Efficient Techniques for Crowdsourced Top-k Lists

Luca de Alfaro
UC Santa Cruz
luca@ucsc.edu

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

Neoklis Polyzotis *
Google Inc.
npolyzotis@google.com

Abstract

We propose techniques that obtain top-k lists of items out of larger itemsets, using human workers to perform comparisons among items. An example application is to short-list a large set of college applications using advanced students as workers. A method that obtains crowdsourced top-k lists has to address several challenges of crowdsourcing: there are constraints in the total number of tasks due to monetary or practical reasons; tasks posted to workers have an inherent limitation on their size; obtaining results from human workers has high latency; workers may disagree on their judgments for the same items or provide wrong results on purpose; and, there can be varying difficulty among tasks of the same size. We describe novel efficient techniques and explore their tolerance to adversarial behavior and the tradeoffs among different measures of performance (latency, expense and quality of results). We empirically evaluate the proposed techniques using simulations as well as real crowds in Amazon Mechanical Turk. A randomized variant of the proposed algorithms achieves significant budget saves, especially for very large itemsets and large top-k lists, with negligible risk of lowering the quality of the output.

Introduction

We address the problem of obtaining top-k lists of items out of arbitrarily large itemsets using crowdsourcing. An example application of a crowdsourced top-k query is selecting applications for college admissions. Educational institutions typically receive thousands of applications out of which they select a small set of the higher ranked applicants. Another example is the efficient selection of the top applicants for positions in industry, a process crucial for the success of companies. In both these examples the agents performing the comparisons between the applications are humans that apply some expertise. Non-expert crowds may be suitable in other instances. For example, a startup company in search of an appealing company logo may resort to the crowd to obtain a small set of candidate logos out of a set of millions of images. The company board can then make the final decision for the logo by manually examining that small set.

Previous studies applicable to the top-k problem in the context of crowdsourcing have major limitations. The work in (Marcus et al. 2011) is particularly wasteful in resources. Approaches that require less budget are applicable to very small top-k lists (Davidson et al. 2013) (Polychronopoulos et al. 2013) or just to finding the maximum (Venetis and Garcia-Molina 2012a) and are rigid with respect to the error model or the form of human computation tasks. Also, existing proposals in the literature generally lack a robust defense mechanism against the problem of spamming which is rampant in crowdsourcing (Ipeirotis 2010). The main contributions of our work are the following:

- We describe a class of algorithms that obtain top-k lists from the crowd for arbitrarily large itemsets. The algorithms issue comparison tasks whose total number is linear in the size of the input itemset, and the latency, measured as the number of roundtrips to the crowdsourcing service, is logarithmic in the size of the itemset.
- We propose a budgeting strategy, that based on an analytic estimate of the impact of adversarial users to the output, distributes the available budget efficiently across the stages of the algorithm.
- We propose a randomized variant of the top-k algorithms that can reduce the required budget drastically by taking a negligible risk of compromising the quality of the output result.
- We report results from experiments that test the performance of several instantiations of the proposed methods using simulations of human crowds and real crowds of Amazon's Mechanical Turk. The results show tolerance of the proposed techniques against errors and vandalism. We draw conclusions on the efficiency of the budgeting strategy, the randomized variant and the trade-off among latency, cost and quality of results. Using equal budgets, the method provides higher quality output than unbalanced rank estimation (Wauthier, Jordan, and Jojic 2013) and tournament algorithms (Polychronopoulos et al. 2013), while it has comparable performance with the method in (Davidson et al. 2013) incuring much lower latency. The randomized approach leads to significant budget saves, that can exceed 50% with a very low risk of losing items of the top-k list. The randomized algorithm is particularly beneficial for very large itemsets and large top-k lists.

*Author contributed to this work while at UC Santa Cruz.

Related work

A natural approach to the top-k problem are tournament style algorithms, as in (Venetis and Garcia-Molina 2012a) which obtains maxima and (Polychronopoulos et al. 2013) which obtains small top-k lists. The work in (Polychronopoulos et al. 2013) has comparable performance to the methods in (Venetis and Garcia-Molina 2012a) for the max problem and employs a technique that addresses random spamming but not vandalism, i.e. adversarial spamming by workers who invert the correctness of results. It suffers a major limitation as it can obtain top-k lists only for k's that are smaller than the size of the ranking tasks of human workers. That work and relevant research on ranking tasks in crowdsourcing (Marcus et al. 2011) has confirmed that there is an inherent limitation to the size of crowdsourced ranking tasks. On the other hand, the size k of a top-k list may be high. For example, we may query a database of millions of images for the top-100 list of images. The tournament algorithm would not scale, since we would have to ask human workers to provide rankings for sets of more than 100 images, a very large task for human computation. The method studied in (Davidson et al. 2013) is a randomized tournament approach in the initial stages of execution. When this method obtains a reduced enough candidate set, it invokes the method described in (Feige et al. 1994) to obtain the top-k list. The method in (Feige et al. 1994) has a very high latency, since it is essentially a heapsort algorithm for noisy comparisons, where comparisons take place sequentially and not in parallel. Thus, the technique in (Davidson et al. 2013) is also of high latency. Moreover, the method's analytic results hold under specific assumptions for the error functions of human workers. In practice, it is difficult to have any knowledge on worker error distribution a priori (Venetis and Garcia-Molina 2012b). Also, the study does not consider the case of adversarial human workers, which are a major challenge in human computation. Worker tasks are restricted to pairwise comparisons, while in reality human workers can perform tasks containing significantly more than two items. Our proposed techniques do not have this restriction. The work in (Ciceri et al. 2016) assumes prior knowledge on the quality of the items, which is not realistic for many applications. Relevant to our problem is the work in (Ailon 2012), which presents a way of sampling a quasilinear number of pair-wise comparison results for the purpose of learning to rank, but performs full sorting with no emphasis on the top-k. A randomized sorting algorithm was studied in (Wauthier, Jordan, and Jojic 2013), in which predicted permutations are more accurate near the top rather than the bottom of the sorted list.

Problem statement

We define a set of items I with cardinality n. A *ranking* of the itemset is a permutation of the n items. Let σ denote a ranking. By $\sigma(i)$ we denote the *rank* of an item i in ranking σ, that is, its position in the ranking. So, for the top element t in σ, $\sigma(t) = 1$. For two items i and j if $\sigma(i) < \sigma(j)$ we say that i is *ahead of* j in ranking σ. A *top-k list* of a ranking σ, is the set of items that contains every item i in σ for which $\sigma(i) \leq k$. If the top-k list is a ranking itself, where the positions in the list correspond to the ranks of the items in ranking σ, the list is called a *top-k ranked list*.

