**

You must count the number of heads that lie within the green box.

- If the green box is not immediately visible, check the image boundaries.
- The heads that need to be counted may not be completely visible in the image - they may be covered by other objects in front of them. You are still expected to count them.
- If the number of heads is too large to be counted precisely or if you're having trouble counting them, check the box "Too many to be counted precisely". In such cases, **you are still expected to provide an approximate/estimate count.**
- If a head crosses the green lines, it is to be counted only if more than 50% of it lies within the green box.
- Clicking on the image will open it in a new tab/window. You can zoom in and out using your browser's controls.
- Count carefully** - Good performers will be rewarded with **bonus**.
- As an example, consider the following image



- The green box is visible on the boundary of the image. (Click/zoom to see)
- The image has 13 heads, marked in blue.
- The heads marked in red are not counted.
- A is covered by another face but is still counted.
- B, C, D and E are not counted because they cross green box and more than 50% of the heads are not inside the box.
- F is counted because it crosses the green box and more than 50% of the head is inside the box.

(b) Crowd dataset

Figure 3.9: Counting Interface: Instructions shown to workers

swer questions incorrectly. We examine this behavior in greater detail in A and find that groups of workers, as a whole, are generally accurate despite making errors individually.

Given the generated segmentation trees, as well as the outcomes of the generated tasks, we are able to simulate the runs of different algorithms on the two datasets and compare them on an equal footing.

### 3.5.2 Variants of algorithms

**Algorithms for Both Datasets.** For the above datasets, we evaluate the following algorithms:

- FS: our `FrontierSeeking` algorithm from Section 3.3;
- OnlyRoot: This algorithm queries only the root node of the segmentation tree, to test how workers perform without any algorithmic decomposition;
- Optimal: Given our worker behavior model, a worker’s answer is expected to be accurate only if the number of objects to be counted is  $< d^*$ . Thus, any algorithm requires at least  $\lceil \frac{\text{TrueCount}}{d^*} \rceil$  questions to count accurately, even if it knows the exact nodes to query (i.e., it was an omniscient oracle), and also had the ability to magically create questions with  $d^*$  objects combining arbitrary portions of the image. We call this `Optimal` since it is a lower bound for any algorithm given our error model.

**Algorithms for Crowd Dataset.** For the crowd dataset, we additionally evaluate the following algorithms:

- Face-ML: a pre-trained face detector from [47], to test how just machine learning performs;

**Algorithms for Biological Dataset.** For the biological dataset, we also evaluate the following algorithms:

- Bio-ML: the sum of the prior counts from our machine learning algorithm from Section 3.4.3;
- AA-Split: the AA algorithm, aggregating the counts from the immediate next level of the implicit segmentation tree below the terminating frontier discovered by AA, revealing how much descending below the initial terminating frontier will help accuracy
- AA: our `ArticulationAvoidance` algorithm (Section 3.4.1), that stops at the initially determined terminating frontier;

**Ground Truth.** For both datasets, we denote the true counts of images by `Exact`. While the (generated) images in the biological dataset have a known ground truth, the images in the crowd dataset were evaluated independently and agreed upon by two annotators. Two images and their ground truth counts were taken from [59].

**Accuracy.** The error of our algorithms is calculated as:  $\frac{|\text{TrueCount} - \text{WorkerCount}|}{\text{TrueCount}}$ . The percentage accuracy is therefore  $100 \times (1 - \text{Error})$ . We also use the percentage of images where `TrueCount = WorkerCount` as another accuracy metric for the biological dataset.

### 3.5.3 Results

In this section, we describe the results of our algorithms when run on the two real datasets described above.

#### *How do the JellyBean algorithms compare with the theoretically optimal oracle algorithm on cost?*

**Crowd Dataset Optimality.** For the crowd dataset, we compare the performance of `FS` against `Optimal`. Averaging across images, the number of questions asked by `FS` is within a factor 2.3 of `Optimal`. Further, this factor does not cross 2.75 for any image in the dataset. This is especially low considering how hard the images in this dataset are (see Figure 3.7). `Optimal` has perfect information about the location of objects in the image, and is able to generate components having precisely  $d^*$  objects (without having to obey connectedness as with `AA` or `FS`). Note that this factor is different from competitive ratio  $CR$  defined in Section 3.3. While `Optimal` represents a theoretical lower bound on the performance of any algorithm,  $OPT$  was defined for a given segmentation tree.

**Biological Dataset Optimality.** For the biological dataset, we compare the performance of `FS` and `AA` against `Optimal`. The average number of questions asked by `AA` is within a factor 1.35 of `Optimal`, which is significantly lower than the 2.3 factor for `FS`. This indicates that leveraging information from computer vision algorithms helps bring `AA` closer to “oracle” optimality.

<i>On both datasets with hundreds of objects, the costs of algorithms <code>FS</code> and <code>AA</code> are within a small constant factor—between 1 to 2.5—of <code>Optimal</code>.</i>
--

#### *How accurate are the JellyBean algorithms relative to machine learning baselines (ML)?*

**Crowd Dataset Accuracy.** For the crowd dataset, we compare the performance of `FS` against `Face-ML`. As we have noted previously, this is an extremely challenging dataset with images that look very different from each other. As a result, `Face-ML` fails to detect any faces for 7 out of the 12 images (i.e., making 100% error on 58.3% of the

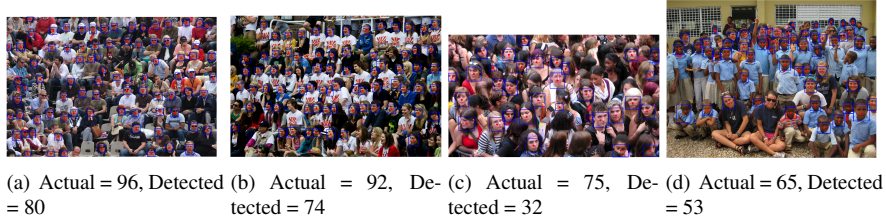


Figure 3.10: Performance of Face-ML

images), and has an average accuracy of 70.1% on the remaining, demonstrating how challenging face detection can be to state-of-the-art computer vision algorithms. In comparison, FS counts people in all these images with an average accuracy of 92.5% for an average cost of just \$1.17 per image. In particular, for the 5 images where Face-ML did detect heads (shown in Figure 3.10), the average accuracy of our algorithm was 97.5%. The accuracy of FS, which is independent of the domain and does not need dataset specific training, on this difficult dataset demonstrates that crowdsourcing can be very powerful for tasks such as counting.

