# Human-Powered Top-k Lists

Vassilis Polychronopoulos
UC Santa Cruz
vassilis@soe.ucsc.edu

Luca de Alfaro
UC Santa Cruz
luca@soe.ucsc.edu

James Davis
UC Santa Cruz
davis@cs.ucsc.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Neoklis Polyzotis
UC Santa Cruz
alkis@cs.ucsc.edu

## ABSTRACT

We propose an algorithm that obtains the top-k list of items out of a larger itemset, using human workers (e.g., through crowdsourcing) to perform comparisons among items. An example application is finding the best photographs in a large collection by asking humans to evaluate different photos. Our algorithm has to address several challenges: obtaining worker input has high latency; workers may disagree on their judgments for the same items; some workers may provide wrong input on purpose; and, there is a varying difficulty in comparing different items. We provide experimental evidence for the good performance of the algorithm, using extensive simulations and actual experiments with workers from Amazon's Mechanical Turk.

## Keywords

median rank aggregation,tournaments, crowdsourcing

## 1. INTRODUCTION

Crowdsourcing, i.e., harnessing human computation from a large crowd of workers, has gained popularity as a method to tackle simple tasks that machine computation cannot solve, e.g., attaching keywords to an image or inferring sentiment from a piece of natural-language text. In turn, recent studies have proposed algorithms that combine crowdsourcing with automated computation in order to solve more complex problems. For instance, the algorithms in [6] can categorize an item within an ontology by asking the crowd to compare the item against a few carefully-selected concepts in the ontology.

In this paper, we introduce an algorithm to obtain the top-$k$ items out of a larger itemset by using the crowd to evaluate the "goodness" of items. An example instance of this problem is selecting a handful of the most "appealing" photographs out of a larger set. Another example is selecting the few most qualified candidates for a specific job out of a pool of applicants. In both of these cases, the quality of an item (a photograph or a job candidate) cannot be computed by an automated method. Instead, our algorithm assigns to the crowd the task of ranking a *small* number of such items. By carefully selecting these ranking tasks and combining their results, the algorithm generates the $k$ best items, as judged by the crowd. An underlying assumption here is that the crowdsourcing service provides access to workers who can perform such rankings. Hence, we may use Amazon's Mechanical Turk to rank photographs, but we will rely on a specialized service of experts to rank job applicants.

Our algorithm has to address several challenges that stem from the usage of crowdsourcing. Recruiting human workers and obtaining their answers incurs a high latency and may involve a monetary expense. In most crowdsourcing settings, there is a three-way tradeoff among quality of results, latency and monetary cost; optimization of one quantity comes in the expense of the two others. Some crowdsourcing applications are expected to return results at interactive speed. There are analytical methods for real-time crowdsourcing [2] that optimize latency. The problem that we consider may involve large itemsets and may require a large number of comparison tasks, therefore, in many cases we cannot realistically expect the results to be obtained in high speed. Our technique focuses on optimizing monetary cost and makes several calls to the service before obtaining the final results. It is thus not suitable for real-time applications, however, it keeps latency reasonably low by limiting the number of calls to the crowdsourcing service. The algorithm cannot simply employ large ranking tasks, as there is an inherent constraint on the size of a task. For instance, a worker may be able to rank up to a maximum of ten photographs at a time. Another challenge stems from the different judgements made by human workers on the same ranking task, which in turn introduces uncertainty in the ranking of items. The algorithm can aggregate answers for the same ranking task from several workers in order to reduce uncertainty, but this requires more calls to the crowdsourcing service and hence increases latency and expense. Uncertainty also depends on the difficulty of ranking tasks, e.g., ranking photographs may be more straightforward than comparing applicant resumes. Moreover, workers may be more prone to errors depending on the crowdsourcing service. For instance, it is well known that Amazon Mechanical Turk has a considerable population of (hard to detect) spammer workers who intentionally provide false answers to tasks.

Previous studies have addressed specific variants of the top-$k$ problem. In [8], Venetis et al. explore algorithms for obtaining the maximum (i.e., top-1) item assuming some a-priori knowledge about the errors in the answers obtained from the crowdsourcing service. In contrast, our algorithm solves the generalized top-$k$ problem without requiring this knowledge. A recent study [3] explored the more general top-$k$ problem assuming that workers rank two items at a time. Our algorithm does not have this restriction and can issue ranking tasks of several items each. Depending on the underlying domain, this strategy can reduce greatly the number

of calls to the crowdsourcing service without compromising the accuracy of the final top-$k$ items. Finally, several studies[5, 1] deal with the problem of fully sorting a set of items using human workers to perform comparisons. The proposed algorithms can be used to solve the top-$k$ problem by first obtaining the full sorted order and then returning the first $k$ items. However, this approach wastes money and time to perform uninteresting comparisons beyond the first $k$ items. This waste can be significant if $k$ is much smaller than the total number of items, which is common in practice.