We measure the expense of a crowdsourcing algorithm as the total number of Human Intelligence Tasks (HITs) it issues to the crowd. Each time we issue a batch of HITs to the crowd to complete in parallel, we say that an extra *roundtrip* is executed. If we desire multiple workers to provide answers to the same task we need to issue separate HITs with identical content. We measure the latency of a crowdsourcing algorithm as the number of roundtrips it requires for termination.

We assume that the items in I can be ranked based on some attribute of interest. We call this ranking the *baseline ranking*, denoting it with β. Assuming β is unknown, our goal is to obtain the k items that are close to the top-k items in the ranking β, by issuing a limited number Q of ranking tasks to the crowd.

A Recursive Crowdsourced Top-k Algorithm

We introduce an algorithm for the top-k problem that leverages the knowledge from comparisons that the crowd has completed to decrease the set of candidate items of the top-k list rapidly by pruning away items that are not likely to be in the top-k list.

The algorithm decouples the size of the human comparison tasks from the size of the top-k list and can output results for arbitrarily large top-k lists.

In this section, we describe the algorithm and reason about its correctness in obtaining the top-k list of a larger itemset assuming that human workers provide correct answers, i.e. answers that are consistent with the baseline ranking. In reality, human workers can return incorrect results and lead to errors in the output of the algorithm. In the section *Handling inaccuracy of crowds* we describe techniques that make the algorithm robust against inaccurate answers of the crowd.

Figure 1 shows the pseudocode of the algorithm that we call *Crowd-Top-k*. Its input is an instance of the top-k problem. The algorithm has two stages: the *reduction* stage and the *endgame* stage.

We assume that the ranking tasks issued to the crowd have a limited size s. If the itemset I is large enough such that it can be partitioned into more than k partitions ($|I|/s > k$), the algorithm is in the reduction stage. It partitions I and obtains rankings through crowdsourcing. Once the method gets the rankings of the partitions through crowdsourcing, it picks the maxima, i.e. the top-1 items of each partition, and forms a new itemset $\hat{I}_m$. It then calls itself to obtain the top-k list of the itemset $\hat{I}_m$, i.e. the top-k list of the maxima from all the partitions of I. Thereafter, it uses the knowledge from the top-k ranked list of the maxima (structure $\hat{T}_m$) and the rankings over the subsets of I, to make comparison inferences and prune away items that are not candidates for being in the top-k list of I. Every partition of I whose maximum item is not in the top-k list of $\hat{I}_m$ can be discarded. This is due to transitivity of comparison results which are consistent with the baseline ranking; the rank of any item in

1 `Crowd-Top-k` (I,k,s)

Input : Itemset I, integer k, integer s (size of partitions)
Output: Top-k ranked list of items in I

2 **if** $(|I|/s) \leq k$ **then**
3 return *endgameTopk*(I,k,s);
4 *Partition* I *in subsets* $\{S_1,...,S_{|I|/s}\}$;
5 *Obtain full ranking* R_i *of each* S_i *through crowdsourcing*;
6 $\hat{I}_m \leftarrow$ *set of max items from all subsets* S_i;
7 $\hat{T}_m \leftarrow$ `Crowd-Top-k` $(\hat{I}_m,k,s)$;
8 $C \leftarrow \emptyset$;
9 **foreach** *item* i *in* $\hat{T}_m$ **do**
10 $C \leftarrow C \cup i$;
11 $R \leftarrow$ ranking R_v where i belongs to R_v;`//obtained in line 5`
12 **foreach** *item* j *in* R *different than* i **do**
13 **if** $((\hat{T}_m(i) + R(j)) \leq k)$ **then**
14 $C \leftarrow C \cup j$;
15 *return endgameTopk*(C,k,s);

Figure 1: Recursive algorithm *Crowd-Top-k* for the top-k problem

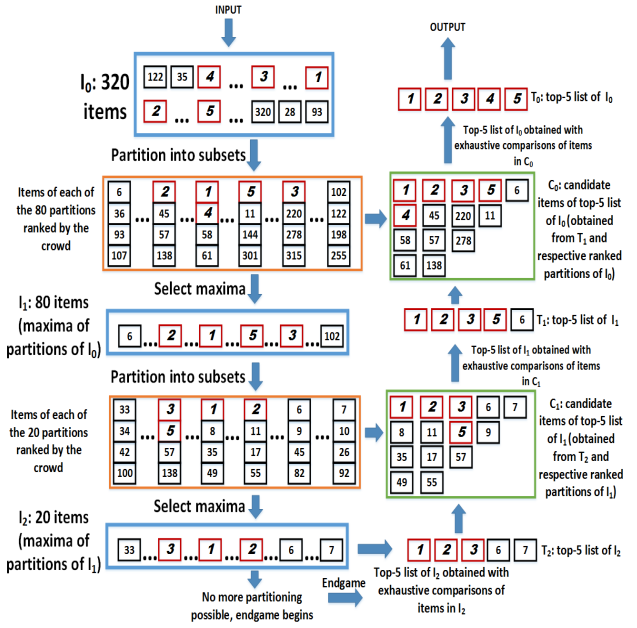

Figure 2: Example of execution of the *Crowd-Top-k* algorithm to obtain the top-5 list of an itemset of 320 items using crowdsourced ranking tasks of size 4

such a partition is at most the rank of its maximum, yet, the maximum has lost to more than k items. Thus, no item of the partition can be in the final top-k list. For each partition of I whose maximum belongs to the top-k ranked list of $\hat{I}_m$, assuming correct comparisons, we can infer the following: the final rank of an item of the partition is at least the rank of the maximum of the partition in the top-k ranked list $\hat{T}_m$ of $\hat{I}_m$ plus the rank of the item in the partition. For items with rank that can still be less than k, that is, their rank in the partition augmented by the rank of the partition's maximum item in $\hat{T}_m$ is less or equal than k, stay in the game and form part of the new set of candidate items for the top-k list of I (set C).

The new set of candidate items C contains $1+2+...+k = \frac{k\cdot(k+1)}{2}$ items if $k < s$, and $(k-s)\cdot s + \sum_{i=k-s+1}^{k} k-i+1$ for $k \geq s$, which in both cases is $O(k \cdot s)$. The set of candidate items cannot be further partitioned into k partitions. For this reduced candidate set, the algorithm calls a method that can obtain the top-k list of small itemsets for large k and small itemsets. We informally call the problem of obtaining top-k lists from itemsets where k is a significant portion of the size of the itemset as the 'Endgame top-k' problem. We later describe several ways of implementing this method.

