Cascade: Crowdsourcing Taxonomy Creation

Lydia B. Chilton[1], Greg Little[2], Darren Edge[3], Daniel S. Weld[1], James A. Landay[1]

[1]University of Washington [2]oDesk Research [3]Microsoft Research Asia
hmslydia@cs.uw.edu, glittle@odesk.com, Darren.Edge@microsoft.com, {weld, landay}@cs.uw.edu

ABSTRACT

Taxonomies are a useful and ubiquitous way of organizing information. However, creating organizational hierarchies is difficult because the process requires a global understanding of the objects to be categorized. Usually one is created by an individual or a small group of people working together for hours or even days. Unfortunately, this centralized approach does not work well for the large, quickly-changing datasets found on the web. Cascade is an automated workflow that creates a taxonomy from the collective efforts of crowd workers who spend as little as 20 seconds each. We evaluate Cascade and show that on three datasets its quality is 80-90% of that of experts. The cost of Cascade is competitive with expert information architects, despite taking six times more human labor. Fortunately, this labor can be parallelized such that Cascade will run in as fast as five minutes instead of hours or days.

Author Keywords

Crowdsourcing; human computation; algorithm; information architecture.

ACM Classification Keywords

H.5.3. Information interfaces and presentation (e.g., HCI): Web-based interaction.

INTRODUCTION

Although taxonomies are a useful and ubiquitous way of organizing information, creating these organizational hierarchies is difficult because the process requires a global understanding of the objects to be categorized. Currently, most taxonomies are created by a small group of experts who analyze a complete dataset before identifying the essential distinctions for classification. Unfortunately, this process is too expensive to apply to many of the user-contributed datasets, e.g. of photographs or answers to questions, found on the Internet. Despite recent progress, completely automated methods, such as Latent Dirichlet Allocation (LDA) and related AI techniques, produce low-quality taxonomies. They lack the common sense and language abilities that come naturally to people.

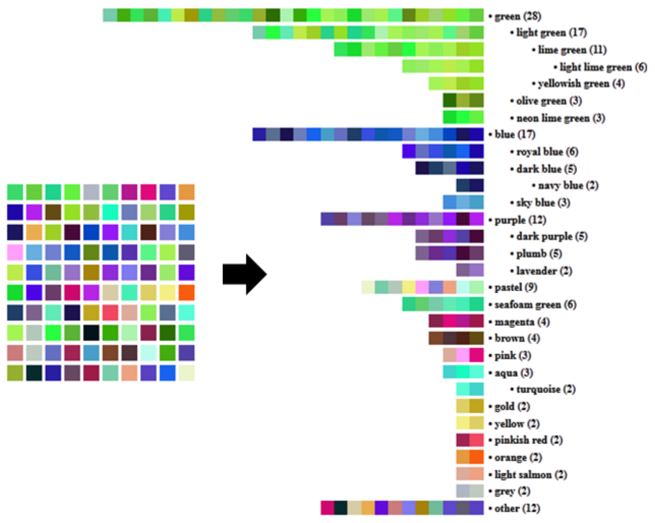


Figure 1. Example input and output of Cascade. The input is 100 random colors; the output is a taxonomy of the colors.

This paper presents Cascade, a novel method for creating taxonomies. Cascade is a crowd-algorithm that coordinates human labor with automated techniques. Each worker need only make a small contribution with no long-term time commitment or learning curve. None of the workers needs to have a global perspective of the data or the taxonomy under construction. Figure 1 contains example results.

Crowdsourcing has become a popular way to solve problems that are too hard for today's AI techniques, such as translation, linguistic tagging, and visual interpretation. Most successful crowdsourcing systems operate on problems that naturally break into small units of labor, e.g., labeling millions of independent photographs. However, taxonomy creation is much harder to decompose, because it requires a global perspective. Cascade is a unique, iterative workflow that emergently generates this global view from the distributed actions of hundreds of people working on small, local problems.

In this paper we first describe the lessons learned from early, failed attempts to design a crowd workflow for building taxonomies. We next present the Cascade algorithm and the three human intelligence task (HIT) primitives used to implement it. We demonstrate the results of running Cascade on five representative data sets. We evaluate Cascade in three ways: we compare its time and cost to that of four expert information architects who were paid to taxonomize the same data, we count the number of mistakes in Cascade's output and interpret it as an error rate, and we compare the coverage of the categories in Cascade to those of the expert-made taxonomies.

This paper makes the following contributions:

1. We present Cascade: a novel *crowd algorithm* that produces a *global understanding of large datasets.* Cascade is an online algorithm that can update the taxonomy as new data arrives. The tasks given to workers are quick and parallelizable.
2. We propose three *HIT primitives*, simple task interfaces, used as building blocks by Cascade and useful for future crowd algorithms.
3. We introduce *Global Structure Inference* as a way to combine independently-generated judgments into a cohesive taxonomy.
4. We evaluate Cascade on three datasets showing that Cascade can perform close to expert level agreement (80-90% of expert performance) for a competitive cost.

RELATED WORK

Crowdsourcing complex tasks

The idea of using small units of human labor in an algorithm was introduced by TurKit [15] and demonstrated on simple workflows such as iteratively improving image descriptions. More recent work has demonstrated more complex crowd workflows. CrowdForge [10] uses a MapReduce-like framework for writing articles by mapping separate workers to different aspects of the article (e.g., the outline, the facts, the quotes, etc.) and then composing the results in a Reduce step. Turk-o-matic [12] asks workers to break down the task and then creates subtasks for more workers to do. Mobi [20] solves problems, like travel planning, that have global constraints met by workers creating to-do items for other workers to resolve. Legion:Scribe [16] breaks down audio transcription tasks into small chunks and uses sequence alignment to combine results in real-time. The challenge in designing a crowd workflow is to break down a task into pieces that can be done independently and then combined. Cascade solves a particularly hard version of this problem: when data must be combined to create a global understanding.

Clustering and Taxonomy Creation

Machine learning algorithms, such as LDA [4], can automatically cluster data. While these automated approaches are fast and cheap (compared with human labor), they have several limitations. First, they require a human to explicitly specify a set of features for each media type, such as the bag-of-words representation for text or computer-vision features for images. Second, their performance is bad; lacking common sense, LDA often creates incoherent clusters. Finally, they have trouble *naming* the resulting categories; incapable of creating an overarching abstraction, they can only identify a common feature (i.e., a word or phrase in the case of text).