The main idea of our algorithm is to set up a tournament to gradually reduce the input itemset down to the top-$k$ items. Within each round of the tournament, the algorithm compares items by issuing several ranking tasks to the crowdsourcing service. The number of workers assigned to each task is chosen adaptively, based on the perceived complexity of computing a ranking. Detecting this complexity in a robust and adaptive fashion is one of the key technical challenges that we address in our work. More concretely, the main contributions of our work can be summarized as follows:

- We define an algorithm to solve the top-$k$ problem that employs human workers to compare several items at a time (Section 3.1). The algorithm is parameterized by a component to aggregate the answers of several workers on the same ranking task, and a component to adaptively select the number of workers for the ranking of a specific set of items.
- We introduce instantiations for the aforementioned components (Sections 3.2 and 3.3) that work adaptively and do not require a-priori knowledge about the errors committed by human workers or the existence of spammer workers. The resulting top-$k$ algorithm can thus be deployed in different problem domains with minimal tuning.
- We present an experimental study of the algorithm using both simulations of human crowds and actual workers from Amazon Mechanical Turk. The results demonstrate that the algorithm computes accurate results even if ranking becomes difficult or there are many spammer workers (up to 40% of the worker population). Moreover, the algorithm matches the performance of a specialized competitor for the top-1 problem, and offers significant savings compared to the state-of-the-art method that sorts the entire itemset.

## 2. PROBLEM STATEMENT

**Preliminaries.** We consider a set of items $O$ with cardinality $n$. A *ranking* of the itemset is a permutation of the $n$ items, where the first elements of the permutation are the highest ranked items and the last elements of the permutation are the lowest ranked items. Let $\sigma$ and $\tau$ denote two rankings. By $\sigma(i)$ we denote the *rank* of an item $i$ in ranking $\sigma$, that is, its position in the ranking. For the highest ranked element $t$ in $\sigma$, $\sigma(t) = 1$ and likewise for $\tau$. For two items $i$ and $j$ if $\sigma(i) < \sigma(j)$ we say that $i$ is *ahead of* or *better than* $j$ in ranking $\sigma$. Given an integer $k$, an item $i$ is a top-$k$ item in $\sigma$ iff $\sigma(i) \leq k$. A top-$k$ list is the set of all top-$k$ items. Note that, by the definition, a top-$k$ list is a set and not a ranking. That is, it does not capture the rank of the items within the list.

We assume the existence of a crowdsourcing service that allows a requester (in this case, our algorithm) to post tasks that can be completed by human workers. We consider a specific type of task defined as follows: Given a set of $s$ items, ask a worker for a ranking of the $s$ items according to the worker's perception. We term this type of task a $s$-ranking task. Of course, the answers of different workers may differ for the same task, either due to differences in perception of item "goodness" or simply because workers may act as spammers who return random answers just to collect the reward. For this reason, we cannot assume any collective properties on the answers from different workers, e.g., that the returned rankings will define a total order. Note that the result of a task is the ranking of the $s$ items and not just the set of top-$k$ items. As we discuss later, this requirement is crucial in order to ensure the robust aggregation of answers from different workers. We assume that each task is issued in a crowdsourcing service as a single Human Intelligence Task (HIT).

We associate two cost metrics with using a crowdsourcing service, namely latency and expense. We measure latency as the number of roundtrips to the crowdsourcing service, where a roundtrip involves the parallel issuing of several tasks and subsequently the collection of answers from the workers. We assume that there are enough workers to work the posted tasks in parallel, and hence it is desirable to issue many tasks in each roundtrip. Each roundtrip may be in the order of several hours, and so it is also desirable to reduce the total number of roundtrips. The expense metric involves the reward paid to workers for the completed tasks. We adopt the common practice of paying the same amount for each task, and hence we measure expense in terms of the total number of posted tasks, across all roundtrips.

**The top-$k$ problem.** We assume the existence of a ground-truth ranking $\beta$ that sorts the items in $O$ according to some property of interest, as judged by the workers of the specific crowdsourcing service. For instance, $\beta$ represents the ranking of photographs based on their appeal or the ranking of job candidates based on their qualifications. Ranking $\beta$ is unattainable for practical purposes, as it would require asking and aggregating the preference of all workers for all items in $O$. It is not used by our algorithms; we only use it to reason about correctness. In our evaluation section, we evaluate accuracy of results by comparing against a known "gold standard" ranking $\beta$.