Example Figure 2 shows the execution of the *Crowd-Top-k* algorithm to obtain a top-5 list ($k = 5$) of an itemset I_0 with 320 items, using crowdsourced ranking tasks of size 4. In the figure, we tag each item with a number which is its rank in the baseline ranking. The algorithm is unaware of the baseline ranking. For the top-5 items, which the algorithm aims to retrieve, we use larger italic font. The left part of the figure is the reduction phase, and the right part of the figure is the endgame which crawls back to the initial recursive call to obtain the final top-k list. Initially, the algorithm partitions the itemset into 80 partitions of size 4, and the crowd ranks the items of each partition. We observe that item 4 happens to fall into the same partition with item 1 in this random partitioning. The algorithm forms itemset I_1 from the maxima of the partitions and calls itself on I_1. Since I_1 is a big itemset (80 items), the method partitions it into 20 partitions of size 4 and obtains rankings of the items in each partition using the crowd. Note that item 5 happens to fall into the same partition as item 3. The method forms set I_2 from the maxima of the partitions of I_1 and calls itself on I_2. The I_2 is small (20 items), and cannot be partitioned in more than 5 partitions, therefore, the endgame begins. The algorithm obtains the top-5 list of I_2 using a method for the endgame top-k problem. In this example, we obtain the top-5 list T_2 by exhaustive comparisons of the items in I_2. Top-5 ranked list T_2 does not contain items 4 and 5, as they do not belong to I_2. The endgame proceeds by popping the recursive stack to obtain the candidate items of the top-5 list of itemset I_1. It forms C_1 from the partitions of I_1 that contain the top-5 items of I_2 and includes items whose rank in I_2 can still be less than or equal than k. It then obtains the top-5 ranked list T_1 of I_1 performing exhaustive comparisons in C_1. The list T_1 includes item 5 retrieved from the partition whose maximum is item 3. The endgame proceeds, forms candidate set C_0 from partitions of I_0 and T_1 and obtains the final top-5 list of I_0, which includes item 4 obtained from the partition

of I_0 whose maximum is item 1. During the backtracking, the size of each of the candidate sets C_i is less than $k \cdot s$, where s is the size of the ranking tasks.

Assuming that the size s and k are small compared to the size of the input itemset, we can consider them fixed and prove the following result.

Theorem 1. *Algorithm* Crowd-Top-k *issues a number of ranking tasks that is linear in the size of the input itemset I and has a logarithmic latency, assuming a bound in the size of partition s and the size of the top-k list.*

Proof To reason about the algorithm's complexity, we first need to calculate the number of recursive calls throughout the algorithm's execution. At each recursive call, the size of the candidate set is reduced by s. For $k \cdot s$ less than the current size of the input itemset the recursion continues. Thus, for the number of recusrive calls r the following condition holds:

$$\frac{n}{s^{\mathrm{r}}} \leq k \cdot s \Rightarrow n \leq k \cdot s^{r+1} \Rightarrow \frac{n}{k} \leq s^{\mathrm{r}+1} \Rightarrow$$
$$\Rightarrow log(\frac{n}{k}) \leq (\mathrm{r}+1)\log s \Rightarrow \log_s(n/k) - 1 \leq \mathrm{r}$$

Thus, the number of recursive calls is:

$$\mathrm{r}(n,k,s) = \max\{\lceil \log_s(\frac{n}{k}) - 1 \rceil, 0\}$$

Denoting the latency of the endgame method with L (which can be constant for some endgame methods but is by definition constant if we fix k, s), the overall latency of the *Crowd-Top-k* algorithm is thus:

$$\mathrm{lrec}(n,k,s) = (L+1) \cdot r(n,k,s) + L$$

We denote the number of ranking tasks issued by the endgame method, which is a function of k and s to be U. The number of ranking tasks is at most:

$$\mathrm{tasksrec}(n) = n \cdot \sum_{i=1}^{\mathrm{r}(n,k,s)} \frac{1}{s^i} + (r(n,k,s)+1) \cdot U$$
$$\leq \frac{n}{s-1} + (r(n,k,s)+1) \cdot U$$

because the geometric series sum $\sum_{i=1}^{\mathrm{r}(n,k,s)} \frac{1}{s^i}$ is bounded by limit: $\frac{1}{s-1}$.

Since $r(n,k,s)$ is logarithmic $\mathrm{tasksrec}(n) \in O(n)$ for fixed s,k. □

For small top-k lists ($k < s$), where the tournament algorithm is applicable, the *Crowd-Top-k* algorithm would require $\frac{n}{s-1} + k^2 \cdot \lceil \log_s(\frac{n}{k}) \rceil$ tasks.

For example, if $n = 10,000$, $k = 5$ and $s = 10$, the tournament algorithm would need $2,000$ ranking tasks and 11 rountrips, while the *Crowd-Top-k* algorithm would require roughly $1,200$ ranking tasks in 6 roundtrips. The algorithm in (Feige et al. 1994), which is the same as the algorithm in (Davidson et al. 2013) when we risk no loss for the items in top-k list, has a latency in $\Omega(n)$, a fact that can make these methods prohibitively time consuming for large itemsets. In the presence of randomization, the latency in (Davidson et al. 2013) best-case increases linearly with the size of the top-k list.

Endgame top-k algorithms

The recursive algorithm *Crowd-Top-k* is generic with respect to the method that it uses to obtain the top-k list when the endgame begins. We are free to choose the algorithm that obtains the top-k at the endgame, i.e. algorithm *endgame-Topk* of line 3 and line 15 of the pseudocode. Each one has a different tradeoff among cost, latency and quality of answers. We explored several methods for the implementation of the endgame. The four endgame methods described below are applicable to any size of top-k lists. For smaller top-k lists, that is, for $k < s$, the tournament algorithm in also applicable in the endgame.

Human-powered sort algorithm One option is to use the human-powered sorts algorithm (Marcus et al. 2011) which issues a quadratic number of ranking tasks in batches. We implemented the *Compare* operator proposed in (Marcus et al. 2011) and used it to obtain top-k lists for small itemsets. The number of HITs cannot be smaller than the lower bound $\frac{n(n-1)}{s(s-1)}$ where n is the number of items and s the size of the batch. The algorithm's advantage is that it issues all necessary tasks in a single roundtrip. For $n = k \cdot s$, which is the base case of the recursion, it issues a number of comparison tasks in the order of $O(k^2 \cdot s^2)$.