Several machine-learning techniques have been refitted to use human inputs. CrowdKernel [18] learns an SVM kernel from human pairwise comparisons. Matrix completion has been used to do clustering with spare pairwise labels [19]. Additional, discriminative labels [17] can be found for image datasets using a mixture of probabilistic principal component analyzers (MPPCA). Crowdclustering [6] uses human-created clusters of a subset of the data and Variational Bayes method to create unlabeled clusters of data and subclusters. These are novel and intriguing ways to combine human and machine effort to complete labeling, and find clusters in data. In contrast, Cascade is an end-to-end system that requires only quick, easy tasks for workers and produces human readable category labels on a taxonomy organizing all the data.

There are several distributed human taxonomy creation efforts. First, Wikipedia maintains a taxonomy of all its pages. The work is coordinated though talk pages, not through a crowd algorithm. Although this is a success in taxonomy creation, it requires a large time commitment overhead and thus it is not easy for individuals to contribute in small increments. Second, Card Sorting [8] is a technique for members of a group to contribute to an organization of their data. Participants create labeled clusters for all data and the similarities between participants' clusters tells the moderator how people mentally cluster the space. Card sorting is an investigative technique. It is not designed to produce a usable categorization. Third, Folksonomies are community-created labeled clusters of data. WordNet -- a database of human-contributed semantic knowledge of links between works -- has been used to turn folksonomies into hierarchies of concepts [13]. It seems that human common sense and linguist ability are currently necessary for taxonomy creation. Cascade uses these human abilities and breaks down the tasks into short, easy units of labor.

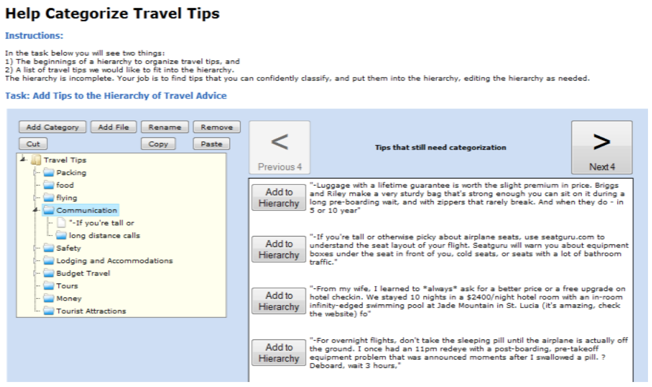
INITIAL APPROACHES

The Cascade algorithm evolved from a sequence of initial prototypes. Our experience resulted in several surprising observations that informed our ultimate design of a crowdsourced taxonomy workflow.

Iterative Improvement

Iterative improvement is a general crowdsourcing pattern first described in TurKit [15]. Iterative improvement has proven successful at using multiple workers to build on and improve each other's image descriptions, and to collectively decipher blurry text or bad handwriting.

CrowdFlower and Karampinas, et al. [9] have applied iterative workflows to clustering. Although it is possible for this method to work, our experiences were largely negative. We applied iterative improvement to taxonomy creation by giving workers a list of items and an editable hierarchy interface. Workers were asked to improve the taxonomy by



Help Categorize Travel Tips

Instructions:

In the task below you will see two things:
1) The beginnings of a hierarchy to organize travel tips, and
2) A list of travel tips we would like to fit into the hierarchy.
The hierarchy is incomplete. Your job is to find tips that you can confidently classify, and put them into the hierarchy, editing the hierarchy as needed.

Task: Add Tips to the Hierarchy of Travel Advice

Tips that still need categorization

"-Luggage with a lifetime guarantee is worth the slight premium in price. Briggs and Riley make a very sturdy bag that's strong enough you can sit on it during a long pre-boarding wait, and with zippers that rarely break. And when they do - in 5 or 10 year"

"-If you're tall or otherwise picky about airplane seats, use seatguru.com to understand the seat layout of your flight. Seatguru will warn you about equipment boxes under the seat in front of you, cold seats, or seats with a lot of bathroom traffic."

"-From my wife, I learned to *always* ask for a better price or a free upgrade on hotel checkin. We stayed 10 nights in a $2400/night hotel room with an in-room infinity-edged swimming pool at Jade Mountain in St. Lucia (it's amazing, check the website) fo"

"-For overnight flights, don't take the sleeping pill until the airplane is actually off the ground. I once had an 11pm redeye with a post-boarding, pre-takeoff equipment problem that was announced moments after I swallowed a pill. ? Deboard, wait 3 hours,"

Figure 2. Early prototype interface: iterative improvement

adding, deleting, or moving categories or by placing items in the taxonomy (Figure 2). We observed that the iterative improvement interface suffered from two main problems:

1. **The taxonomy grows quickly, making the tasks more time consuming and overwhelming as time goes on.** Creating the first category and placing a few items in it is quick and easy. However, when 50 categories must be read in order to figure out whether or not an item belongs in any of the existing categories, the work becomes so challenging that single workers have a hard time making contributions in a short time frame.
2. **The task had many options for how to contribute (add categories, place items, merge categories, etc.) and workers had trouble selecting tasks.** Although giving workers options for how to contribute made the task very flexible, it also meant workers had to decide what was important to do next. It was not always obvious what meaningful work should be done or how long it would take. A similar result was found in the Mobi system [20].

We concluded that we needed to break down the taxonomization task into manageable units of work and have an algorithm coordinate the work.

Category Comparison

Similar to the ESP Game [1], we found that workers are eager and willing to suggest category labels for data. The problem is deciding which labels are useful. The final taxonomy should have a non-redundant set of categories with parent-child relationships. We tried many approaches for directly soliciting these relationships. Figure 3 shows one of our prototypes: a matching game where workers can select pairs of related category labels.

We found that workers were wildly inconsistent in the relationships they saw in the category labels. For example, some thought "air travel" was the same as "flying," while others thought it was a superset. Some thought "packing" was the same as "what to bring", while others thought "what to bring" was a subset of "advice" but "packing" was not.

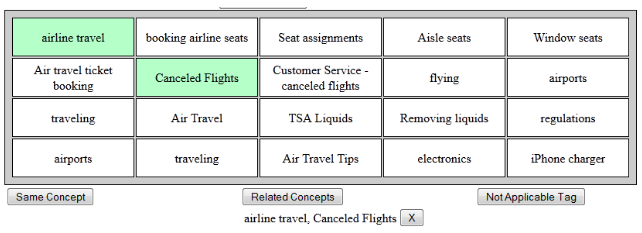


Figure 3. Early prototype interface: category comparison

We concluded that it was a mistake to ask workers to compare abstractions with abstractions. The differing assumptions people make about abstractions are too hard to write down, and they render individual worker judgments incomparable. Instead, judgments should be made relating actual abstractions to data: in this case, relating categories to items.