Our goal is to compute $k$ items that are close to the top-$k$ items in the ideal ranking $\beta$, by asking a limited number of the aforementioned ranking tasks. Hence, the problem that we solve can be defined informally as follows: *Given an itemset $O$ and a positive integer $Q$, compute a subset $\Omega$ of $O$ such that $\Omega$ contains $k$ items, the computation of $\Omega$ requires at most $Q$ ranking tasks, and the rank $\beta(i)$ of each item $i$ in $\Omega$ is close to the interval $[1, k]$.* We make the intuitive assumption that an algorithm can infer the relative position of items in $\beta$ by issuing a sufficient number of ranking tasks. Specifically, if several workers answer the same ranking task and agree that $i$ is before $j$ in their answers, then most likely $\beta(i) < \beta(j)$. Deciding when there is enough agreement among workers to infer this relationship is a key technical challenge that we have to solve. We assume that $k < s$, that is, $k$ is smaller than the largest possible ranking task that can be executed by a human worker.

It would be possible to produce a tighter problem definition (and one that is more amenable to analysis) by making specific assumptions about the distribution of errors in the answers of human workers. However, as noted in a previous study [9], this distribution depends heavily on the similarity of items in the unknown ranking $\beta$ and is hence very difficult to obtain in practice.

## 3. TOP-$K$ ALGORITHM

We now introduce our algorithm for the top-$k$ problem defined previously. The algorithm works in iterations, where in each iteration it issues several $s$-ranking tasks, processes the answers provided by workers and then prunes away items that are not likely to be among the top-$k$.

In what follows, we introduce the algorithm and then describe

```
Input  : Itemset O, integer k, integer s
Output: k items in O
Data: candidate_set: an itemset, partition_set: a set of itemsets
1  candidate_set ← O;
2  while |candidate_set| > k do
3      partitions ← Partition(candidate_set, s) ;
4      Mark each itemset p in partitions as incomplete ;
5      candidate_set ← ∅;
6      while partitions contain incomplete subsets do
7          foreach incomplete subset p in partitions do
8              Post s-ranking tasks for p  // Section 3.3;
9              R ← answers of the s-ranking tasks for p ;
10             r ← top-k items for p as aggregated from R //
               Section 3.2;
11             if p is completed then
12                 candidate_set ← candidate_set ∪ r
13 return candidate_set;
```

Figure 1: Tournament algorithm for the top-$k$ problem.

two of its key components: how to aggregate the answers of different workers, and how to determine the number of workers assigned to each task.

## 3.1 Algorithm Definition

Figure 1 shows the pseudocode of the algorithm. The algorithm receives as input the itemset $O$, the target number $k$ for the top items, and the size $s$ of the ranking tasks $(s > k)$ , and outputs $k$ items from $O$ that are deemed to be the top-$k$ items in $O$ based on the results of ranking tasks.

The algorithm maintains a variable *candidate_set* that comprises items from $O$ that are candidates for the top-$k$ output. Initially, this candidate set is the entire itemset $O$. The algorithm then proceeds in iterations, where in each iteration some items in *candidate_set* are pruned based on the results of $s$-ranking tasks. Specifically, each iteration initially partitions (in a random fashion) the items in *candidate_set* in disjoint subsets of size $s$. Each subset corresponds to a group of $s$-ranking tasks that will be posted to the crowdsourcing service[1]. Initially, each subset $p$ is marked as incomplete to denote that the algorithm has not yet computed the top-$k$ items in $p$. Subsequently, the algorithm posts $s$-ranking tasks for each incomplete subset $p$. The number of such tasks is determined based on the difficulty of comparing items in $p$. We discuss one method for this in Section 3.3. Note that the posting of $s$-ranking tasks for the incomplete subsets occurs in parallel, i.e., the algorithm does not wait for the answers for one subset before posting the tasks for another.

Once all the tasks have been posted, the algorithm gathers answers for each subset $p$ and aggregates them to determine the top-$k$ items in $p$. We discuss one possible aggregation method in Section 3.2. We use the aggregation results to measure consensus among workers; we think of consensus as a predictor of correctness and in the case of high consensus we consider the results correct and mark $p$ as completed. The top-$k$ items of completed subsets are inserted in the candidate set for the next iteration. The determination of completeness is closely coupled with the method to determine the number of tasks for each subset $p$ and we discuss this further in Section 3.3.

The iteration continues until all subsets have been marked as completed. At that point, the candidate set of the next iteration comprises the top-$k$ items from each subset $p$, and hence the size of the candidate set has been reduced by a factor of $s/k$. When the number of candidate items is not greater than $s$, the algorithm issues one final set of $s$-ranking tasks and aggregates the answers to compute the top-$k$ output. Overall, the algorithm will perform $log_{s/k}(|O|)$ iterations, where each iteration may comprise several roundtrips to the crowdsourcing service. A single roundtrip is delineated by the issuing of $s$-ranking tasks and the processing of their answers. Lines 7-12 in Figure 1 constitute a roundtrip and are executed in parallel for each incomplete subset $p$.