**Biological Dataset Accuracy.** For the biological dataset, FS has an average accuracy of 96.4%. In fact, the minimum accuracy for all images in the dataset is 94%. Next, we compare AA to Bio-ML, our supervised, confidence-map based machine learning algorithm, whose counts and partitions we also use as input to AA. We observe that out of 20 images, AA gets the correct Exact count for 17 (85%) of the images, while the ML approach gets only 9 (45%) images exactly correct. The fact that AA gets the count correct for so many images is a further indication of how much easier this dataset is compared to the previous, due to the items being homogeneous and easily identifiable and distinguishable. To study the errors further, we plot a histogram of the deviation from the correct counts in Figure 3.11(a). The x-axis shows the deviation from Exact for an image, and the y-axis shows the frequency, or number of images for which the count estimated by an algorithm deviated for a specific x-value. Ignore the blue bar (AA-split) for now. We observe that even though the counts provided by FS are 96.4% accurate, they deviate by more than 5 for 18/20 images. AA is significantly better – only 3 images deviating by counts of 1, 2, and 3 respectively. In comparison, Bio-ML estimate deviates by at least 5 for 7 images. Thus, AA, which leverages both crowds and computer vision algorithms, outperforms both FS and Bio-ML, often being very close to or equal to the right answers.

*On the crowd dataset, FS has a much higher accuracy of 97.5% relative to 70.1% for Face-ML on the 5/12 images Face-ML works on; for the remaining 7/12 images, Face-ML detects no faces at all.*

*On the biological dataset, FS has an accuracy of 96.4%. In comparison, AA increases the accuracy to 99.87%, returning exact counts for 85% of the images (off on the rest by counts of 1 to at most 3), while Bio-ML gets only 45% correct (off on the rest by counts of at least 5).*



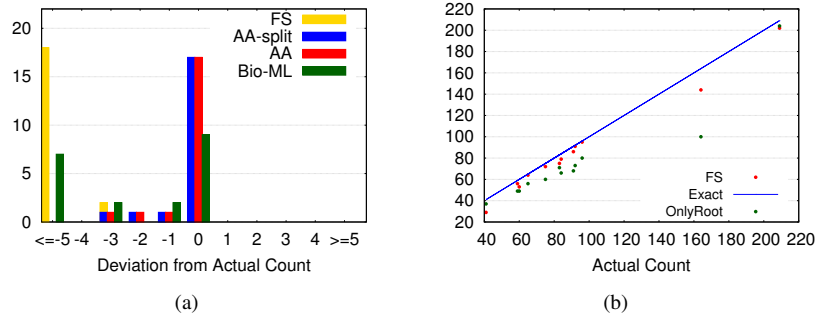


Figure 3.11: Counting Accuracy. (a) Biological Dataset: Number of Images vs. Deviation from actual counts (b) Crowd Dataset: Estimated count vs. Actual Count – Each point represents an image

### *How expensive are the Jellybean algorithms (FS and AA)?*

**Crowd Dataset Cost.** In Figure 3.12 we plot the cost of counting an image from the crowd dataset using FS against the number of objects in that image. Each vertical slice corresponds to one image with its ground truth count along the x-axis, and dollar cost incurred along the y-axis. The cost is of the order of just a *few dollars even for very large counts*, making it a viable option for practitioners looking to count (or verify the counts of) objects in an image. It is possible to reduce our costs further by using additional heuristics. For example, if we get multiple responses with `WorkerCount`  $\gg d^*$ , we may choose not to present this node to more workers — small deviations in the `WorkerCount` for this node are not going to impact our traversal strategy. Another possible heuristic could be to show the segmentation tree to workers and have them point out nodes where the number of objects is close to  $d^*$ . We leave the exploration of such additional heuristics for future work.

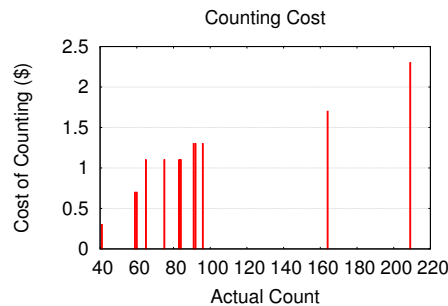


Figure 3.12: Cost of counting for the crowd dataset

**Biological Dataset Cost.** The average cost of counting an image from the biological dataset incurred using AA was \$1.6, as compared to \$2.7 using FS. The average cost of counting per object was 0.63¢ for AA and 1.25¢ for FS. This significant reduction (2×) for AA is a direct result of our merging algorithm which skips the larger granularity image segments and directly elicits accurate counts on the generated components.

*On both datasets with hundreds of objects, the algorithms FS and AA return accurate results at the cost of a few dollars per image. The cost of AA is approximately half of the cost of FS per object, indicating that “skipping ahead” in the segmentation tree using information from computer vision algorithms followed by merging partitions cleverly helps reduce cost significantly.*

**How accurate are the JellyBean algorithms (FS and AA) relative to directly asking workers to count on the entire image (OnlyRoot)?**

**Crowd dataset.** We now compare OnlyRoot, i.e., only asking questions at the root, versus FS. We plot the results in Figure 3.11(b). The x-axis marks the ground truth (Exact) counts of images, while the y-axis plots the predicted counts by different algorithms. Each vertical slice corresponds to an image, and each point on the plot corresponds to the output of an algorithm for a given input image. We find that *average accuracy of OnlyRoot is 81.8% as compared to the 92.5% of FS*. Furthermore, on the 5 images that the pre-trained classifier worked on, OnlyRoot *has an average accuracy of only 82.4% as compared to 97.5% accuracy of FS*. We observe that performing our top-down expansion of the segmentation tree and splitting the image into smaller pieces improves counts significantly for most images. Our algorithm performs consistently better in terms of counts. The only image for which the baseline is better has TrueCount = 209 – fairly close to the approximate number (200) provided by workers for the whole image.

