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 focus on the problem of obtaining top-k lists of items from larger itemsets, using human workers for doing comparisons among items. An example application is short-listing a large set of college applications using advanced students as workers. 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 against prior art 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.

1 Introduction

We address the problem of obtaining top-k lists of items out of arbitrarily large itemsets using crowdsourcing, a natural and commonly occurring problem. 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.

Previous studies applicable to the top-k problem in the context of crowdsourcing have major limitations. 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. Moreover, the methods cannot tackle top-k lists that are larger than the size of the ranking tasks given to humans. The work in [Marcus *et al.*, 2011] is particularly wasteful in resources. The method studied in [Davidson *et al.*, 2013] invokes the method described in [Feige *et al.*, 1994] to obtain the top-k list for a reduced itemset. 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. Worker tasks are restricted to pairwise comparisons, while in reality human workers can perform tasks containing significantly more than two items. 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]. 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 *et al.*, 2013], in which predicted permutations are more accurate near the top rather than the middle of the sorted list. 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 crowdsourced top-k algorithms that have logarithmic latency and require a number of crowdsourcing tasks which grows linearly with the size of the itemset.
- We propose a budgeting strategy aiming to efficiently address the impact of adversarial users, and 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 with simulations and real crowds. 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. 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.

2 Problem Statement

We have a set of items I and we assume that the items in I can be ranked based on some characteristic of interest. We call this ranking the *baseline ranking*, and we denote 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.

3 A Recursive Crowdsourced Top-k Algorithm

We introduce a novel recursive algorithm, called *Crowd-Top-k* [de Alfaro *et al.*, 2016], for the top-k problem, that decouples the size of the top-k list from the size of the ranking tasks given to humans. It can therefore work for any size of input itemset and top-k list. The method splits the input itemset into partitions of size s and gives each partition to human workers for ranking. After obtaining the rankings from the crowd, the algorithm uses the maximum elements from each partition to form a new set. It then recursively calls itself on this new set. As long as the set has at least $k \cdot s$ items, the algorithm is in the *reduction* phase, obtaining rankings and recursively calling itself on the set of maxima from each partition. When the set has less than $k \cdot s$ items, that is, when it cannot be partitioned further into k or more partitions of size s, the algorithm gets into its *endgame* phase. It uses an *endgame method* (see Section 3.1) to obtain the top-k list of the small set (using exhaustive pairwise comparisons is a feasible option for that, as that set is small). Through comparison inferences, the method obtains the top-k list for each input set of the reduction phase, until it obtains the top-k list of the initial input set.

Example Figure 1 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

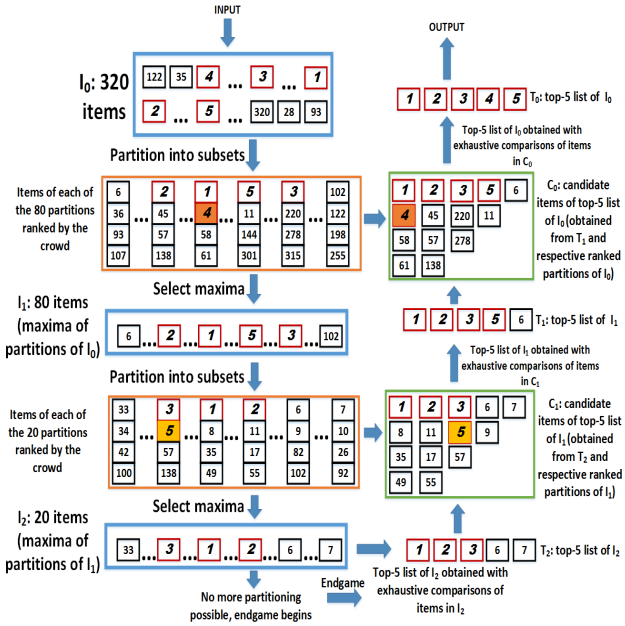


Figure 1: Example of execution of the *Crowd-Top-k* algorithm to obtain the top-5 list of an itemset of 320 items using crowdsourced ranking tasks of size 4

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 constant k, s, the algorithm has a logarithmic latency as measured by the number of recursive calls. The total number of ranking tasks is linear in the size of the input itemset. We omit the proof due to space constraints.

3.1 Endgame Top-k Algorithms

We informally call the methods that obtain top-k lists for small itemsets, endgame algorithms. Algorithm *Crowd-Top-k* is generic with respect to the method that it uses to obtain the top-k list when the endgame begins. 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. Though it is expensive in terms of number of tasks required, its advantage is that it issues all necessary tasks in a single roundtrip. The method described in [Wauthier *et al.*, 2013] called Unbalanced Rank Estimation (URE) which estimates the rank of each item by issuing a limited number of pairwise comparisons in a single roundtrip is another option. We

also implemented a comparisons inference algorithm that issues comparisons in several roundtrips by selecting the most informative comparisons. The problem of selecting the most informative comparisons bears similarities to the 'Next Votes Problem' that Guo et al. studied in [Guo *et al.*, 2012]. Since this problem is hard to solve optimally, we employ a heuristic. Finally, we implemented a top-k variant of the quick-sort algorithm. For small top-k lists the tournament algorithm can be used in the endgame.