One observation is that the algorithm computes the correct top-$k$ items if workers always return correct answers. However, due to worker disagreements and errors, and the existence of spammers, this guarantee is not feasible in the general case. Intuitively, the algorithm is likely to return highly ranked items if we have robust methods to aggregate the noisy answers for the same subset $p$ and to allocate more tasks to difficult subsets $p$. In what follows, we describe our implementation of these two methods.

## 3.2 Aggregating Answers to $s$-Ranking Tasks

Given a subset of items $p$, the goal is to aggregate several answers for the $s$-ranking tasks of $p$ in order to determine the most likely top-$k$ items in $p$. To solve this problem, we borrow the median rank aggregation method which has been proposed for the merging of ranked lists in the context of web-search results [4]. The method works as follows. Let $\tau_1, \ldots, \tau_m$ be the rankings returned by $m$ workers. Recall that each $\tau_j$ contains a ranking of $k$ items from $p$. Let $i$ be an item appearing in the rankings. Median-rank aggregation assigns to $i$ the median of its ranks in the different rankings $\tau_1, \ldots, \tau_m$, and then generates an aggregate ranking $\tau$ by sorting items according to their median ranks.

As shown in [4], the aggregated ranking $\tau$ has several desirable properties. Intuitively, $\tau$ can be seen as an "averaged" ranking that minimizes the average distance to the initial rankings $\tau_1, \ldots, \tau_m$. We can thus treat $\tau$ as the consensus ranking that arises from the answers of different workers, and the first $k$ items in $\tau$ as the most likely top-$k$ items in $p$. We establish the following result for median rank aggregations:

THEOREM 3.1. *The lowest possible median rank of a top-k item in the output of a median rank aggregation of $m$ rankings over a set of $s$ items is:*
L=$\lceil \frac{(q-1)s+(k-1)(m-q+1)+1}{m} \rceil$, $q = \lfloor \frac{m+1}{2} \rfloor$

We omit the proof due to space limitations. The significance of the theorem is that, when using median rank aggregation, it is sufficient to ask workers only for a ranking of the top-$L$ items of each subset instead of a ranking of all $s$ items, which can require less effort by the workers.

## 3.3 Determining the Number of Tasks

**Basic approach** A simple way is to allocate the same number of workers to all $s$-ranking tasks and to vacuously consider the results of all aggregations as complete. We call this strategy *basic*. This way we minimize the number of roundtrips but the algorithm likely overspends HITs on easy tasks and spends less than required on hard tasks.

**Adaptive approach** We propose an adaptive approach that uses a method to judge whether a subset is incomplete and determines the number of tasks posted for each subset at every roundtrip. By requesting a larger number of additional answers, the algorithm may finalize faster the top-$k$ items since consensus may be more likely. However, this also means a higher expense. On the flip side, by requesting fewer additional answers at each iteration, the algorithm can spend resources more judiciously but it will also increase latency since additional answers come at the cost of a roundtrip.

[1] At most one subset may contain fewer than $s$ items. If this subset has fewer than $k$ items, then no tasks need to be posted.

For the time being, suppose that we have a reliable method to detect whether the result of median-rank aggregation (see previous section) has high certainty and hence we can mark a subset $p$ as complete. Let $\psi_1, \ldots, \psi_n$ be a sequence of positive integers such that $\psi_1 < \cdots < \psi_n$, where $n$ is a parameter of our scheme. We initially post $\psi_1$ tasks per subset $p$, where $\psi_1$ represents the minimum count of human workers whose answers can hopefully provide a consensus in median-rank aggregation. We found that $\psi_1 = 3$ is a reasonable default. If additional answers are required, then we post an additional set of $\psi_2 - \psi_1$ tasks, which means that the algorithm will have a total of $\psi_2$ answers to evaluate the completeness of $p$. We continue in the same fashion until we post a total of $\psi_n$ tasks which is the maximum. At that point, $p$ is marked as complete by default.

In practice, we found that it works well to have three levels $\psi_1, \psi_2, \psi_3$ ($n = 3$) corresponding to easy, medium and high difficulty, with $\psi_1 = 3$ and $\psi_2 = \lfloor \psi_3 - \psi_1 \ /2 \rfloor$. The number of subsets at each iteration $i$ is: $p_i = (|C_i| \text{ div } s) + d_i$, where $C_i$ is the candidate set at iteration i and $d_i$ is 1 if $(|C_i| \bmod s) > k$ and 0 otherwise. For $i > 1 : |C_i| = (p_{i-1}) \cdot k + (1 - d_{i-1}) \cdot (|C_{i-1}| \bmod s)$. Starting with $C_1 = O$, we can compute all $p_i$'s until the last iteration and obtain the total number of subsets. Since the total number of posted tasks cannot exceed $Q$, we define $\psi_3 = \lfloor \frac{Q}{\Sigma_i p_i} \rfloor$.