Clustering

In order to elicit both clusters and cluster names from workers, we prototyped an interface that presented workers with a small number of items (8-10) and asked them to suggest categories that fit at least two items (Figure 4). Although workers found the task easy and intuitive, the quality of the categories was not as good as when we generated category suggestions for single items. Restricting workers to naming categories that satisfied multiple items encouraged workers to name overly broad categories. They often gave categories names such as "good tips" or "advice" because they would fit multiple of the 8 items displayed. However, these are not useful category names. Moreover, what workers wanted to do instead was name categories they felt were good even if they only fit one item. Workers know these categories are good because of their knowledge of the world. For example, workers can use their common sense and intuition about travel advice and infer that "TSA Security" might fit multiple items even though it only fits one item in the 8-item subset.

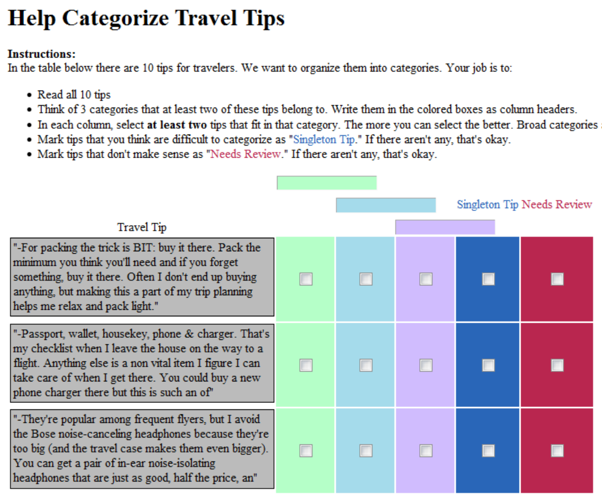
We decided that the clusters workers found in a small subset of the data were often unnatural and forced, and that it was better to allow workers to suggest categories that fit one item very well rather than fit multiple items more loosely.

THE CASCADE ALGORITHM

We now discuss Cascade's inputs and outputs, the parameters governing its execution, its three HIT primitives, and the flow of the algorithm itself.

Inputs

Cascade takes two inputs: a set of *items* to be categorized and a descriptive phrase (the *topic*) identifying these items. An example of inputs to Cascade are 100 responses to the question "What is your best travel advice?" with the topic "Travel Advice."


Help Categorize Travel Tips

Instructions:
In the table below there are 10 tips for travelers. We want to organize them into categories. Your job is to:

- Read all 10 tips
- Think of 3 categories that at least two of these tips belong to. Write them in the colored boxes as column headers.
- In each column, select **at least two** tips that fit in that category. The more you can select the better. Broad categories are good.
- Mark tips that you think are difficult to categorize as "Singleton Tip." If there aren't any, that's okay.
- Mark tips that don't make sense as "Needs Review." If there aren't any, that's okay.

Travel Tip				Singleton Tip	Needs Review
"-For packing the trick is BIT: buy it there. Pack the minimum you think you'll need and if you forget something, buy it there. Often I don't end up buying anything, but making this a part of my trip planning helps me relax and pack light."	☐	☐	☐	☐	☐
"-Passport, wallet, housekey, phone & charger. That's my checklist when I leave the house on the way to a flight. Anything else is a non vital item I figure I can take care of when I get there. You could buy a new phone charger there but this is such an of"	☐	☐	☐	☐	☐
"-They're popular among frequent flyers, but I avoid the Bose noise-canceling headphones because they're too big (and the travel case makes them even bigger). You can get a pair of in-ear noise-isolating headphones that are just as good, half the price, an"	☐	☐	☐	☐	☐

Figure 4. Early prototype interface: item clustering

In theory, Cascade can handle item-sets of arbitrary size, but the algorithm's expense grows super-linearly in the number of items. To date we've tested on five datasets, ranging in size from 22 to 200 items. In the future, we plan to use co-occurrence statistics to optimize the cost on large inputs.

Output

The output of Cascade is a *taxonomy* consisting of labeled *categories* and associated items. More precisely, Cascade generates a tree whose nodes are labeled with a textual string, called a category; the tree's root is labeled with the topic input, and an '*other'* node is added as a child of the root if necessary. Items may appear in multiple categories in the taxonomy; for example, a travel tip about flight deals may appear in "air travel" and "saving money."

Parameters

Cascade's behavior is guided by a set of parameters. Default values which were used in our experiments are noted where applicable.

- Let n be the number of items input.
- Let $m \leq n$ be the number of items considered in Cascade's *initial item-set*, default = 32.
- Let $n' = n-m$ denote the number of items in the *subsequent item-set*.
- Let k be the *replication factor* (the number of workers who may be asked to repeat a step), default = 5.
- Let g be the first voting threshold, default = 2.
- Let h be the final voting threshold, default = 4.
- Let t be the maximum number of items shown to a worker at once, default = 8.
- Let c be the maximum number of categories shown to a worker who is selecting the best category for an item, default = k = 5.
- Let s be the maximum number of categories shown to a worker who is judging relevance of categories to an item, default = 7.
- Let q be the minimum size of a category, default = 2.
- Let p be the percentage of items two categories must have in common in order to be deemed similar, default = 75%.

By changing the values of these parameters, the designer can trade off cost and running time against taxonomy quality. Decreasing k or increasing t, c, and s will lower the cost of execution, presuming workers will still accept the task. Finding the optimal values for these parameters is the subject of future research.

Primitive Worker Tasks

Before describing Cascade's overall control flow, we first present the three types of HITs that are shown to workers. The order in which these tasks are generated depends on the characteristics of the input items and can be complex, but the primitives are individually quite simple. Here we present them abstractly; Figure 5 displays concrete instances of these tasks.

- *Generate:* (t items) → t categories
  The Generate HIT presents a worker with t items and asks her to generate one suggested category for each item. The categories do not need to be distinct, but they often are. She may elect not to name a category for some items.

- *SelectBest:* (1 item, c categories) → 1 category
  The SelectBest HIT shows a worker a single item and c distinct category tags and asks her to pick the single best tag.

- *Categorize:* (1 item, s categories) → bit vector of size s
  The Categorize HIT presents a worker with a single item and s categories and asks the worker to vote whether the item fits each of the categories.