**Biological Dataset.** For the biological dataset, the OnlyRoot baseline performs poorly, achieving an accuracy of < 75% for 14/20 images. In comparison, FS counts with an accuracy of 96.4% for all images. Further, AA has 100% accuracy on 17/20 images as shown in Figure 3.11(a), indicating that using vanilla crowdsourcing without applying our JellyBean algorithms can lead to low accuracy.

*On the crowd dataset, FS estimates counts with > 90% accuracy on 9/12 images, relative to 2/12 images for OnlyRoot. Further, on the 5 challenging images where Face-ML failed, FS provides a significant 15% higher accuracy compared to OnlyRoot.*

*On the biological dataset, FS and AA improve the accuracy by 27% and 30.7% as compared to OnlyRoot.*

**Do we get any additional benefit from counting nodes below those we determine as the terminating frontier?**

*There is no benefit, at least for biological dataset.*

**Biological dataset.** For the biological dataset, we already start with a set of merged image segments (or components) corresponding to the output of our AA algorithm. We have seen above (Figure 3.11(a)) that aggregating worker count estimates from just each of these components yields accurate results (17 out of 20 images exactly correct).

We now explore whether going a level below this initial terminating frontier in this tree helps, i.e., does splitting these components into smaller sub-segments and eliciting worker counts on these smaller sub-images help. Note that while this means a strictly higher cost because we query a larger number of image segments, our experiments on the crowd dataset have shown that splitting segments, or traversing the segmentation tree, gives the potential for improved accuracy. The full details of this splitting algorithm, titled `AA-Split` are discussed in Section 3.4. The output from this algorithm corresponds to the blue bar in Figure 3.11(a). Based on the figure, we observe that this algorithm gives us no improvement over just querying our original components, in spite of showing smaller image segments and incurring a higher cost. We attribute this to two factors — (1) the `TrueCounts` in components created by `AA` are close to 20, and (2) our estimate of  $d^* = 20$  is conservative. Together these imply that the worker counts obtained on the components created by `AA` are accurate by themselves. So, while splitting the merged components could be beneficial for certain datasets, for the purpose of this work, just our `AA` algorithm with worker counts on generated components is sufficient for the biological dataset.

### 3.5.4 Aggregating Worker Answers

As noted previously, individual workers make mistakes but groups of workers are generally ‘good’. However, having responses from multiple workers necessitates a way to aggregate their answers. In this section, we provide experiment results for different aggregation functions, including `median` that we have seen performs well.

Mutual agreement among the workers’ responses is a useful indicator of worker response quality, i.e., `WorkerCounts` having ‘high’ agreement are trustworthy, as multiple workers have independently verified this count. We quantify this notion of ‘high’ agreement with the fraction of workers who agree on a particular count. We return the `WorkerCount` having maximum agreement as the consolidated count, if this fraction is greater than a threshold  $a$ . Otherwise, if there is disagreement, we require another scheme to aggregate the different responses elicited from the workers. In particular, we experimented with `mean`, `median` and `maximum` denoted by `avg`, `mid` and `max` respectively. These aggregation functions have been evaluated on (i) accuracy and (ii) cost for both datasets in Figure 3.13(a) and Figure 3.13(b) respectively.

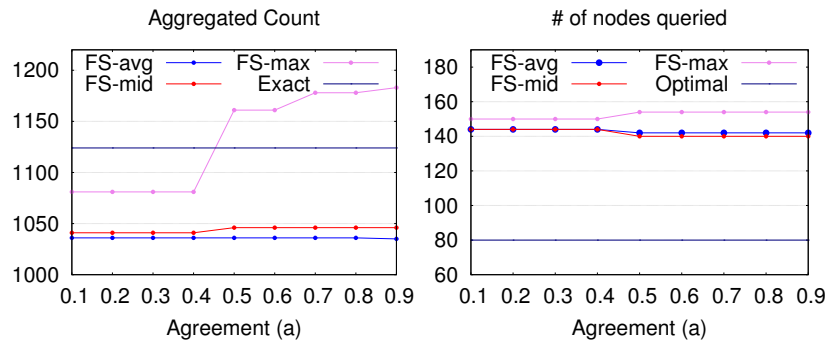
In these figures, the x-axis shows the agreement threshold  $a$  as noted previously. For the plots denoting ‘Aggregated Count’, the y-axis shows the count obtained by aggregating worker answers, summed over all images in the respective dataset. The prior counts have also been shown along with the ground truth for the biological dataset. For the plots denoting ‘# of nodes queried’, the y-axis shows the total number of nodes queried in the segmentation trees of these images. We make the followings observations:

- `max`: Higher values of agreement are rare. This implies that for higher  $a$ , our algorithm ends up using the `max` aggregation for more segments – causing the

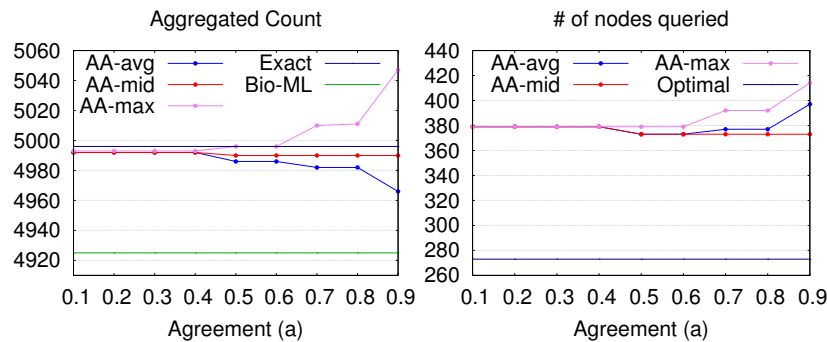
consolidated count to increase with  $a$ . The number of nodes queried also increases with  $a$  for the same reason. We observe that the number of nodes queried by max aggregation is always more than median and avg. This is because the chosen worker answer tends to be above the threshold and hence max goes deeper in the segmentation tree.