The only remaining question is how to judge completeness for a subset $p$. In other words, given the answers to the $s$-ranking tasks for $p$ and the $k$ items resulting from median-rank aggregation, how can we determine that they represent the top-$k$ items with high certainty. Here we rely on the following intuition: If workers tend to disagree on their answers, this likely implies existance of spammers or difficult comparisons and allocating more workers to the task is likely beneficial in terms of quality. On the other hand, aggreement among workers can imply an easy task, and therefore accurate results, in which case we mark the s-ranking task as complete.

To measure agreement among the answers of workers, we resort again to the mechanics of median-rank aggregation from Section 3.2. We describe our technique with an example. Suppose that $p = \{A, B, C, D, E\}$, $k = 3$, and the answers we obtain from three workers are as follows: $\tau_1 = (A, B, C, D, E)$, $\tau_2 = (A, C, B, D, E)$ and $\tau_3 = (A, B, D, C, E)$. Median-rank aggregation will assign the following median-rank $r(i)$ to each item $i$: $r(A) = 1, r(B) = 2, r(C) = 3, r(D) = 4, r(E) = 5$, and hence the top-3 items are $\{A, B, C\}$. Note that the median ranks of the top-3 items are the same as if the workers were in perfect agreement, i.e., they all returned $(A, B, C)$ as their answer. Indeed, one can observe that there is high agreement in their answers, even though items may be ordered differently. This is reflected in the number of items that are assigned each median rank: there is exactly one item ($A$) with median rank 1, exactly one item ($B$) with median rank 2, and so on. In contrast, suppose that the returned answers corresponded to disagreement, as follows: $(A, D, C, B, E)$, $(C, A, B, D, E)$, $(B, E, A, C, D)$. Here, the median ranks are as follows: $r(A) = 2, r(B) = 3, r(C) = 3, r(D) = 4, r(E) = 5$. The top-$k$ items are again $\{A, B, C\}$, but among them no item is given median rank equal to one, and two items have median rank equal to three. Overall, the intuition is that we can measure worker agreement by examining the ties in the assigned median ranks.

Formally, let $\mathbf{r}$ denote a vector of length $L$, where $L$ is defined as the lowest median-rank that can be assigned to the $k$ items in the output of median-rank aggregation (Theorem 3.1). The component $\mathbf{r}[i]$ denotes the count of items (among the $k$ output items of median-rank aggregation) that are assigned median rank $i$. In the case of perfect agreement among workers, we can show that $\mathbf{r}[i] = 1$ for $1 \leq i \leq k$, and $\mathbf{r}[i] = 0$ for $k < i \leq L$ (see also our example). Conversely, disagreement results in $\mathbf{r}[i] = 0$ for some $i \in [1, k]$, or $\mathbf{r}[j] > 1$ for some $j \in [1, L]$. Using this idea, we measure the uncertainty of median-rank aggregation by the distance between $\mathbf{r}$ and the reference vector that represents perfect agreement. If this distance exceeds a threshold $t$ then we infer that workers disagree too much to infer the top-$k$ items with high certainty, and hence we mark $p$ as incomplete. We experimented with several distance metrics and settled on cosine similarity because of its robustness in our experiments. Threshold $t$ can be quantized to a few values, each one corresponding to a specific sensitivity to disagreements.

**Speculative and Redistributive optimizations** We employed the following optimization to reduce the number of roundtrips: at the beginning of each iteration, the starting point for the number of tasks (among $\psi_1, \ldots, \psi_n$) is determined as the $\psi_j$ of the previous iteration where most tasks where marked as complete. As an example, if most subsets were marked as complete using $\psi_2$ answers (the medium difficulty in our three-level scheme of $\psi_1 < \psi_2 < \psi_3$) in the previous iteration, then the next iteration will start with $\psi_2$ tasks per subset by default. The intuition is that comparisons can only become more difficult as iterations proceed. We observed that the use of an adaptive starting point reduced significantly the number of roundtrips to the crowdsourcing service. We call this optimization *speculative* as it assumes that we will need more workers without checking it. Finally, we tested an optimization that we call *redistributive*: we recompute $\psi_3$ and $\psi_2$ at every iteration using the remaining budget and considering the current candidate set as the initial itemset. This allows the algorithm to increase $\psi_3$ if initial iterations do not exhaust their allocated budget.