Unbalanced rank estimation The study in (Wauthier, Jordan, and Jojic 2013) presents a randomized algorithm for obtaining the sorted list of a large itemset by pairwise comparisons. The method is simple; every pairwise comparison has a fixed probability of being chosen and sent to the crowd in a single rountrip. The items are sorted based on the number of items against which they have won in comparisons. That is, each item's estimated rank is proportional to the fraction of the items winning it over all items with which it has been compared to. The authors provide an analysis for the expected quality of the results. The quality increases as the sampling rate increases. They prove that the expected accuracy is significantly higher for the top items of the list.

Comparisons inference algorithm We consider a method that is based on inferences due to transitivity. Initially, we randomly pick some comparisons that we issue to the crowd. Then, we choose the comparisons that are most informative, that is, the ones that lead to the highest number of inferred comparisons. The problem of selecting the most informative comparisons bears some similarity to the 'Next Votes Problem' that Guo et al. studied in (Guo, Parameswaran, and Garcia-Molina 2012). Finding the optimal solution is NP-hard. The analysis in (Guo, Parameswaran, and Garcia-Molina 2012) does not extend to the top-k problem. Since the optimal solution is difficult, in our study, we employ a heuristic that examines the inference gains for each of the pairwise comparisons and sorts all unknown comparisons by their inference gains.

Quick-sort top-k variant We consider a variation of quick-sort adapted for obtaining the top-k. The method randomly picks a pivot and finds its rank in the itemset by comparing against all items. The comparison tasks to compute the rank of the pivot are all issued in parallel in a single

roundtrip maintaining low latency. The rank of the pivot is the number of items against which it has lost, augmented by 1. If the rank of the pivot is k the top-k list is the pivot and all the items that have won the pivot in comparisons. If the rank of the item is larger than k, we can exclude the pivot from the top-k list along with all the items that it has won. We then repeat the same process and obtain the top-k list of the items that are still candidate members of the top-k list.

Handling inaccuracy of crowds

The key for obtaining high quality output with the *Crowd-Top-k* algorithm is to obtain high quality results using crowdsourcing for each of the subsets S_i of the input itemset I. (line 5 of pseudocode in Figure 1)

Rank aggregation of multiple answers to a ranking task

We assign each ranking task to multiple workers and obtain a single ranking out of the potentially different rankings that workers provide for a particular subset.

Producing a single ranking out of multiple rankings over the same set of items is a process called *rank aggregation*. The *optimal* aggregated ranking is the one that minimizes its distance from all the rankings with respect to a specific ranking distance metric. Two prominent binary distance metrics for rankings are the *Kendall tau* distance and the *footrule* distance.

Computing a footrule-optimal aggregation is tractable. Computing a Kendall-optimal aggregation is equivalent to the *minimum feedback arc set* problem which is NP-hard. An early result by Diaconis and Graham (Diaconis and Graham 1977) proved that the Kendall Tau distance and the Spearman's footrule distance are 'equivalent', since they are within a factor of 2 from each other. As shown in (Fagin, Kumar, and Sivakumar 2003), the median rank aggregation method gives a footrule-optimal aggregation in many cases. In (Fagin et al. 2004), median rank aggregation was proven to be optimal, within constant factor of 2, for rankings without ties. We choose it as the method of rank aggregation of the *Crowd-Top-k* algorithm.

Distributing budget to ranking tasks

Human workers that contribute to Human Intelligence Tasks can be honest or spammers. Spammers can be *random spammers* or *adversarial spammers* (also called *vandals*). For ranking tasks, a random spammer answers with a random ranking of the items of the task, whereas an adversarial spammer provides the correct ranking but in reverse order.

We use two different methods of allocating budget to ranking tasks, the first for the case when we expect random errors and spammers, and a second for when there also exist adversarial spammers in the crowd. In the former case, we use the adaptive algorithm proposed in (Polychronopoulos et al. 2013) that can estimate the difficulty of tasks or the presence of an usually high number of random spammers in a median rank aggregation on the fly, based on the diversity of answers. This technique adaptively uses existing budget by allocating fewer workers to seemingly easy tasks and placing more effort on seemingly difficult tasks. The method is not relevant in the presence of adversarial spammers, because vandals provide the same answer so we cannot use diversity as a red flag. For this case, we use a budgeting strategy based on an analytic estimation of the impact of vandals on the output of the algorithm.

In crowds that undergo quality control (e.g. crowds tested with the Gold Standard Data method of Crowdflower) we generally see zero or negligible percentage of vandals and we only expect random errors and incidental random spamming, as in fact even honest workers may approximate the behavior of random spammers if fatigue or other factors lowers their quality. A less moderated crowd such as that of Amazon Mechanical Turk can have a non-negligible percentage of vandals. As noted in a previous study (Venetis and Garcia-Molina 2012b), the distribution of errors by honest workers is very difficult to obtain, because it depends on the similarity of items in the baseline ranking. However, it is feasible to estimate the percentage of vandals in the crowd by sampling from the crowd using a known dataset. This is because vandals provide a very special answer to ranking tasks; if the answer we obtain is close to an inverted ranking, we can infer that this is likely the result of vandalism.

Addressing vandalism In the presence of adversarial spammers, we need to assign the largest possible number of workers to each aggregation. The suitable number of allocated workers depends on the budget constraints and the impact that the prevalence of vandals in a particular aggregation can have on the final output result. We provide an analytic estimate of this impact based on a simplified error model. Exploiting this analysis, we devise a principled method for allocating workers, given a particular budget Q for ranking tasks. In the experimental study, we check the robustness of the method under realistic error assumptions.

We assume the percentage of vandals in the crowd is known. In practice, it is feasible to estimate the actual percentage, by sampling answers from the crowd using a set of items for which the correct ranking is known, before executing the algorithm.

We approximate reality by assuming that workers are either correct or adversarial.

We define error in the output top-k list, as the percentage of items of the top-k list that we lose; in other words, the number of non-members of the top-k list that appear in the final set of k items that is returned by the algorithm.

We denote with m_i the number of workers that we assign to a ranking task at recursive depth i, where m_0 is the number of workers we allocate per partition at the initial call of the algorithm. We denote with z_i the number of partitions at recursive call i, with z_0 being the number of partitions at the initial execution.

We seek the vector $(m_0, ...m_{\mathrm{r}(n,k,s)})$ that minimizes the expected error due to vandalism, where $\mathrm{r}(n, k, s)$ is the number of recursive calls of the algorithm (see proof of Th. 1) .

We denote the probability that a worker is a vandal, which is roughly equal to the percentage of workers in the crowd, by v. We assume that we have a limited budget Q of crowdsourced ranking tasks.