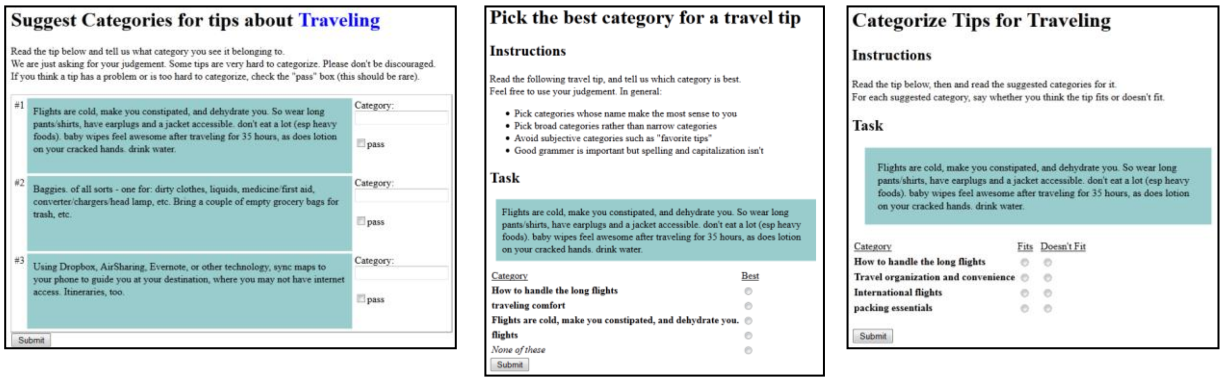
Algorithm Steps

Described at the highest level, the Cascade algorithm takes every item and solicits multiple suggested categories for it from different workers. A new set of workers then votes on the best suggested category for each item. Cascade then asks workers to consider every item with all of these 'best' categories and judge relevance. Cascade next uses this data to eliminate duplicate and empty categories and to nest related categories, creating a hierarchy.

We first present an outline of the steps of Cascade and then describe the steps in full detail.

0) Select m items to be the *initial item-set*. All other items belong to the *subsequent item-set.*

1) Show each of the m items to k workers, who each **generate** a category suggestion for each item.


Suggest Categories for tips about Traveling

Read the tip below and tell us what category you see it belonging to.
We are just asking for your judgement. Some tips are very hard to categorize. Please don't be discouraged.
If you think a tip has a problem or is too hard to categorize, check the "pass" box (this should be rare).

#1	Flights are cold, make you constipated, and dehydrate you. So wear long pants/shirts, have earplugs and a jacket accessible. don't eat a lot (esp heavy foods). baby wipes feel awesome after traveling for 35 hours, as does lotion on your cracked hands. drink water.	Category: ☐ pass
#2	Baggies. of all sorts - one for: dirty clothes, liquids, medicine/first aid, converter/chargers/head lamp, etc. Bring a couple of empty grocery bags for trash, etc.	Category: ☐ pass
#3	Using Dropbox, AirSharing, Evernote, or other technology, sync maps to your phone to guide you at your destination, where you may not have internet access. Itineraries, too.	Category: ☐ pass

Submit

Pick the best category for a travel tip

Instructions

Read the following travel tip, and tell us which category is best.
Feel free to use your judgement. In general:

- Pick categories whose name make the most sense to you
- Pick broad categories rather than narrow categories
- Avoid subjective categories such as "favorite tips"
- Good grammer is important but spelling and capitalization isn't

Task

Flights are cold, make you constipated, and dehydrate you. So wear long pants/shirts, have earplugs and a jacket accessible. don't eat a lot (esp heavy foods). baby wipes feel awesome after traveling for 35 hours, as does lotion on your cracked hands. drink water.

Category	Best
How to handle the long flights	○
traveling comfort	○
Flights are cold, make you constipated, and dehydrate you.	○
flights	○
None of these	○

Submit

Categorize Tips for Traveling

Instructions

Read the tip below, then and read the suggested categories for it.
For each suggested category, say whether you think the tip fits or doesn't fit.

Task

Flights are cold, make you constipated, and dehydrate you. So wear long pants/shirts, have earplugs and a jacket accessible. don't eat a lot (esp heavy foods). baby wipes feel awesome after traveling for 35 hours, as does lotion on your cracked hands. drink water.

Category	Fits	Doesn't Fit
How to handle the long flights	○	○
Travel organization and convenience	○	○
International flights	○	○
packing essentials	○	○

Submit

Figure 5. HIT Primitives: Generate, SelectBest, Categorize

2) Show each of the *m* items and the category suggestions for it. Ask *k* workers to **select the best** category for each item by voting. Filter out suggested categories with insufficient votes.

3) For all items, for all best suggested categories, ask *k* workers to vote whether the category applies to the item or not. This **categorize**s the items.

4) Using the item-category membership information, run a fully-automated process, called *global structure inference,* to produce a taxonomy from the categories.

5) If there are items in the subsequent item-set, ask *k* workers to vote on whether the items fit into the categories already created.

6) Regenerate the taxonomy with the new item-category membership data.

7) If there are items that do not fit into any of the categories during step 5, create a new item-set from the uncategorized items. Repeat the algorithm (from step 1) on these uncategorized items.

The algorithm continues until all the items are categorized. Following are the implementation details for each step.

Step 1. Intentional Category Over-Generation
HIT primitives: *Generate* is called $\lceil m/t \rceil k$ times.
Output: *k* suggested categories for each of the *m* items.

The first step of Cascade is to show each item to *k*=5 people, and have each of them suggest a category for it. We present items in groups of *t*=8. Although multiple items are displayed together, a category suggestion are independent – workers do not have to name categories that apply to multiple items in that HIT. We display multiple items together so that workers get more context about the item-set as a whole. An additional benefit of showing multiple items together is that workers can easily skip items in the HIT. If a worker is unable to think of a category, it is better for her to skip it than to suggest something awkward, forced, or overly-specific. We allowed workers to skip at most 2 of every 8 items.

Step 2. Best Category Suggestion Vote
HIT primitives: *SelectBest* is called *mk* times.
Output: a set *C* of best suggested categories (of size ~1.5*m*)

In Step 1, *k*=5 individual workers attempted to categorize each item, resulting in up to five *suggested categories* for each item. If an item was skipped in Step1, there will be fewer than *k*=5 suggested categories for it. If any of the suggested categories are exact-string duplicates, we remove the duplicate, also leaving us with fewer than *k*=5 suggested categories. In this step, we show each worker one item and *c*<=*k* of its remaining suggested categories. Workers vote for the one they think is best or select "None." Any suggested category that gets two or more votes is passed on to the next step. We call these the *best suggested categories*, and refer to the set as *C*.