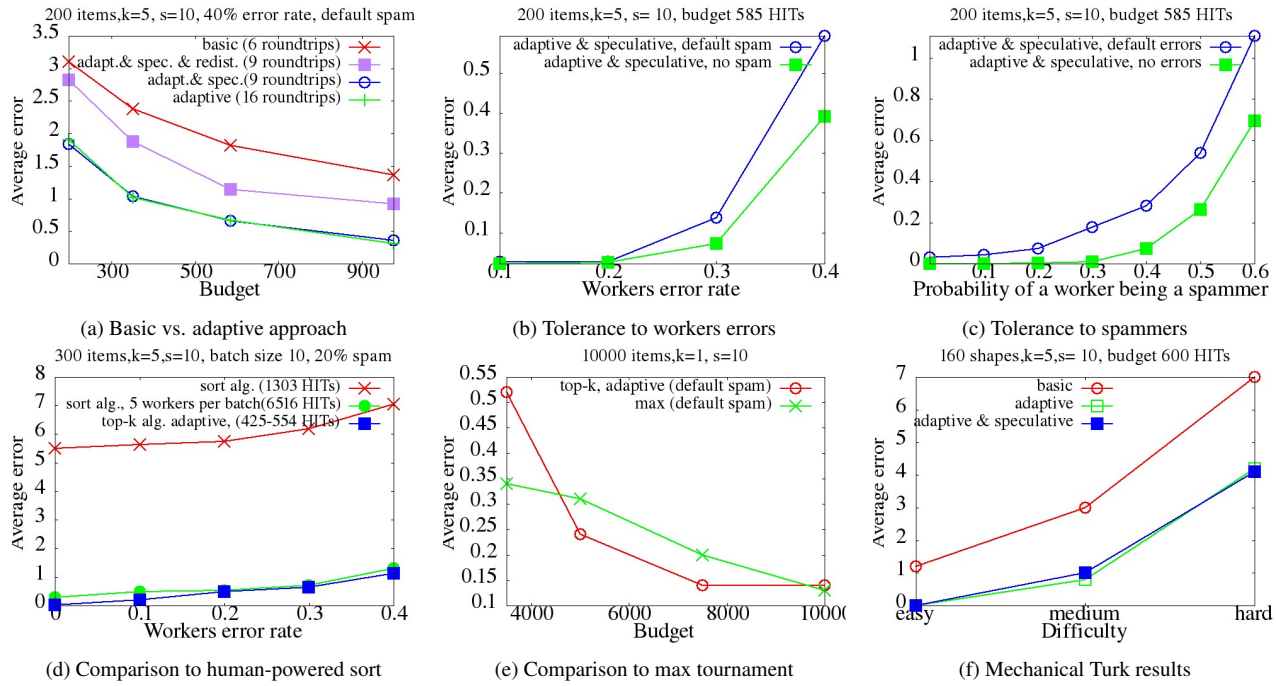
# 4. EXPERIMENTAL STUDY

## 4.1 Simulations

**Data** We generated synthetic itemsets of varying size $n$=50, 100, 200, 300. Every item $i$ in $O$ is associated with a real *value* $V(i)$ that determines a ground-truth ranking $\beta$ for the elements–a high value implies a high rank and vice versa. We generated three itemsets corresponding to different methods of assigning values to items: the first itemset assigns values uniformly at random in the interval $[1, n]$; the second itemset assigns integer values $1, 2, \ldots, n$; and, the third dataset assigns values from a Gaussian distribution with mean $\lceil \frac{n+1}{2} \rceil$ and deviation $\frac{n}{4}$, meaning that on average more than 98% of items take values in the [1,n] range with the majority centered around the mean. Our results for the third value distribution are better than for the other two and seem to favor our algorithm, since it is easier to differentiate the top-k items. We therefore omit this itemset from the presented results. The results for the uniform-random and integer-valued itemsets were practically the same, thus, we consider only the simplistic integer-valued itemset as it makes it easier to interpret our experimental results.

**Algorithms** We implemented both the *basic* and *adaptive* approaches and compare them. Unless otherwise noted, the adaptive method allocates any remaining HITs (up to the specified budget $Q$) to the very last $s$-aggregation that also determines the top-$k$ items. We also consider the extension of the adaptive approach with the *speculative* and *redistributive* heuristics.

**Modeling of human workers** A human worker may be either an honest worker or a spammer, with a probability that we vary in our experiments. A spammer returns any of the $s!$ permutations of the items. An honest worker behaves according to the model that Louis Thurston described in 1924 [7], formalizing the fuzzy way that humans perceive intensity of physical stimuli. Specifically, we assume that the worker perceives the value of an item $i$ by sampling




(a) Basic vs. adaptive approach

(b) Tolerance to workers errors

(c) Tolerance to spammers

(d) Comparison to human-powered sort

(e) Comparison to max tournament

(f) Mechanical Turk results

Figure 2: Results of experimental study

from a normal distribution that is centered around the actual value $V(i)$. Figure 3 gives an example of this process, assuming that the worker is presented with three items to rank. The perceived values determine the ranking returned by the worker. Clearly, the probability of committing errors (with respect to the ground-truth ranking) increases with the variance of the distribution.