Picking a worker from the crowd is a Bernoulli trial, so the probability that an aggregation of m workers contains d vandals is $\mathrm{B}(m,d) = \binom{m}{d} v^d (1-v)^{m-d}$. When a ranking task is performed by multiple workers and the majority of those workers are adversarial, the aggregation will return an inverted ranking because the median ranks of all items will be inverted. Therefore, the probability that median rank aggregation outputs an inverted ranking is:

$$g(m) = \sum_{i=\lceil \frac{m}{2} \rceil}^{m} \mathrm{B}(m,i)$$

In the worst case, all the items of the top-k list that belong to a partition can be lost due to the adversarial behavior. This is because the minimum item of the partition, which is erroneously indicated as maximum, is likely smaller than the maxima of other partitions and would be excluded from the candidate set, along with all the items in the partition.

Since the partitioning is random, the expected number of items of the top-k list in a partition is k/z_i.

Thus, for the expected error e due to vandalism of a single partition at recursive depth i:

$$\mathrm{E}(\mathrm{e}) \leq g(m_i) \cdot \frac{k}{z_i}$$

Using the union bound of the expected error across all partitions across all recursive calls:

$$\mathrm{E}(err) \leq \sum_{i=0}^{\mathrm{r}(n,k,s)} z_i \cdot g(m_i) \cdot \frac{k}{z_i} = k \cdot \sum_{i=0}^{\mathrm{r}(n,k,s)} g(m_i)$$

where $\mathrm{r}(n,k,s)$ is the number of recursive calls.

We seek solution $(m_0, ...m_{\mathrm{r}(n,k,s)})$ that minimizes the above error bound subject to the budget constraint:

$$\sum_{i=1}^{\mathrm{r}(n,k,s)} z_i \cdot m_i \leq Q$$

This is an optimization problem that resembles a non-linear variant of the knapsack problem (assigning a negative sign to the error bound to convert it to a maximization problem). Solving the above optimization problem provides us with a budgeting strategy that distributes the available budget to ranking tasks across recursive calls. For large itemsets that require several recursive calls, exhaustive search of all combinations that maximally satisfy the budget constraint is infeasible due to combinatorial explosion. We thus employ a greedy approach similar to the ones used for knapsack to obtain an approximately optimal solution to the optimization problem.

Randomized variant

We propose and implement a randomized variant of the recursive algorithm *Crowd-Top-k* that reduces the cost of the endgame by undertaking a small risk of losing items of the top-k list from the final result.

The key observation that allows us to devise a randomized algorithm is that forming the new candidate set using the top-k maxima and items from partitions where the top-k maxima belong is necessary because of the non-zero probability that more than one items of the top-k list of the given itemset may fall in the same partition. Therefore, the top-k list $\hat{T}_m$ of the maxima, and the top-k list of the itemset always have a non-empty intersection but are often not the same.

We call the event that exactly w items of the top-k list of the itemset fall in the same partition a *w-fold collision*. The top-w item of a partition whose maximum belongs to $\hat{T}_m$, can belong to the top-k list of I with a probability that is at most the sum of the probabilities of f-fold collisions taking place, for all f where $w \leq f \leq k$. If this bound is low enough so that it is in practice negligible we can keep in the candidate set only the top-$(w-1)$ items of the partitions whose maxima belong to $\hat{T}_m$ instead of the number of items that are necessary in the standard version of the algorithm. In the extreme case where a f-fold collision, where $f \geq 2$, is highly unlikely, executing the endgame is redundant; the top-k list of the itemset is most likely $\hat{T}_m$ itself, i.e. the top-k list of the maxima of the partitions. We can quantify an upper bound of the probability of a w-fold collision happening. As an example, for an itemset of $1,000,000$ items, partition size of 5 and $k = 10$, the probability that more than one of the items of the top-k list fall in the same partition cannot exceed $2.5 \cdot 10^{-4}$. Thus, skipping the execution of the endgame and returning the top-k list of the maxima $\hat{T}_m$ as the top-k list of the entire itemset carries a risk of at most 0.025% of missing items of the top-k list. We can deem this as low, and in many cases, it is significantly lower than the risk of losing items due to vandalism or errors by the crowd.

Formally, the probability $q(w)$ of a w-fold collision occuring at a given partition at recursive call i is:

$$q(w) = \binom{k}{w} \cdot \frac{1}{z_i}^{w} \cdot (1 - \frac{1}{z_i})^{k-w}$$

and for the probability across all partitions at recursive call i we can use the union bound to obtain: $\Pr[w,i] \leq z_i \cdot q(w)$.

The probability $r(c,i)$ of losing an item of the top-k at recursive call i by promoting only $c \cdot k$ items to the endgame methods instead of $O(s \cdot k)$ is bounded by : $\sum_{w=c+1}^{w=k} z_i \cdot q(w)$.

The randomized variant takes a risk threshold as additional input. This represents the level of risk of losing items of the top-k list due to the randomization which the user of the algorithm deems acceptably low. The algorithm then pre-calculates the value c_i for each recursive call i, which maintains the total risk bound less or equal to the risk threshold, according to the above upper bound. During the execution of the algorithm, at each recursive call i, the algorithm invokes the endgame top-k method for c_i items of each partition, which correspond to the top-c_i items of partitions where items of $\hat{T}_m$ belong, instead of $O(s)$ items in the non-randomized case. For $c_i = 1$, there is no invokation of the endgame top-k algorithm; the algorithm returns the top-k list of the maxima $\hat{T}_m$ as the top-k list of the itemset.

Experimental study with simulated crowds

Methodology

Data We conducted experiments using synthetic data and a simulation of crowds to evalutate the performance of the proposed algorithms.

Each item has a distinct integer value that represents its quality and neighboring items in the baseline ranking have value distance of one. We also experimented with the more realistic assumption of the Gaussian distribution, which favors our methods since the top items are easier to distinguish, but in the interest of fairness we report results under the equi-distant assumption for the value of the items.

Modeling of human workers Human workers can be either honest workers or spammers, where spammers can be either random or adversarial as described above.

Honest workers report the truth consistent with the value of the items but with occasional random swaps according to the Thurstonian model (Thurstone 1927) that describes how human agents distort the perception of physical stimuli. The same model is also used in (Venetis and Garcia-Molina 2012a) and (Polychronopoulos et al. 2013).

Parameters of experiments		
Parameter	**Range**	**Default**
Size of itemset	1,000-100,000	**10,000**
k (size of top-k list)	5-50	**50**
Size of partition	-	**2-10**
Random spam	10%-40%	**20%**
Vandals	1%-15%	**1%**
Error rate	10%-40%	**25%**
Budget	4K-1.5M	**15K**

The above table shows how we vary the experimental parameters and their default values.