- avg: The count aggregated by avg decreases with the increase in  $a$ . This is explained by the fact that workers tend to err on the lower side of TrueCount more often as we see in Appendix A.
- mid: For both counts and number of nodes queried, mid is insensitive to changes in agreement threshold  $a$ . This insensitivity to agreement threshold  $a$  is important, as practitioners don't have to tune any parameter.

The above experiments with different combinations of agreement threshold  $a$  and aggregation functions indicate that  $a = 0.5$  and taking median of worker responses works best. Note that median aggregation for  $a \geq 0.5$  is equivalent to simply taking the median of all worker responses.



(a) Crowd dataset: Counting accuracy and # of nodes queried



(b) Biological dataset: Counting accuracy and # of nodes queried

Figure 3.13: Aggregating worker answers

## 3.6 Related Work

The general problem of finding, identifying, or counting objects in images has been studied in machine learning, computer vision and crowdsourcing communities. Many of the existing techniques, however, make limiting assumptions on the types of images, or are heavily dependent on large volumes of training data. We discuss recent related work from each of these areas and compare them against our approach.

**Unsupervised learning.** A number of recent solutions to object counting problems tackle the challenge in an unsupervised way, grouping neighboring pixels together on the basis of self-similarities [60], or similarities in motion paths [61]. Another work [62] leverages depth information from vertical Kinect sensors to count the number of people entering or leaving a room. However, unsupervised methods have limited accuracy, and the computer vision community has therefore considered supervised learning approaches. These fall into three categories:

**Counting by detection.** In this category of supervised counting algorithms, a object detector is used to localize each object instance in the image. Training data for these algorithms is typically in the form of images annotated by bounding boxes for each object. Once all objects have been located, counting them is trivial. Most of the current object detectors (including our approach detailed in Section 3.4) operate in two steps — generating a *confidence map* that gives the probability of a pixel belonging to an object, followed by *thresholding* and *non-maxima suppression* to find locations of the objects [58]. However, object detection is an unsolved problem in itself even though progress has been made in recent years [44].

**Counting by regression.** Algorithms in this category bypass the detection phase and try to learn a mapping from image properties like texture to the number of objects. This mapping is inferred using one of the large number of available regression algorithms in machine learning e.g., neural networks [63, 64]. For training, images are provided with corresponding object counts. In such methods [?], the mappings from local features to counts are global, that is, a single function’s learned parameters are used to estimate counts for the entire image or video. This works well when crowd densities are uniform throughout the image or video — a limiting assumption that is largely violated in real life applications because of perspective, non-uniform density and changes in viewpoint. A characteristic feature of perspective is that objects become smaller in images, as their distance from the recording camera increases. For example, in Figure 3.1, the density of crowd in bottom-right (closer to camera) is much less than in the top part of the image.

**Counting by annotation.** A third approach has been to train on images annotated with dots. Instead of bounding boxes, each object here is annotated with a dot. For instance, in [67], an image density function is estimated, whose integral over a region in the image gives the object count. Another recent work accomplishes the counting task in

extremely dense crowds by leveraging the repetitive nature of such crowded images [59]. The techniques in this work are however only applicable in densely crowded images.

A common theme across these methods is that they deliver accurate counts when their underlying assumptions are met but are not applicable in more challenging situations. This guides us to leverage the ‘wisdom of the crowds’ in counting heterogeneous objects, which may be severely *occluded* by objects in front of them.

**Image segmentation.** Segmentation of images into meaningful chunks is an active area of research in the computer vision community. Approaches for counting pedestrians for surveillance in images and videos involve segmentation based on motion [68]. Foreground extraction has also been dealt with in [65], where pedestrians are being counted. Text extraction [60] can also be used to get foreground blobs, aiding in segmentation. Ahmed et. al. [69] is another recent work on interactive segmentation, where users provide tags for the objects to be segmented. An image database is then queried using this tag to gather exemplars that help localize the object in image. Modified versions of watershed transform have been shown to segment microorganisms in images [70]. Multi-level active learning approaches have been investigated [71] to leverage human annotations at varying levels of richness and manual effort.

**Crowdsourcing for image/video analysis.** The above considerations indicate the requirement of human input in the object counting pipeline. The idea of using human inputs for complex learning tasks has recently received attention; in [72], the authors present a hybrid crowd-machine classifier where crowds are involved in both feature extraction and learning. Although crowdsourcing has been extensively used on images for tasks like tagging [73], quality assessment [74] and content moderation [75], the involvement of crowds in image analysis has been largely restricted to generating training data [76, 77]. In a recent work, human answers to questions like ‘Does this bird have a blue belly?’ were combined with image analysis techniques to classify bird images into species [78]. In [79], the authors get discriminative features from the crowd for bird classification using an online game. Another approach is to ask workers to describe the differences between pairs of images [80]. Human input has also been explored for tracking objects in videos [81]. This enables detection of objects at interactive speed.

In a recent feasibility study of crowdsourcing for malaria image analysis [82], non-expert players achieved a counting accuracy of more than 99%. In our work, we build on this feasibility study to identify and propose solutions to the challenges that arise when using crowds to estimate counts in images across different application settings. Our approach is aligned with the findings in [83]—global decisions (counting) are possible with each worker seeing a small part of the input image. Recent work [84] has examined the use of humans for large-scale selectivity estimation. For example, they display 100 images with a single person in each image, and ask workers how many of these images contain old individuals. While operating on more images at a time, the

paper addresses a simpler problem since each image only needs to be evaluated as a YES/NO, as opposed to our case, where a count is desired.