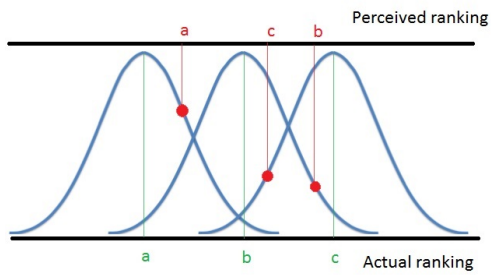



Figure 3: Example of a swap in the baseline ranking

We treat the top-$k$ items as a set, and so swaps that take place within the top-$k$ set are irrelevant and do not affect our metrics. Hence, we model the error rate of honest workers as the probability that there is a swap of two items of value distance equal to $k$. This probability corresponds to a specific variance of the normal distribution of the error model. As an example, an error rate of 30% for $k$=5 corresponds to a probability of more than 49% in swapping two neighboring items in a ranking task. We assume that all honest workers make errors with the same rate, that is, we do not assume the existence of workers with higher or lower expertise than others.

**Parameters** We fix the size of the ranking tasks at 10, the threshold beyond which workers abandon tasks at a high rate, as explained in [5]. The following table shows the remaining parameters of the experimental study, how we vary them and their default values.

| Parameters of experiments | | |
|---|---|---|
| **Parameter** | **Range** | **Default** |
| k | 1,5 | **5** |
| $s$ (size of ranking task) | - | **10** |
| Workers per ranking task | 3-15 | **3** |
| Spammer prob. | 10%-60% | **20%** |
| Error rate | 10%-40% | **25%** |
| Budget | 198-10000 | **585** |
| Threshold $t$ | - | **0.85** |

**Error metric** Let $e_1, .., e_k$ be the top-$k$ items returned by the algorithm. We define the error metric $\epsilon_r = \frac{\Sigma_{i=1}^{k} V(\beta_i) - \Sigma_{i=1}^{k} V(e_i)}{V(\beta_1) - V(\beta_k)}$ where $\beta_1, ..., \beta_k$ are the top items in the ground-truth ranking. We base our metric on values instead of ranks so as to take into account the similarity among the returned top-$k$ items and the actual top-$k$ items. Note that the metric acquires a natural interpretation for the integer-valued itemset we use in the experiments, where items with neighboring ranks in $\beta$ have a value distance equal to one. Hence, the factor $V(\beta_1) - V(\beta_k)$ is equal to $k$ and the formula expresses the average value error per item in the top-k list. As an example, when obtaining the top-1 list, $\epsilon_r$=1 means that on average our rankings will return the item with ground-truth rank equal to 2.

For every arrangement of the parameters we conducted 50 experiments (using different seeds for our random number generation) so that the average error value converges to its actual expected value. The error we report in the plots is the average error $\epsilon_r$ over all 50 runs.

**Basic vs. adaptive approach** Figure 2(a) shows the performance of the algorithm for three levels of available budget, namely 198, 351, 585 and 975 HITs, corresponding to 3, 9, 15 and 25 tasks respectively per subset of $s$ items for the basic algorithm. We observe that the adaptive variant reduces error significantly (between 1.5x and 2x) compared to the basic variant. The reason is the adaptive allocation of workers depending on the difficulty of ranking tasks, which increases in the later iterations. We also see that the

speculative heuristic can reduce latency significantly, cutting down the number of roundtrips by 44%. On the other hand, the redistribution heuristic performs worse than expected and results in a higher error. The reason is that the problem becomes dramatically more difficult in the last iteration, and hence the algorithm can benefit more from keeping the $\psi_i$ levels lower in the initial iterations, reserving an ample budget for the last iteration. Given these results, from now on we focus on the adaptive variant augmented with the speculative heuristic.

**Tolerance to workers errors**. Figure 2(b) shows the error metric as a function of the error rate of honest workers. The two curves correspond to no spammers and to the default probability of getting a spammer worker. As expected, the output error increases as the workers commit more errors. However, we observe that the algorithm can achieve high accuracy even for high error rates. For instance, for an error rate of 30%, which corresponds to a probability of more than 49% of swap for neighboring items, the algorithm returns the top-k list with average error less than 0.2, which corresponds to returning elements $\beta_1, \ldots, b_{k-1}, \beta_{k+1}$, i.e., missing only $\beta_k$ and substituting it with the next best element $\beta_{k+1}$.