Error measure The error measure we use is: $\epsilon_r = \frac{\Sigma_{i=1}^{k} V(\beta_i) - \Sigma_{i=1}^{k} V(e_i)}{V(\beta_1) - V(\beta_k)}$ as in (Polychronopoulos et al. 2013) where $\beta_1, ..., \beta_k$ are the items in the baseline top-k list and $V(\beta_i)$ is the quality value of item i.

Cost metric We model the amount of work, and thus the cost, of a HIT as $c = \frac{s \cdot \log_2 s}{2}$, where s is the size of the ranking tasks. If all HITs have the same size throughout the execution of the algorithm, the amount of work is proportional to the number of HITs. Otherwise, HITs with higher sizes require more effort.

Results

Comparative study for small top-k lists We compare the *Crowd-Top-k* algorithm against the tournament algorithm which is applicable to small top-k lists (smaller than the size of a ranking task). We set the minimum number of workers per aggregation to 3, for high quality. We use the tournament algorithm itself as the endgame method of the recursive algorithm *Crowd-Top-k*. We also compare against the algorithm in (Feige et al. 1994) using roughly 100,000 pairwise comparisons, which is equivalent to 6000 ranking tasks of size 10 according to the cost metric.

Figure 3 shows the result of an experiment that obtains the top-5 list of $10,000$ items with a fixed budget of $6,000$, default spammer percentage and increasing error rate. The *Crowd-Top-k* algorithm performs better for all error rate levels while the spread between the performance of the two methods increases for higher error rates. For an error rate of 10% the *Crowd-Top-k* algorithm yields results with almost zero error. The results confirm the superiority of the recursive approach to the tournament algorithm. The number of roundtrips fluctuates for the two methods, ranging from 11 to 30, due to the adaptive allocation of tasks to human workers. The method of (Feige et al. 1994) has a very high latency ($> 10,000$ roundtrips) which makes it unsuitable in real settings and for this setting it also achieved poorer quality results than the recursive algorithm.

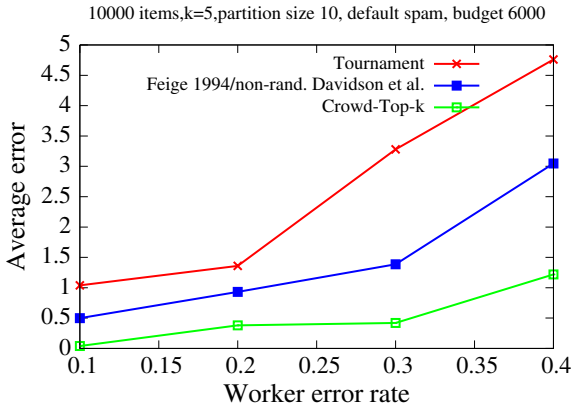

Figure 3: Comparative performance of the *Crowd-Top-k* algorithm and the tournament algorithm for increasing error rate

Performance of the *Crowd-Top-k* algorithm for large top-k lists ($k > s$) We conducted experiments with several instantiations of the *Crowd-Top-k* algorithm for large top-k lists using different methods for the endgame.

The human-powered sort algorithm (Marcus et al. 2011) for the endgame was highly inefficient for the top-k problem in terms of required budget and tolerance to errors, so we omit the results of its performance. We tune the methods we tested appropriately to use the same budget so that the comparison is fair and meaningful. The budget is set to $15,000$ pairwise comparison tasks ($s = 2$) for obtaining the top-50 list of an itemset of $1,000$ items. We report the results of five different methods to obtain the top-k list:

- Unbalanced rank estimation (URE) proposed in (Wauthier, Jordan, and Jojic 2013) to retrieve the top-k list out of the entire itemset.
- Recursive algorithm *Crowd-Top-k* with unbalanced rank estimation (URE) as the method for the endgame.
- Recursive algorithm *Crowd-Top-k* with the comparisons inference algorithm as the method for the endgame. We obtain the same number of pairwise comparisons (three times the size of the dataset) from the crowd in all roundtrips.
- Recursive algorithm *Crowd-Top-k* using the quick-sort top-k variant algorithm for the endgame.

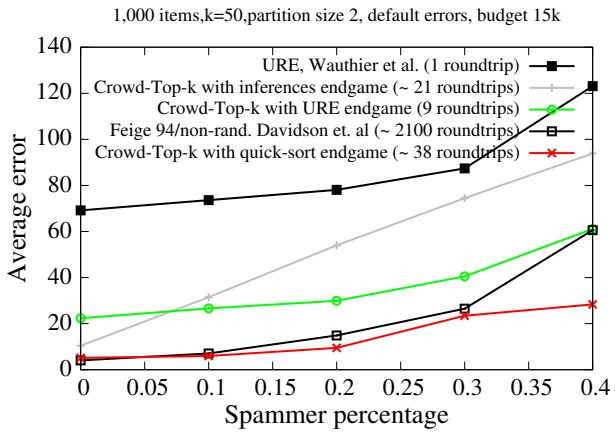

Figure 4: Comparative performance of methods for large top-k lists

- Method described in (Feige et al. 1994) for the top-k problem which has the same order of latency with (Davidson et al. 2013) and it is in fact the same algorithm when it is not randomized.

Figure 4 shows the results of the five methods for default error noise and increasing percentage of random spammers. The unbalanced rank estimation method has the smallest latency as it uses a single roundtrip but provides the poorest quality results. The *Crowd-Top-k* algorithm using the quick-sort top-k variant in the endgame is the method that makes the most efficient use of the budget providing the highest quality output. The method of (Feige et al. 1994) is providing results of comparative quality, yet, the latency is prohibitively high (>2000 roundtrips). The *Crowd-Top-k* method using unbalanced rank estimation for the endgame maintains the error low and the latency at 9 roundtrips. It is a balanced approach that provides low output error with low latency. The *Crowd-Top-k* method with the comparisons inference algorithm provides good results comparable to those of the quick-sort top-k variant for lower spammer percentages and with lower latency but as the spammer percentage increases the output error approaches that of the unbalanced rank estimation. The comparisons inference algorithm is vulnerable to error propagation which explains the increase in the output error as spammers increase.

Handling of vandals We described a method that returns the number of workers that we should allocate to the ranking tasks at each recursive call for a given budget, in the presence of vandals (adversarial spammers). The probabilities we use in our analysis were upper bounds obtained from union bounds and may not be tight.