Step 3. Adaptive Context Filtering
HIT primitives: *Categorize* is called $m\lceil |C|/s \rceil k + mk$ times
Output: item-category membership decisions

Phase 1: We take the best suggested categories and partition them into groups of *s*=7 categories. In a *Categorize* HIT, we present each item with each category group to *k*=5 workers. Workers vote on whether each item fits each category or not. Any category that gets at least *g* = 2 out of five votes goes to Phase 2.

Phase 2: In a *Categorize* HIT, we present each item with all the categories that had at least *g* = 2 votes for fitting in that item. There will hopefully be fewer than *s*=7 categories. If there are more, we break the categories into group of *s*=7. We get *k*=5 workers to vote, and any category that gets at least *h*=4 out of five votes officially fits in that category.

We call this 2-phase process Adaptive Context Filtering. We do Adaptive Context Filter instead of straight categorization because we observed that workers vote differently depending on the group of categories we present to them. We observed that workers are often tempted to select at least one category that fits an item, even if none of them fit very well. Thus, we treat the categories found in Phase 1 as *potentially applicable categories*. In Phase 2, we vote on all the potentially applicable categories for an item together, as a group. When all the categories are at least potentially applicable to the item being shown, the context for selecting the categories is better and workers make better decisions.

Step 4. Global Structure Inference
HIT primitives: none
Output: a taxonomy with *m* items

At this stage, we have set of category labels generated by workers. We filtered out some of the category labels, and then we categorized all *m* items into all the remaining category labels. Global structure inference uses this category membership data to organize the categories into a cohesive taxonomy. There are three steps:

1. *Remove insignificant categories*. Remove any category that has fewer than q=2 items.
2. *Remove duplicate categories*. For any two categories that share more than p =75% of their items, we remove the category with fewer items, breaking ties by random choice.
3. *Create nested categories*. For any category c_{sm} that shares more than p=75% of its items with another category c_{lg}, make c_{sm} a subcategory of c_{lg}.

The result of global structure inference is a taxonomy where all categories have at least two items, sibling categories represent distinct concepts, and subset categories are properly nested under their super category.

Step 5. Categorize Subsequent Item-Set
HIT primitives: Categorize is called $(n\text{-}m)\lceil |C|/s \rceil k$ times
Output: updated item-category membership decisions

Categorize the subsequent item-set on the existing categories.

Step 6. Update Taxonomy
HIT primitives: none
Output: a taxonomy with *n* items

Rerun Global Structure Inference on the item-category membership data from Step 5.

When to Recurse
The taxonomy is complete if it sufficiently explains all the items. There are two conditions for items that are not sufficiently explained:

1) Items completely uncategorized
2) Items only in categories with 20 or more items

If either of these conditions are met, we create a new item-set of the insufficiently explained items and rerun the algorithm from Step 1. As the algorithm reruns and generates new categories for these items, we apply the original items to the new categories as well (as in Step 5), and rerun global structure inference. This produces a taxonomy with new categories that explain the previously unexplained data.

	Running time	HITs	Cost
Step 1 : Intentional Category Over-Generation	$O(\lceil m/t \rceil k)$	20	\$3.20
Step 2: Best Category Suggestion Vote	$O(mk)$	160	\$3.20
Step 3. Adaptive Context Filtering – Phase 1	$O(m\lceil |C|/s \rceil k)$	1100	\$22
Step 3. Adaptive Context Filtering – Phase 2	$O(mk)$	160	\$3.20
Step 5. Categorize Subsequent Item-Set	$O((n\text{-}m) \lceil |C|/s \rceil k)$	320	\$6.40
Iteration 1 total		1760	\$38
Iteration 2 total		1760	\$38
Total:		3520	\$76

Table 1. Asymptotic running time of Cascade with n=64, m=32 and other values at their stated default values.

It is obvious why we would want to rerun Cascade while there are items that are still uncategorized. However, it is less obvious that we need to rerun Cascade while there are items that are categorized, but only in large categories. We call items only in large categories "loosely categorized items." If a worker suggested a category that has 50% of the items in it, we want to make sure we find subcategories within that category to help users browse the items in it.

The total cost and running time of Cascade is dependent on worker output and how many iterations are needed. An example of the cost and running time for a dataset with two iterations (n=64 and m =32) is presented in Table 1.

EXPERIMENTS

To test the performance of Cascade we run the algorithm on three datasets obtained from Quora.com and present the taxonomies it produces.

Datasets
Quora.com is an online question-and-answer site. Often the questions are open-ended, such as "What are your best travel hacks?" Hundreds of people respond with valid tips and opinions. The responses are all valid, and this is the type of domain where a taxonomy would help users get a global picture of the data and navigate the responses. We picked 3 open-ended questions posted on Quora.com and created a taxonomy of items in the replies. The three datasets are summarized in Table 2.

Often, a single response will contain multiple tips in bulleted lists, numbered lists, or paragraphs. We manually broke these responses into separate items. Previously, we

Abbreviation	**Topic**	**# items**
editWriting	"What are some tips for editing your own writing?"	22
sideProjects	"How can I increase my productivity on my side projects at the end of the day when I'm tired from work?"	67
travel	"What are your best travel hacks?"	100

Table 2. Topics and size of item-set

have had the crowd do this breakdown. It is a straightforward step but not a part of the Cascade algorithm.

We randomized the order of the items to avoid any effects of our workers seeing items in the order they were generated.

Implementation

We implemented the primitive HITs in HTML and JavaScript, which served as ExternalQuestions on Mechanical Turk (MTurk). To dispatch HITs, we used TurKit [15]. Python scripts were used to process data between steps.

RESULTS

editWriting was the smallest item-set with 22 items. In one iteration, Cascade produced a taxonomy with 15 categories, 8 of which were top-level (Figure 6). Global structure inference correctly eliminated three redundant categories and eliminated insignificant singleton categories, such as "reformat" and "write, delete, rewrite."

sideProjects is a mid-sized domain with 67 items. In the first iteration, 32 items were used to generate a taxonomy with 22 categories. Subsequent processing generated no new categories, and global structure inference created the parent-child relationships as shown in Figure 6.

travel was the largest item-set with 100 items. In the first iteration, 32 items were used to generate a taxonomy with only 7 categories. That taxonomy left 66 of the 68 items in the subsequent item-set insufficiently categorized. The next iteration of Cascade yielded a taxonomy with 51 items, fitting all the items (Figure 6).