**Tolerance to spammers** Figure 2(c) shows the error metric as a function of the probability of having a spammer worker answer an $s$-ranking task. The two curves correspond to the default error rate and to an error rate of zero, i.e., honest workers that do not commit errors. The results demonstrate that the algorithm is highly resilient to spammers, with a low error even for unrealistically high percentage of spammers (e.g., more than 40%). The key observation here is the algorithm can battle spamming effectively by detecting disagreements and posting more tasks as needed.

**Comparison to human-powered sorts** We implemented the *Compare* operator proposed in [5] and used it to obtain the top-$k$ items by performing a full sort of $O$. Figure 2(d) compares our algorithm with Compare for two possible allocations of HITs to the operator. Our algorithm outperforms Compare in terms of accuracy and total number of HITs. Specifically, to match the accuracy of our algorithm, Compare requires an order of magnitude more HITs. When allotted twice as many HITs as our algorithm, Compare has an error that is 6x higher. The reason is that Compare computes a full ranking of the items, which is clearly wasteful for our problem.

**Comparison to max tournament** We implemented the tournament max algorithm presented in [8] for obtaining the top-1 item. We tuned tournament-max to the average behavior of our users, which includes both spammers and honest workers who commit mistakes. Figure 2(e) shows the error metric for tournament-max and our proposed adaptive approach. As shown, the performance of our algorithm is comparable, yet the difference is that we do not require any a-priori knowledge about the behavior of human workers. Moreover, our technique can work for $k > 1$.

## 4.2 Experiments over Mechanical Turk

**Data** We tested three types of itemsets of size 160. For $k = 5$ and $s = 10$, this size ensures that all comparison tasks will be exactly $s$ in size, and the tournament will terminate in 5 iterations, decreasing the size of the candidate set to 80, 40, 20, 10 and 5 respectively. For all generated itemsets, the items are shapes and the goal is to find the top-$k$ items with the largest area. The first itemset comprises squares whose edge lengths range from 20 to 180. The difference in size among squares is easy to discern, therefore we consider this itemset as modeling an easy case. The second itemset comprises polygons of varying area and varying vertex count (between 4 and 10). The difference in shapes makes it harder to discern differences in area, and hence we consider this a medium-difficulty itemset. Finally, we create a high-difficulty itemset by reducing the difference in area size among polygons.

**Workers** We used the workers of Amazon's Mechanical Turk without imposing any restrictions nor requiring qualification tests. We paid $0.02 per task. We did not allow our algorithm to exhaust the budget (600 HITs) in every experiment, instead, we were more conservative in the last iteration choosing to continue posting HITs only if the algorithm kept identifying an $s$-ranking task as incomplete.

**Error metric** The error metric is the same as for the simulations, using the normalized area of each item as the value (that is, the area of the item divided by the difference in area between neighboring items). The results we report for each case are the average error from 3 identical experiments.

**Results** Figure 2(f) shows the error metric for our algorithm as a function of the difficulty of the itemset. For comparison, we also plot the basic variant and the adaptive variant without the speculative heuristic. The trends validate our simulation results, showing that the adaptive variant yields a substantial drop in the overall error in all cases. The errors increase sharply for the difficult itemset, but the actual ranking task is extremely challenging due to the very small differences in area among shapes. (In some sense, we made the task a bit too difficult for human workers.)

# 5. CONCLUSIONS

We introduced an algorithm to obtain the top-$k$ items from a large itemset, using the judgements of human workers in order to compare items. The distinguishing characteristics of our approach are that it allows workers to examine several items at a time, it does not require any a-priori knowledge about the errors of human workers, and it adapts dynamically to the varying difficulty of comparing items and the existence of spammers. Experimental results demonstrated that our algorithm yields accurate results, even when honest workers are likely to make mistakes and there is a large number of spammer workers.

**Acknowledgements.** This material is based upon work supported by the National Science Foundation under Grant No. 1251827.

# 6. REFERENCES

[1] N. Ailon. Active learning ranking from pairwise preferences with almost optimal query complexity. In *NIPS*, 2011.

[2] M. S. Bernstein, D. R. Karger, R. C. Miller, and J. Brandt. Analytic methods for optimizing realtime crowdsourcing. *CoRR*, abs/1204.2995, 2012.

[3] S. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, 2013.

[4] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.

[5] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *Proc. VLDB Endow.*, 5(1):13–24, Sept. 2011.

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

[7] L. L. Thurstone. A law of comparative judgement. *Psychological Review*, 34:273–286, 1927.

[8] P. Venetis and H. Garcia-Molina. Dynamic max algorithms in crowdsourcing environments. Technical report, Stanford University, August 2012.

[9] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. In *CrowdKDD 2012*. Stanford, August 2012.