A question we need to address is whether the budgeting strategy works efficiently in practice. We conducted experiments to evaluate the performance of the budgeting strategy. For an itemset of 1,000 items, $s = 2$ and $k = 20$ the algorithm is called 5 times, one initially and 4 recursively. The budgeting algorithm returns a vector $V = (m_0, ...m_4)$ where m_i is the number of workers per partition at recursive call i, with m_0 being the workers per partition at the initial call of the algorithm. If the budgeting strategy provides a roughly optimal budget arrangement we expect the error of the output to increase when we use a different budget arrangement. In particular, we expect the error to get higher as the distance of the vector that represents the budget arrangement from the vector of the optimal budget arrangement V increases.

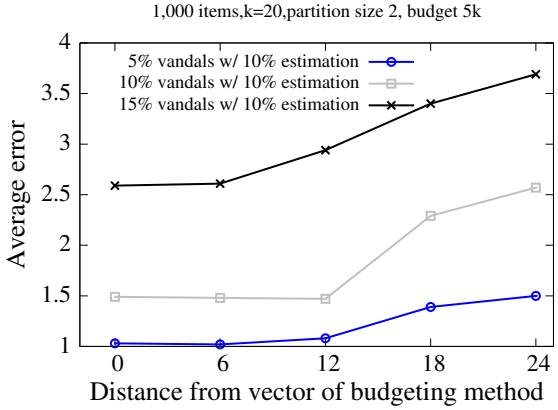

Figure 5: Performance of budgeting strategy

Figure 5 demonstrates the error for three levels of vandals in the crowd, at 5% and 10% and 15%. We evaluate the *Crowd-Top-k* algorithm with URE for the endgame using several budget arrangements that maximally satisfy the budget constraints, that is, even one pair of extra workers at some stage would exceed total available budget for ranking tasks which stands at 5,000. The total budget stands at 17,000. The x-axis represents the Manhattan distance to the vector V that the budgeting method returns as optimal arrangement.

We assume that the estimation of the vandal percentage of the crowd stands at 10%. The random errors performed by honest workers are at the default level. For an actual percentage of vandals at 10% we observe that in the vicinity of vector V, that the budgeting strategy returns, we obtain the lowest error results. It is reasonable to assume that the vandal percentage estimate may be an underestimate (e.g. -50%) or an overestimate (e.g. +50%), since it is the result of sampling. We thus report results for the same level of vandal percentage estimate at 10% but for the actual levels of budget standing at 5% and 15%. We observe that the results are qualitatively the same as when the actual percentage is at 10%, and the lowest error is in the vicinity of the vector V of the budgeting strategy.

The results confirm the relevance of our analysis and show that our strategy distributes the budget efficiently across the algorithm's stages even for difficult comparisons where honest workers do not always provide correct answers. In particular, for budgeting vectors that are close to V the performance is virtually unchanged (can be slightly inferior, or even slightly superior as V is not necessarily the real optimal arrangement since it is derived from an analysis based on union bounds), while for vectors with high distance from

V, the error increases in all cases. We also observe a small sensitivity of the budgeting strategy to errors in the estimate of the vandals percentage from crowd sampling.

Performance of randomized variant We expect the randomized approach to be more efficient for very large itemsets and large k's. This is because for large k's we feed many items to the endgame, and the number of pairwise comparisons increases quadratically with k. Also, for large itemsets the risk of w-fold collisions during partitioning decreases. We evaluate the randomized recursive algorithm *Crowd-Top-k* with URE for the endgame, for a large itemset of 100,000 items, $k = 50$, and partition size of 10. The size of comparison tasks is different in the reduction stage of the algorithm and the endgame stage (10 and 2 respectively). In the interest of fairness, we report the amount of work according to the cost metric rather than the number of HITs. Otherwise, reporting only the number of HITs would favor the results, since the randomized approach achieves budget saves only for the comparison tasks that take place in the endgame stage, which in this case are smaller and therefore require less effort.

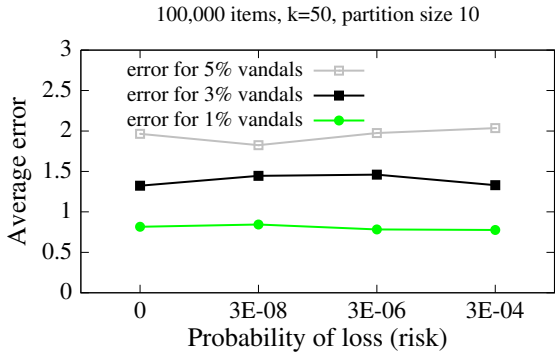

Figure 6: Error for increasing probability of loss

Figure 6 demonstrates the error as the risk increases from zero (non-randomized) to $3 \cdot 10^{-4}$ for default errors by honest workers and vandals at 1%, 3% and 5%. The risk is the probability that we lose some item of the top-k list from the final result, due to the randomization. We observe that the error is essentially unchanged and even paradoxically decreases slightly instead of increasing in some cases. This happens because the risk of loss remains negligibly low, but the number of items that we send to the URE method drops significantly. For constant parameter $c = 200$ of the URE method, the sampling rate increases, and for a smaller dataset this leads to an enhanced output. Still, the required budget decreases because the number of all possible pairwise comparisons decreases quadratically, and we require significantly less budget to achieve comparable or even superior quality. We can see the use of budget, expressed in required amount of work, in the following table.

Risk	0	$3 \cdot 10^{-8}$	$3 \cdot 10^{-6}$	$3 \cdot 10^{-4}$
Amount of work (in million)	1.511	0.908	0.782	0.649
Budget save	0%	42%	48%	57%

The figures in the table above show that we achieve a budget save that approaches 50% for a very low risk of $3 \cdot 10^{-6}$ and even larger saves for higher levels of risk, without noticeable change in the quality of results.

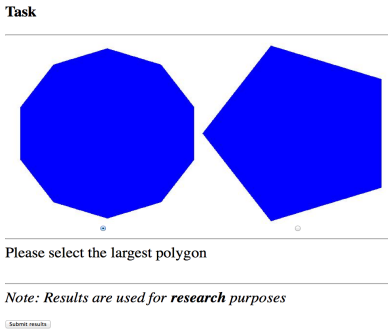

Figure 7: Example of an increased difficulty pairwise comparison task with polygons addressed to real workers of Amazon Mechanical Turk

Experimental study with real crowds

Methodology

Data We tested three types of item sets of cardinality 320. For $k = 5$ and $s = 4$, the algorithm runs in the same way as the example of figure 2, yet, we used quick-sort style endgame instead of exhaustive. The first item set is a set of squares whose edges range from 60 to 380, increasing one pixel at a time. The difference in size among squares is easy to discern, therefore, we consider the overall task of finding the top-5 squares as an easy task. The second itemset is a set of polygons with vertices 4-10 alternately and their area increasing by a constant factor, a unit area. Setting the task as the comparison of the areas of the shapes, we expected the items of the second item set to be harder to compare than the squares. Thus, we consider this dataset as being of medium difficulty. Finally, we created a polygon dataset decreasing the size of the unit area so that the polygons become very difficult to compare. In figure 7 we can see a snapshot of a pairwise comparison task among polygons.