**Summary.** While there have been many studies on computer vision for counting and segmentation, either (a) the described settings are stylized or make application-specific limiting assumptions, or (b) the designed algorithms have relatively low accuracy in practice. Compared to the computer vision algorithms described, our approach to count objects is generic—it can be used to count heterogeneous, occluded objects in diverse images. From the crowd research community, to the best of our knowledge, there has been no principled study of the cost-accuracy trade-off for optimizing complex computer vision tasks. Given the accuracy of our algorithms, this makes our work the first step towards efficient generation of large scale accurate training data for complex computer vision problems.

### 3.7 Summary

We tackle the challenging problem of counting the number of objects in images, a ubiquitous, fundamental problem in computer vision. While humans and computer vision algorithms, separately, are highly error-prone, our JellyBean suite of algorithms combine the best of their capabilities to deliver high accuracy results at relatively low costs for two separate regimes or modes, while additionally providing optimality guarantees under reasonable assumptions that we validate in the thesis.

Our JellyBean algorithms were shown to (a) be within a  $2.75\times$  factor of the best possible oracle algorithm in terms of cost when operating without computer vision help, and within a  $1.3\times$  factor of the best possible oracle algorithm, with average cost reduced by almost half, when operating in concert with computer vision and employing articulation-avoiding partition merging schemes, (b) have high accuracy relative to both computer vision baselines (e.g., 18% increase in accuracy on images where the baselines detect faces) as well as vanilla crowdsourcing (e.g., 15% increase in accuracy).

While our algorithms already have low cost and high accuracy, we believe that additional techniques, such as (a) human-aided partitioning of images, (b) better interfaces to remove the ambiguity of “partial” objects, (c) asking fewer questions on images with existing high count estimates, (d) using humans to “skip-ahead” in a segmentation tree, can further reduce cost. We leave the exploration of these directions as future work.

# CHAPTER 4

## Conclusions

In this thesis, we developed crowd-powered open-ended data processing algorithms for fundamental problems of organization and counting. For both problems, we started by investigating human behavior. The findings of these investigations led us to develop accurate models of human behavior — for clustering, it allowed us to formalize the notion of granularities and perspectives; whereas for counting, it helped us characterize counting errors. With these worker behavior models, we formulated the open-ended tasks as algorithmic problems.

For clustering, we developed cost-efficient, accurate algorithms for identifying the consensus organization and incorporated this workflow into a cost-effective and robust workflow for organizing a collection of objects. We demonstrated that ORCHESTRA organizes items at  $24\times$  higher accuracy for the same cost. Similarly, for counting, we showed that our algorithms were theoretically optimal or near-optimal, in that they ask as few questions as possible to humans; and they perform very well in practice, increasing counting accuracy by at least 15% on images that no individual worker or computer vision algorithm can count correctly, while not incurring a high cost.

Our work in this thesis has demonstrated the benefits of open-ended crowdsourcing. By making the best use of human ability and time, open-ended tasks have the potential to provide accurate training data at scale for complex applications while incurring low costs. But for this potential to be realized, researcher efforts need to be devoted to understanding the applications, develop open-ended operators and optimize them to ensure quality. Some other applications that can vastly benefit from open-ended crowdsourcing are (i) translation – asking humans to provide correspondances between words in the translated sentence and the translation (in addition to the translation itself) can help us develop more fine-grained language understanding systems, (ii) detection – asking humans to draw polygons around specific objects in images is useful for training accurate detection algorithms in computer vision, but this isn't yet fully understood and optimized by the crowdsourcing community.



## REFERENCES

- [1] V. K. Kirpalani and M. E. Tayab, “Enhancing data quality using human computation and crowd sourcing,” *Journal of Independent Studies and Research*, vol. 13, no. 1, p. 74, 2015.
- [2] C. Eickhoff, “Crowd-powered experts: Helping surgeons interpret breast cancer images,” in *Proceedings of the First International Workshop on Gamification for Information Retrieval*. ACM, 2014, pp. 53–56.
- [3] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, “Crowder: Crowdsourcing entity resolution,” *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1483–1494, 2012.
- [4] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, “Crowdscreen: Algorithms for filtering data with humans,” in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 361–372.
- [5] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, “Human-powered sorts and joins,” *Proceedings of the VLDB Endowment*, vol. 5, no. 1, pp. 13–24, 2011.
- [6] A. D. Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy, “Crowd-powered find algorithms,” in *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 964–975.
- [7] S. Guo, A. Parameswaran, and H. Garcia-Molina, “So who won?: dynamic max discovery with the crowd,” in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 385–396.
- [8] V. Polychronopoulos, L. De Alfaro, J. Davis, H. Garcia-Molina, and N. Polyzotis, “Human-powered top-k lists,” in *WebDB*, 2013, pp. 25–30.
- [9] X. Liu, P. H. Tu, J. Rittscher, A. Perera, and N. Krahnstoever, “Detecting and counting people in surveillance applications,” in *AVSS, 2005*. IEEE, 2005, pp. 306–311.
- [10] C. G. Loukas, G. D. Wilson, B. Vojnovic, and A. Linney, “An image analysis-based approach for automated counting of cancer cell nuclei in tissue sections,” *Cytometry part A*, vol. 55, no. 1, pp. 30–42, 2003.
- [11] J. Russell, S. Couturier, L. Sopuck, and K. Ovaska, “Post-calving photo-census of the rivièrè george caribou herd in july 1993,” *Rangifer*, vol. 16, no. 4, pp. 319–330, 1996.
- [12] A. D. Sarma, A. Jain, A. Nandi, A. Parameswaran, and J. Widom, “Surpassing humans and computers with jellybean: Crowd-vision-hybrid counting algorithms,” in *Third AAAI Conference on Human Computation and Crowdsourcing*, 2015.