EVALUATION

The goal of Cascade is to produce a taxonomy that provides a global understanding of the items. To evaluate that goal, we ask and answer three questions:

1. Are the category labels in the taxonomy as good as labels created by experts?
2. Does the taxonomy have an appropriate hierarchical structure?
3. Is the cost and running time of Cascade competitive with hiring experts?

Good Category Labels

Taxonomies are inherently subjective. Experts often disagree on the categories and level of granularity of the taxonomy. However, given a small pool of experts independently categorizing a dataset, one would expect some of the same categories to appear in multiple experts' taxonomies. In order to compare Cascade's categories to those of experts, we paid four information architects to produce taxonomies for our three datasets.

We performed the following comparison on the taxonomies. For each data set, we took the Cascade-produced taxonomy and the four expert taxonomies. We compared two things:

1. What fraction of the Cascade taxonomy categories are also named in at least one expert taxonomy?
2. What fraction of expert categories are named in another expert taxonomy?

Table 3 contains the results of this comparison. For all three datasets, about 50% of Cascade's categories were also named by an expert. For example, in the editWriting dataset, four out of four experts named a category closely matching Cascade's category "working off an outline" (two experts named it "outlining.") The authors of this paper made the similarity judgments.

	edit-Writing	side-Projects	travel	Avg.
% of Cascade categories shared by at least one expert	47%	50%	53%	**50%**
avg % of expert categories shared by at least one other expert	32%	70%	64%	**55%**

Table 3. Category name quality comparison

When comparing experts to each other, the average expert matching fraction was 32%, 70%, and 64% for the three datasets. This averages to 55% of categories matching another expert's categories across these three hierarchies, compared to the 50% agreement between Cascade and the experts. Comparing these, we note that Cascade had 91% of the category agreement the experts did among themselves.

Mistakes in Hierarchical Structure

Cascade infers a global understanding of the data from the item membership of categories. Cascade removes categories that do not have enough items in them, removes categories that have a high item overlap, and creates a parent-child relationship for categories where one category has high item overlap with the other. These inferences are based on many small judgments by potentially hundreds of different people. We want to know if all those judgments come together to form a coherent hierarchy. In particular, we are looking for three types of mistakes in the Cascade hierarchies:

1. Duplicate categories
2. Missing Parent-Child Relationships
3. Incorrect Parent-Child Relationships

To find the error rate in the hierarchical structure, we divide the number of errors by the number of categories in the taxonomy. The authors of the paper were the judged the errors. editWriting has the smallest error rate of 13% (Table 4), with only 2 errors in 15 categories. Both were duplicate-categories errors. The categories "tips to edit better" and "how to edit better" should have been the same, but Cascade left them both in the taxonomy.

	Edit-Writing	Side Projects	Travel: iteration1	Travel: iteration2
# categories	15	18	7	51
Duplicate Categories	2	2	0	2
Missing Nesting	0	0	0	5
incorrect Nesting	0	3	1	3
Correct Nesting	5	3	1	23
total errors	2	5	1	10
Error rate	**13%**	**27%**	**14%**	**20%**

Table 4. Errors in hierarchical structure

sideProjects had the highest error rate of 27%. This came from 3 incorrect parent-child relationships: 'prioritizing' was the parent of 'commitment,' 'prioritizing' was the also parent of 'consistency,' and 'motivation' was the parent of 'relaxation.' In our judgment, there is no clear reason that prioritizing should be a parent of commitment or consistency, or that motivation should be the parent of relaxation, and thus it is a mistake in the hierarchical structure of the taxonomy. These are errors produced by the automated global structure inference step. It nested 'commitment' under 'prioritizing' because more than 75% of the tips about commitment were also about prioritizing. Although these categories share many tips in common, they aren't semantically related: this is a danger of machine steps. Perhaps a solution would be to have humans check the resulting taxonomy for obvious errors.

Across the three datasets, the average error rate was 18.5%.

There were an impressive number of correct parent-child relationships, especially in the travel dataset, with 23 correct parent child relationship and 3 incorrect ones. Many air-travel and flight-related categories with complicated nesting are expressed with coherent hierarchical structure. For example, "air travel tips" is a parent of "flights," which is a parent of "flight layovers."

Time and Money

It is non-trivial to compare the costs associated with creating a taxonomy with Cascade versus experts. There is a cost-quality-time trade-off. For example, on MTurk, under-priced HITs will eventually get done, but will take a long time. The most basic comparison we provide is the actual costs and times in our run of Cascade and that of our recruited experts (Table 5). Cascade took ~6.5 times longer to complete the HITs, and was 1-3 times as expensive. However, the prices were set fairly arbitrarily. We paid our experts \$25/hour as a set wage. We paid MTurk workers \$0.05 per HIT. The average time to complete a HIT was 21.46 seconds. This equates to \$8.39/hour, which is high for MTurk, where \$3-\$4/hour is more typical. Running the HITs at this standard marketplace rate would reduce the cost of Cascade by a factor of 2, making Cascade's cost competitive with the wage we offered experts.

	editWriting	sideProjects	travel
Cascade Time	7 h 56 m	16h 13 m	16h 32m
Avg expert time	1h 23 m	2h 36m	2h 5 m
Cascade Cost	\$35.40	\$109.45	\$224.45
Average Expert Cost	\$34.87	\$65.13	\$71.38