Workers We used the workers of Amazon's Mechanical Turk without imposing any restrictions nor requiring qualification tests. We paid $0.01 per task, which is the lowest rate for a task. The application shuffles the shapes at each round as dictated by the algorithm at every roundtrip, constructs HTML forms with the shapes on the fly, and uploads the forms to a certified server of UC Santa Cruz through SFTP. Uploading to a certified server was necessary to ensure that workers can fetch the data through HTTPS and submit the results without security warning messages from their browser. If a HIT remained unanswered for more than 25 minutes, it was more likely that it would remain so indefinitely, since new tasks would come on the top of the list

of tasks at Mechanical Turk's page making our tasks less visible. Due to this, the application automatically reposts belated HITs after that time, disposing the previous ones.

Error measure The error measure is the same as for the simulations, having the areas of the squares as values, and considering the area difference among consecutive shapes in the baseline ranking to be the unit area for all three datasets.

Algorithms We tested the *Crowd-Top-k* algorithm with quick-sort style endgame. We allowed for a risk of loss of less than 2% by allowing at most 3 items from each partition to proceed to the endgame. First round uses 1 worker per item and second round 3. Using 7 workers for each comparison in the endgame this corresponds to roughly 800 HITs overall, of which 160 were ranking tasks and the rest pairwise comparisons. This corresponds to a cost of approximately 1220 according to our cost metric, though for both tasks we paid the same lowest possible rate and the workers contributed to them with no problem. It is fair though to reason in terms of cost since a hypothetic platform may allow for lower compensation rates. We also tested the algorithm in (Davidson et al. 2013) assuming $logX = 1$ which roughly corresponds to δ around 15%. This requires roughly 800 pairwise HITs when posting 3 comparisons in the upper levels of the X-tree. Finally, we tested its non-randomized version which is essentially the algorithm in (Feige et al. 1994) requiring roughly 1,300 pairwise comparison HITs.

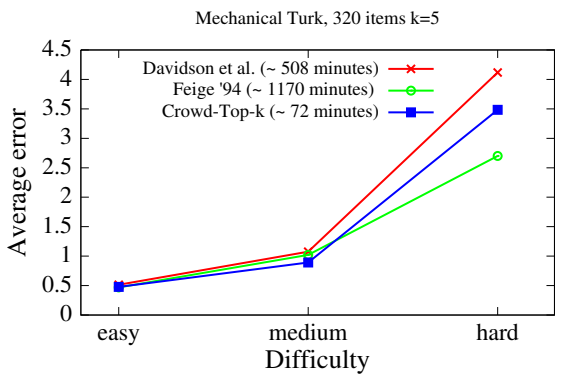

Figure 8: Comparative performance using Amazon's Mechanical Turk crowd

Results

The results in figure 8 demonstate a comparable performance of the three tested methods in terms of quality of results for all three levels of difficulty. What is significantly different is the difference in latencies. The *Crowd-Top-k* has an approximate latency of 13 roundtrips and 72 minutes, while the algorithm of (Davidson et al. 2013) has a latency of 214 roundtrips and 508 minutes (there is only one task in each roundtrip but the total time is significantly higher than the *Crowd-Top-k*'s latency, because of the parallel execution of tasks in each roundtrip of *Crowd-Top-k*). The non-randomized version has a latency of 420 roundtrips and roughly 1170 minutes. The latency of those methods would get prohibitively high for even bigger datasets as it would increase linearly, while the *Crowd-Top-k* algorithm would scale as the latency increases logarithmically.

Conclusions

The techniques we propose make efficient use of available budget and overcome major limitations of prior art. The algorithms have the flexibility to use different methods for solving the 'endgame' problem based on the preferred tradeoffs for latency and quality of results, given a specific budget. They demonstrate high tolerance to random spammers, vandals and errors even for unrealistically high spammer percentages and errors. Applying the randomized approach to very large itemsets, we can reduce the necessary cost drastically, with negligible risk of lowering the quality of results.

Acknowledgement

This research has been supported in part by the NSF Award 1432690.

References

Ailon, N. 2012. An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity. *JMLR* 13(Jan):137–164.

Ciceri, E.; Fraternali, P.; Martinenghi, D.; and Tagliasacchi, M. 2016. Crowdsourcing for top-k query processing over uncertain data. *IEEE Transactions on Knowledge and Data Engineering* 28(1):41–53.

Davidson, S.; Khanna, S.; Milo, T.; and Roy, S. 2013. Using the crowd for top-k and group-by queries. In *ICDT*, 225–236.

Diaconis, P., and Graham, R. L. 1977. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)* 262–268.

Fagin, R.; Kumar, R.; Mahdian, M.; Sivakumar, D.; and Vee, E. 2004. Comparing and aggregating rankings with ties. In *PODS*, 47–58. ACM.

Fagin, R.; Kumar, R.; and Sivakumar, D. 2003. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, 301–312. New York, NY, USA: ACM.

Feige, U.; Raghavan, P.; Peleg, D.; and Upfal, E. 1994. Computing with noisy information. *SIAM J. Comput.* 23(5):1001–1018.

Guo, S.; Parameswaran, A.; and Garcia-Molina, H. 2012. So who won?: Dynamic max discovery with the crowd. In *SIGMOD*, 385–396.

Ipeirotis, P. 2010. Mechanical turk: Now with 40.92% spam. *Behind Enemy Lines blog*.

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

Polychronopoulos, V.; de Alfaro, L.; Davis, J.; Garcia-Molina, H.; and Polyzotis, N. 2013. Human-powered top-k lists. In *WebDB*, 25–30.

Thurstone, L. L. 1927. A law of comparative judgment. *Psychological review* 34(4):273.

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

Venetis, P., and Garcia-Molina, H. 2012b. Quality control for comparison microtasks. In *CrowdKDD*, 15–21. ACM.

Wauthier, F. L.; Jordan, M. I.; and Jojic, N. 2013. Efficient ranking from pairwise comparisons. In *ICML (3)*, 109–117.