- [13] S. Imai and W. Garner, “Discriminability and preference for attributes in free and constrained classification.” *Journal of Experimental Psychology*, vol. 69, no. 6, p. 596, 1965.
- [14] 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*, vol. 21, no. 2, p. 347, 1995.
- [15] S. Handel and S. Imai, “The free classification of analyzable and unanalyzable stimuli.” *Perception & Psychophysics*, 1972.
- [16] D. L. Medin, W. D. Wattenmaker, and S. E. Hampson, “Family resemblance, conceptual cohesiveness, and category construction,” *Cognitive psychology*, vol. 19, no. 2, pp. 242–279, 1987.
- [17] F. Milton and A. Wills, “The influence of stimulus properties on category construction.” *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 30, no. 2, p. 407, 2004.
- [18] R. G. Gomes, P. Welinder, A. Krause, and P. Perona, “Crowdclustering,” in *Advances in neural information processing systems*, 2011, pp. 558–566.
- [19] 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*, 2012, pp. 1772–1780.
- [20] 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*, vol. 2, 2012.
- [21] R. E. Bechhofer, S. Elmaghraby, and N. Morse, “A single-sample multiple-decision procedure for selecting the multinomial event which has the highest probability,” *The Annals of Mathematical Statistics*, pp. 102–119, 1959.
- [22] M. Liedloff, “Finding a dominating set on bipartite graphs,” *Information Processing Letters*, vol. 107, no. 5, pp. 154–157, 2008.
- [23] 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*, vol. 2. IEEE, 2005, pp. 524–531.
- [24] 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*. IEEE, 2009, pp. 248–255.
- [25] H. Heikinheimo and A. Ukkonen, “The crowd-median algorithm,” in *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.
- [26] 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*. ACM, 2013, pp. 225–236.
- [27] Y. Yue, C. Wang, K. El-Arini, and C. Guestrin, “Personalized collaborative clustering,” in *Proceedings of the 23rd international conference on World wide web*. ACM, 2014, pp. 75–84.
- [28] 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*. ACM, 2013, pp. 1999–2008.

- [29] 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. [Online]. Available: <http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7560>
- [30] Y. Sun, A. Singla, D. Fox, and A. Krause, “Building hierarchies of concepts via crowdsourcing,” *arXiv preprint arXiv:1504.07302*, 2015.
- [31] D. Karampinas and P. Triantafillou, “Crowdsourcing taxonomies,” in *The semantic web: Research and applications*. Springer, 2012, pp. 545–559.
- [32] A. Biswas and D. Jacobs, “Active image clustering with pairwise constraints from humans,” *International Journal of Computer Vision*, vol. 108, no. 1-2, pp. 133–147, 2014.
- [33] S. Lad and D. Parikh, “Interactively guiding semi-supervised clustering via attribute-based explanations,” in *Computer Vision—ECCV 2014*. Springer, 2014, pp. 333–349.
- [34] 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.
- [35] O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. T. Kalai, “Adaptively learning the crowd kernel,” *arXiv preprint arXiv:1105.1033*, 2011.
- [36] 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*, vol. 4, no. 5, pp. 267–278, 2011.
- [37] J. Fan, G. Li, B. C. Ooi, K.-I. Tan, and J. Feng, “icrowd: An adaptive crowdsourcing framework,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1015–1030.
- [38] 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 JournalThe International Journal on Very Large Data Bases*, vol. 22, no. 5, pp. 711–726, 2013.
- [39] S. E. Whang, J. McAuley, and H. Garcia-Molina, “Compare me maybe: Crowd entity resolution interfaces.”
- [40] S. E. Whang, P. Lofgren, and H. Garcia-Molina, “Question selection for crowd entity resolution,” *Proceedings of the VLDB Endowment*, vol. 6, no. 6, pp. 349–360, 2013.
- [41] N. Vesdapunt, K. Bellare, and N. Dalvi, “Crowdsourcing algorithms for entity resolution,” *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1071–1082, 2014.
- [42] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 2003.
- [43] R. Szeliski, *Computer vision: Algorithms and Applications*. Springer Science & Business Media, 2010.
- [44] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes challenge: A retrospective,” *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2014.

- [45] A. Nguyen, J. Yosinski, and J. Clune, “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images.” in *Proceedings of Computer Vision and Pattern Recognition*. IEEE, 2015.
- [46] A. E. Carpenter, T. R. Jones, M. R. Lamprecht, C. Clarke, I. H. Kang, O. Friman, D. A. Guertin, J. H. Chang, R. A. Lindquist, J. Moffat et al., “Cellprofiler: image analysis software for identifying and quantifying cell phenotypes,” *Genome biology*, vol. 7, no. 10, p. R100, 2006.
- [47] X. Zhu and D. Ramanan, “Face detection, pose estimation, and landmark localization in the wild,” in *CVPR, 2012*. IEEE, 2012, pp. 2879–2886.
- [48] G. A. Miller, “The magical number seven, plus or minus two: some limits on our capacity for processing information.” *Psychological review*, vol. 63, no. 2, p. 81, 1956.
- [49] NBC News, “Exhibit traces the history of the voting rights act,” in *nbc-news.com/id/8839169/ns/us\_news-life/t/exhibit-traces-history-voting-rights-act*, 2005.
- [50] A. Borodin, *Online computation and competitive analysis*, 1998, vol. 2.
- [51] D. R. Karger, S. Oh, and D. Shah, “Iterative learning for reliable crowdsourcing systems,” in *NIPS*, 2011, pp. 1953–1961.
- [52] V. S. Sheng, F. Provost, and P. G. Ipeirotis, “Get another label? improving data quality and data mining using multiple, noisy labelers,” in *Proceedings of the 14th ACM SIGKDD*. ACM, 2008, pp. 614–622.
- [53] A. Lehmussola, P. Ruusuvuori, J. Selinummi, H. Huttunen, and O. Yli-Harja, “Computational framework for simulating fluorescence microscope images with cell populations,” *Medical Imaging, IEEE Transactions on*, vol. 26, no. 7, 2007.
- [54] M. Dyer and A. Frieze, “On the complexity of partitioning graphs into connected subgraphs,” *Discrete Applied Mathematics*, vol. 10, no. 2, pp. 139–153, 1985.
- [55] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, “Approximation algorithms for bin packing: a survey,” in *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996, pp. 46–93.
- [56] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.
- [57] S. Beucher and F. Meyer, “The morphological approach to segmentation: the watershed transformation,” *Optical Engineering-New York-Marcel Dekker Incorporated-*, vol. 34, pp. 433–433, 1992.
- [58] T. W. Nattkemper, H. Wersing, W. Schubert, and H. Ritter, “A neural network architecture for automatic segmentation of fluorescence micrographs,” *Neurocomputing*, vol. 48, no. 1, pp. 357–367, 2002.
- [59] H. Idrees, I. Saleemi, C. Seibert, and M. Shah, “Multi-source multi-scale counting in extremely dense crowd images,” in *CVPR*. IEEE, 2013.
- [60] N. Ahuja and S. Todorovic, “Extracting texels in 2.1 d natural textures,” in *ICCV 2007*. IEEE, 2007, pp. 1–8.
- [61] V. Rabaud and S. Belongie, “Counting crowded moving objects,” in *CVPR, 2006*, vol. 1. IEEE, 2006, pp. 705–711.