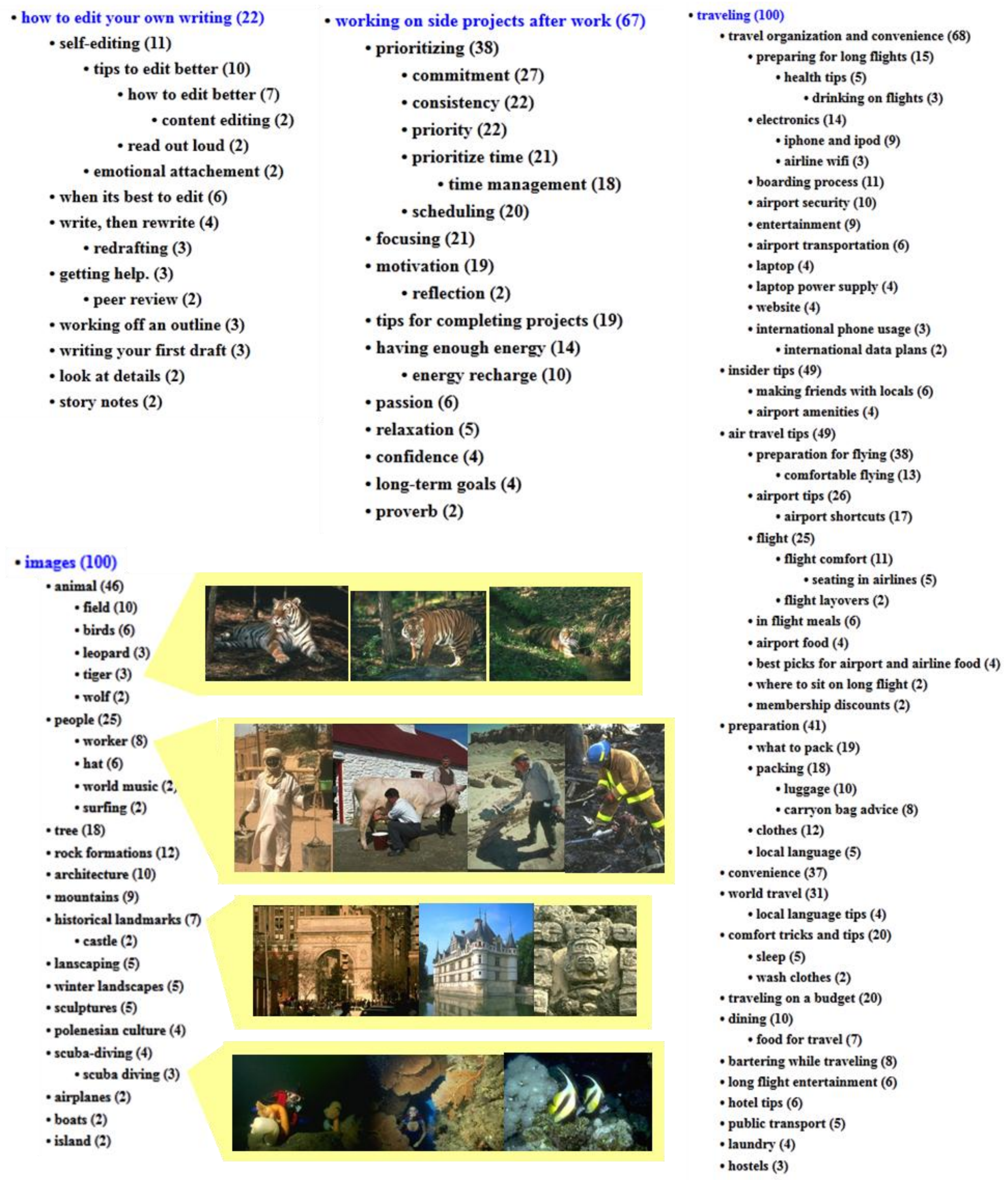
Table 5 Time and cost comparison

The total time spent on all three datasets by the average expert was 6 hours 50 minutes, and the total time spent by MTurk workers was 43 hours 3 minutes. This is a factor of 6.3 more time spent by MTurk workers. Seeing as the work done by workers is basically replicated k=5 times over, the time it would take for a single person to run Cascade on himself (k=1) would be competitive with the expert's time.

More important than comparing total time spent on the algorithm is to think about the minimum amount of time that it would take to run the algorithm if sufficiently many people work in parallel, as is supported by Cascade. Each worker spends on average 21.3 seconds per HIT, and all the HITs in any step can be run in parallel. Thus, assuming Cascade is run in two iterations of 5 steps each, and the maximum time a worker spend on a task was 30 seconds, the entire time it would take to run Cascade would be five minutes.

DISCUSSION

The Cascade algorithm is built on three simple HIT primitives: Generate, SelectBest and Categorize. These primitives are seen in other crowd workflows: Generate is very similar to what the ESP Game [1] and VizWiz [3] ask for. SelectBest is similar to the voting steps in the iterative improvement workflow used by TurKit [14] and Soylent



- how to edit your own writing (22)
  - self-editing (11)
    - tips to edit better (10)
      - how to edit better (7)
        - content editing (2)
      - read out loud (2)
    - emotional attachement (2)
  - when its best to edit (6)
  - write, then rewrite (4)
    - redrafting (3)
  - getting help. (3)
    - peer review (2)
  - working off an outline (3)
  - writing your first draft (3)
  - look at details (2)
  - story notes (2)

- working on side projects after work (67)
  - prioritizing (38)
    - commitment (27)
    - consistency (22)
    - priority (22)
    - prioritize time (21)
      - time management (18)
    - scheduling (20)
  - focusing (21)
  - motivation (19)
    - reflection (2)
  - tips for completing projects (19)
  - having enough energy (14)
    - energy recharge (10)
  - passion (6)
  - relaxation (5)
  - confidence (4)
  - long-term goals (4)
  - proverb (2)

- traveling (100)
  - travel organization and convenience (68)
    - preparing for long flights (15)
      - health tips (5)
        - drinking on flights (3)
    - electronics (14)
      - iphone and ipod (9)
      - airline wifi (3)
    - boarding process (11)
    - airport security (10)
    - entertainment (9)
    - airport transportation (6)
    - laptop (4)
    - laptop power supply (4)
    - website (4)
    - international phone usage (3)
      - international data plans (2)
  - insider tips (49)
    - making friends with locals (6)
    - airport amenities (4)
  - air travel tips (49)
    - preparation for flying (38)
      - comfortable flying (13)
    - airport tips (26)
      - airport shortcuts (17)
    - flight (25)
      - flight comfort (11)
        - seating in airlines (5)
      - flight layovers (2)
    - in flight meals (6)
    - airport food (4)
    - best picks for airport and airline food (4)
    - where to sit on long flight (2)
    - membership discounts (2)
  - preparation (41)
    - what to pack (19)
    - packing (18)
      - luggage (10)
      - carryon bag advice (8)
    - clothes (12)
    - local language (5)
  - convenience (37)
  - world travel (31)
    - local language tips (4)
  - comfort tricks and tips (20)
    - sleep (5)
    - wash clothes (2)
  - traveling on a budget (20)
  - dining (10)
    - food for travel (7)
  - bartering while traveling (8)
  - long flight entertainment (6)
  - hotel tips (6)
  - public transport (5)
  - laundry (4)
  - hostels (3)

- images (100)
  - animal (46)
    - field (10)
    - birds (6)
    - leopard (3)
    - tiger (3)
    - wolf (2)
  - people (25)
    - worker (8)
    - hat (6)
    - world music (2)
    - surfing (2)
  - tree (18)
  - rock formations (12)
  - architecture (10)
  - mountains (9)
  - historical landmarks (7)
    - castle (2)
  - lanscaping (5)
  - winter landscapes (5)
  - sculptures (5)
  - polenesian culture (4)
  - scuba-diving (4)
    - scuba diving (3)
  - airplanes (2)
  - boats (2)
  - island (2)

Figure 6 Taxonomies created by Cascade

[2], and Categorize is explored in Polarity [13] and is a common task on MTurk – it was the original reason Amazon created MTurk. Because Cascade uses common HIT types, workers do not have to spend the overhead of reading special instructions or learning a complex task.

Cascade is essentially bottom-up. We solicit many categories based on single items, we filter out bad categories and then use global structure inference to create a cohesive global picture out of individual Categorize HITs. Global structure inference is the reason Cascade works. It is what combines small, independent contributions into a taxonomy and decides which items we need to rerun Cascade on, thus focusing work where it is needed most.

Cascade allows items to go in multiple categories. This is what allows Cascade to perform global structure inference. For example, we know that "LAX security lines" is a child of "air travel" because all the tips in "LAX security lines" are also children of "air travel." Allowing tips to be in multiple categories, makes . The tip "use kayak.com" is both about "saving money" and "air travel." We see no reason to make workers pick which category it fits better.

In Cascade, we compare data to abstractions, namely an item to a category. We do not compare items to items, or categories to categories. We observed that workers were more comfortable with item-to-category comparisons. Comparing items to each other involves an assumption about the aspect of the item the worker is comparing. The item "use kayak.com" is similar to the item "free wiki at LAX" because it has to do with saving money and "I hate Travelocity!" because it involve a website. It's hard to say which it is more similar to. Comparing categories to categories involves assuming a grounding for the abstraction. (Is the category "air travel" the same as "flights"?) However, "free wiki at LAX" is clearly about "air travel."

FUTURE WORK

The biggest area of improvement for Cascade is its cost in dollars. There are several ways to address this. The first way would be to optimize the categorization step. We could adopt machine learning approaches, we could optimize parameter values of Cascade using decision theory [5] - currently Cascade uses 5-fold redundancy, but some item/category pairs likely need fewer eyes. We envision an adaptive approach. We also plan to apply Cascade to manage volunteers in community-sourcing [6, 10]. In this context the work would be free and done by a worker with domain knowledge.

CONCLUSION

This paper presents a crowd-algorithm that generates a taxonomy over a set of independent data items, such as travel items, color blocks (Figure 1), or images (Figure 6). We show that using three *HIT primitives* -- Generate, SelectBest, and Categorize -- we can power an algorithm where workers do as little as 20 seconds of work. A crucial step in the algorithm is to use *global structure inference* to combine small, independent units of work into the final taxonomy. Compared to expert information architects, the taxonomies Cascade produced were competitive in quality and price. Since Cascade is parallelizable and uses small units of work it can make use of large crowds of people and complete in minutes rather than hours or days.

ACKNOWLEDGEMENTS

Big thanks to Haoqi Zhang, Rob Miller, David Molnar, Jonathan Bragg. This work was supported by the WRF / TJ Cable Professorship, Office of Naval Research grant N00014-12-1-0211, and National Science Foundation grant IIS 1016713.

REFERENCES

1. von Ahn, L., Dabbish, L. Labeling Images with a Computer Game. *CHI 2004*
2. Bernstein, M., et al.. Soylent: A Word Processor with a Crowd Inside.*UIST 2010.*
3. Bigham, J. P., et al.. VizWiz: Nearly Real-time Answers to Visual Questions. *UIST 2010.*
4. Blei, D. M., Ng, A. Y., Jordan, M. I. Latent dirichlet allocation. *The Journal of Machine Learning Research. Volume 3*, 3/1/2003. Pages 993-1022.
5. Dai, P., Mausam, Weld, D.S. Decision-Theoretic Control of Crowd-Sourced Workflows. *AAAI 2010*.
6. Gomes, R., Welinder, P., Krause, A., Perona, P., "Crowdclustering", *NIPS 2011*.
7. Heimerl, K., Gawalt B., Chen, K., Parikh, T., Hartmann, B.. Communitysourcing: Engaging Local Crowds to Perform Expert Work Via Physical Kiosks. *CHI 2012*.
8. Hudson, W. (2012): Card Sorting. In: *Soegaard, Mads and Dam, Rikke Friis (eds.). "Encyclopedia of Human-Computer Interaction". Aarhus, Denmark.*
9. Karampinas,D. Triantafillou. P. Crowdsourcing Taxonomies, 12th Extended Semantic Web Conference. *ESWC 2012.*
10. Kittur, A., Smus, B., Khamkar, S., Kraut, R. E. CrowdForge: crowdsourcing complex work. *UIST 2011*.
11. Kriplean, T., Morgan, J. T., Freelon, D., Borning, A., Bennett, L. Supporting Reflective Public Thought with ConsiderIt. *CSCW 2012.*
12. Kulkarni, A., Can, M., Hartmann, B. Collaboratively crowdsourcing workflows with turkomatic. *CSCW 2012.*
13. Laniado, D., Eynard, D., Colombetti, M. Using WordNet to turn a folksonomy into a hierarchy of concepts. *SWAP 2007.*
14. Law, E., Settles, B., & Snook, A.. Human computation for attribute and attribute value acquisition. *CVPR Workshop on Fine-Grained Visual Categorization 2011.*
15. Little, G., Chilton, L. B., Goldman, M., Miller, R. C. TurKit: human computation algorithms on mechanical turk. *UIST 2010*.
16. Lasecki, W. S. et al . Real-Time Captioning by Groups of Non-Experts. *UIST 2012*.
17. Parikh, D., & Grauman, K. Interactively building a discriminative vocabulary of nameable attributes. *CVPR 2011.*
18. Tamuz, O., Liu, C., Belongie, S., Shamir, O., and Kalai, A.T. Adaptively learning the Crowd Kernel. *ICML 2011.*
19. Yi, J., Rong, J., Jain, A., and Jain, S. Crowdclustering with Sparse Pairwise Labels: A Matrix Completion Approach. *HCOMP 2012.*
20. Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., Horvitz, E. Human computation tasks with global constraints. *CHI 2012*