- [62] X. Zhang, J. Yan, S. Feng, Z. Lei, D. Yi, and S. Z. Li, “Water filling: Unsupervised people counting via vertical kinect sensor,” in *AVSS, 2012*. IEEE, 2012, pp. 215–220.
- [63] S.-Y. Cho, T. W. Chow, and C.-T. Leung, “A neural-based crowd estimation by hybrid global learning algorithm,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 29, no. 4, pp. 535–541, 1999.
- [64] A. Marana, S. Velastin, L. Costa, and R. Lotufo, “Estimation of crowd density using image processing,” in *Image Processing for Security Applications (Digest No.: 1997/074), IEE Colloquium on*. IET, 1997, pp. 11–1.
- [65] D. Ryan, S. Denman, C. Fookes, and S. Sridharan, “Crowd counting using multiple local features,” in *DICTA, 2009*. IEEE, 2009, pp. 81–88.
- [66] D. Kong, D. Gray, and H. Tao, “Counting pedestrians in crowds using viewpoint invariant training,” in *BMVC*. Citeseer, 2005.
- [67] V. Lempitsky and A. Zisserman, “Learning to count objects in images,” in *Advances in Neural Information Processing Systems*, 2010, pp. 1324–1332.
- [68] A. B. Chan, Z.-S. Liang, and N. Vasconcelos, “Privacy preserving crowd monitoring: Counting people without people models or tracking,” in *CVPR 2008*. IEEE, 2008, pp. 1–7.
- [69] E. Ahmed, S. Cohen, and B. Price, “Semantic object selection,” in *CVPR, 2014*. IEEE, 2014, pp. 3150–3157.
- [70] S. Bergner, R. Pohle, S. Al-Zubi, K. Tönnies, A. Eitner, and T. R. Neu, “Segmenting microorganisms in multi-modal volumetric datasets using a modified watershed transform,” in *Pattern Recognition*. Springer, 2002, pp. 429–437.
- [71] S. Vijayanarasimhan and K. Grauman, “Multi-level active prediction of useful image annotations for recognition,” in *Advances in Neural Information Processing Systems*, 2009, pp. 1705–1712.
- [72] J. Cheng and M. S. Bernstein, “Flock: Hybrid crowd-machine learning classifiers,” in *CSCW '15*, 2015.
- [73] C. Qin, X. Bao, R. Roy Choudhury, and S. Nelakuditi, “Tagsense: a smartphone-based approach to automatic image tagging,” in *MobiSys*. ACM, 2011, pp. 1–14.
- [74] F. Ribeiro, D. Florencio, and V. Nascimento, “Crowdsourcing subjective image quality evaluation,” in *ICIP*. IEEE, 2011, pp. 3097–3100.
- [75] A. Ghosh, S. Kale, and P. McAfee, “Who moderates the moderators?: crowd-sourcing abuse detection in user-generated content,” in *EC, 2011*. ACM, 2011, pp. 167–176.
- [76] A. Sorokin and D. Forsyth, “Utility data annotation with amazon mechanical turk,” in *CVPR Workshops*, 2008.
- [77] W. S. Lasecki, Y. C. Song, H. Kautz, and J. P. Bigham, “Real-time crowd labeling for deployable activity recognition,” in *CSCW, 2013*. ACM, 2013.
- [78] S. Branson, G. Van Horn, C. Wah, P. Perona, and S. Belongie, “The ignorant led by the blind: A hybrid human–machine vision system for fine-grained categorization,” *International Journal of Computer Vision*, pp. 1–27, 2014.

- [79] J. Deng, J. Krause, and L. Fei-Fei, “Fine-grained crowdsourcing for fine-grained recognition,” in *CVPR, 2013*. IEEE, 2013, pp. 580–587.
- [80] S. Maji, “Discovering a lexicon of parts and attributes,” in *ECCV 2012. Workshops and Demonstrations*. Springer, 2012, pp. 21–30.
- [81] Y. Wei, J. Sun, X. Tang, and H.-Y. Shum, “Interactive offline tracking for color objects,” in *ICCV 2007*. IEEE, 2007, pp. 1–8.
- [82] M. A. Luengo-Oroz, A. Arranz, and J. Frean, “Crowdsourcing malaria parasite quantification: an online game for analyzing images of infected thick blood smears,” *Journal of medical Internet research*, vol. 14, no. 6, 2012.
- [83] V. Verroios and M. S. Bernstein, “Context trees: Crowdsourcing global understanding from local views,” in *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.
- [84] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh, “Counting with the crowd,” *Proceedings of the VLDB Endowment*, vol. 6, no. 2, pp. 109–120, 2012.

## APPENDIX A

### Worker Behavior in Counting

Our counting algorithms rely on the assumption that workers can count images with up to a threshold of  $d^*$  objects correctly, but make mistakes for images with larger counts. In this section, we provide experimental evidence to justify this model.

We plot the worker errors for the biological dataset in Figure A.1. In Figure A.1(a), each integer value on x-axis denotes a unique worker. The corresponding bar-chart shows the percentage of questions where  $\text{WorkerCount} \neq \text{TrueCount}$  in red bars, and the percentage of questions where  $\text{WorkerCount} < \text{TrueCount}$  in green bars. Similarly, in Figure A.1(b), the y-axis shows the the mean of the errors made by each worker. Finally, in Figure A.1(c), we show the  $\text{TrueCounts}$  on x-axis. The y-axis shows the percentage of workers whose  $\text{WorkerCount} \neq \text{TrueCount}$ . We note the following observations from these plots:

- Figure A.1(a) shows that workers err more often on the lower side of  $\text{TrueCount}$ . Coupled with the observation from Figure A.1(b) that average errors are negative for most workers suggests that negative errors are a big part of overall worker errors.
- Figure A.1(b) shows that the mean of the worker errors are negative. This suggests that workers tend to err more heavily on the lower side of  $\text{TrueCount}$ . Moreover, we also observed that workers erred more *frequently* on the negative side.
- Workers make errors on all possible counts, as can be seen from A.1(c). This is because some workers are spammers — they do the bare minimum to complete the HIT, or they do not understand the task correctly. Even for a single object in the image, there is a small fraction of workers who make mistakes. Asking multiple workers is thus necessary.
- Even though individual workers make mistakes, the group as a whole is good — in Figure A.1(c), the worker error rates do not cross 35% for any value of  $\text{TrueCount}$ .

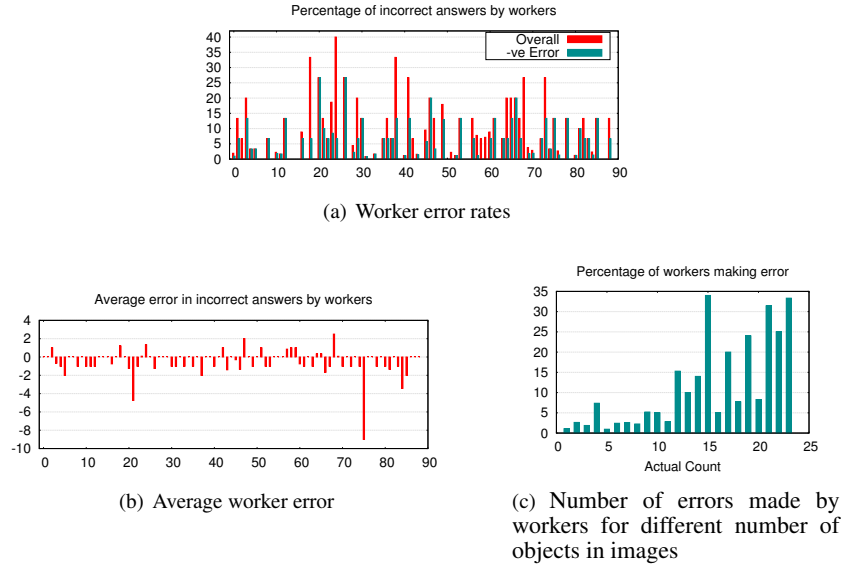


Figure A.1: Worker errors on biological dataset

**Self-confidence in Answers.** We also explored worker behavior using another set of artificially generated biological images. We generated images having 5 to 75 cells (objects) across them and distributed them to 30 workers on Amazon Mechanical Turk, who were then asked to count the number of objects in each image. Workers were also provided the option of selecting a checkbox labeled “Too many to be counted precisely”, in which case they were asked to additionally provide a rough estimate. The results of this experiment are summarized in Table A.1.

We plot the probability of a worker making an error in Figure A.2(a), where the fraction of workers making error (y-axis) has been plotted against the number of objects in the image (x-axis). We observe that a significant number of workers start making errors for images with count threshold 20. For images with counts higher than 35, their answers cannot be trusted. Based on this evidence, we propose a threshold of  $d^* = 20$ . Additionally, as can be seen from Figure A.2(a), the magnitude of error also increases with the number of objects in the displayed image.

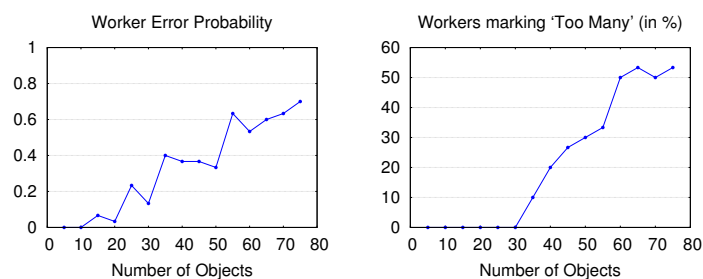
Figure A.2(b) shows the percentage of workers who checked “Too many to be counted precisely”. Beyond 35 objects, workers felt that the number of objects were too many. We also observed that for images with fewer than 60 objects, 75% of the errors were made by workers who did not check this option. This implies that having the workers express confidence in their counts and using this additional information will not help.

Additionally, in Figure A.2(c), we show the distribution of HIT Completion time for the workers. The x-axis denotes the time, and corresponding y-value indicates the percentage of workers who were able to complete the HIT within that time. The median time to HIT completion was around 13 minutes.



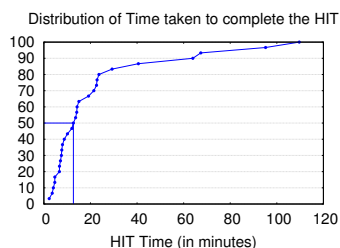
# objects	Incorrect	Too Many	Confident & Incorrect
5	0	0	0
10	0	0	0
15	2	0	2
20	1	0	1
25	7	0	7
30	4	0	4
35	12	3	10
40	11	6	8
45	11	8	7
50	10	9	7
55	19	10	12
60	16	15	7
65	18	16	6
70	19	15	7
75	21	16	10

Table A.1: Validation of Interaction Model and Worker Error Model. Column 1 is the number of objects in the image, Column 2 is the number of workers who counted incorrectly, Column 3 is the number of workers who checked “Too many to be counted precisely”, Column 4 is the number of workers who did not check “Too many to be counted precisely” but still made mistakes, Columns 2,3 and 4 are all out of 30



(a) Error Probability of Workers

(b) % of workers who checked “Too Many to be counted precisely”



(c) Time taken by workers to complete the HIT

Figure A.2: Worker Behavior