3.2 Handling Inaccuracy of Crowds

Our algorithm handles the inaccuracy of crowds through redundancy. We assign each ranking task to multiple workers and obtain one ranking out of the potentially different rankings that workers provide for a particular partition. Producing a single ranking out of multiple rankings over the same set of items is called *rank aggregation*. Our method uses median rank aggregation due to several desired properties [Fagin *et al.*, 2003; 2004]. Inaccuracy of crowds can be due to honest mistakes or spamming. For ranking tasks, spamming can be random, i.e. a user can provide any permutation by chance without doing the work, or adversarial. Adversarial spammers or *vandals* provide inverted or nearly inverted true rankings. We employ the method for adaptive allocation of tasks described in [Polychronopoulos *et al.*, 2013] which can tackle mistakes by workers and random spamming. In settings with significant percentages of adversarial spammers (as can happen in large, unmoderated crowds) those methods are not efficient as they cannot tackle adversarial behavior. Based on an analytic estimate of the impact that adversarial users have on an aggregation, we introduce a budgeting strategy that distributes the available ranking tasks across the recursive calls, in an attempt to battle adversarial behavior. The analysis casts the allocation of the budget to an optimization problem that resembles a non-linear variant of the knapsack problem. We implemented and experimented with this technique under different assumptions for the presence of adversarial spammers in the crowd.

3.3 Randomized Variant of the Crowd-Top-K Algorithm

We propose a randomized variant of the *Crowd-Top-k* algorithm. A key observation that allows us to devise a randomized variant is that the reason we create the new candidate set using the top-k maxima and items from partitions where the top-k maxima belong, is the non-zero probability that more than one items of the top-k list of the input itemset may fall in the same random partition (see also the example we provide in Figure 1). 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*. We formally bound the probability that such a collision occurs at one of the recursive call of the algorithm, using a union bound analysis. 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 randomization which the user of the algorithm deems acceptably low. The method calculates the number of items that we need to consider at every stage of the endgame to maintain the risk below the threshold and creates reduced itemsets during the endgame, reducing the cost of the endgame by accepting a risk of losing items of the top-k list due to random collisions.

4 Experimental Study

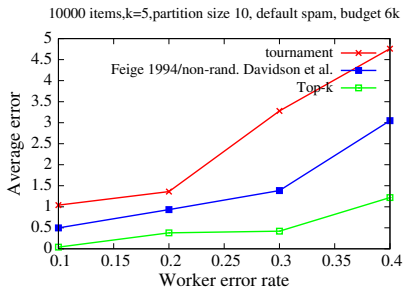
4.1 Simulations

In the simulation setting, human workers can be either honest workers or spammers, where spammers can be either random or adversarial. Each item i has a value $V(i)$ that represents its quality. The error measure we use is: $\epsilon_r = \frac{\Sigma_{i=1}^{k} V(\beta_i) - \Sigma_{i=1}^{k} V(e_i)}{V(\beta_1) - V(\beta_k) + 1}$ 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. 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. We restrict our reporting to the equi-distant assumption, where the qualities are integer numbers and neighboring items in the baseline ranking have value distance of one, but other assumptions (e.g. the Gaussian assumption for values of items) lead to similar or superior results. 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]. Default random spam is 25% and default error rate is 20%.

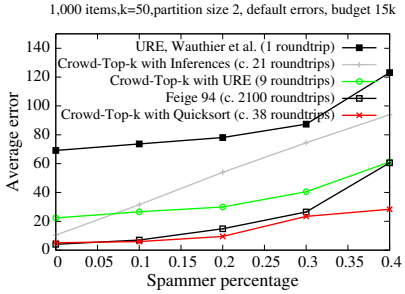
Figure 2(a) shows the performance against the tournament approach for a small top-k list (k= 5 for ranking tasks of size $s = 10$) The results confirm the superiority of the recursive approach over the tournament algorithm. The number of roundtrips fluctuates for the two methods, ranging from 11 to 30, 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 *Crowd-Top-k algorithm*.

Figure 2(b) shows comparative results of five methods. 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.

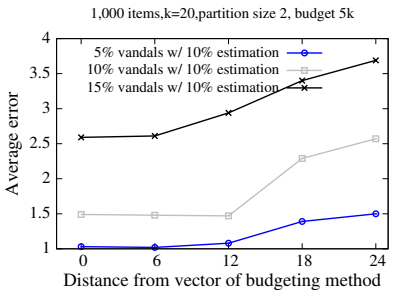
Figure 2(c) demonstrates the error for three levels of vandals in the crowd, at 5% and 10% and 15%. We evaluate the *Crowd-Top-k* algorithm with URE for the endgame using several budget arrangements that maximally satisfy the budget constraint. The x-axis represents the Manhattan distance of the budget arragement to the arrangement that our budgeting method returns as optimal arrangement. The results confirm the relevance of the analysis, as the error increases with increasing distance from the budget distribution of the budgeting method.



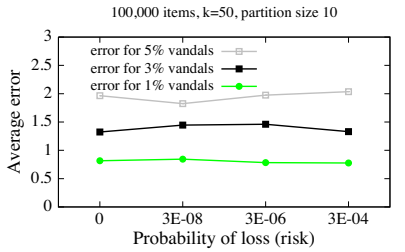
(a) Comparative performance against the tournament algorithm for small top-k lists



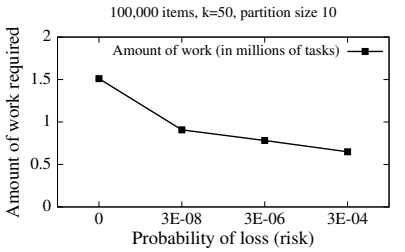
(b) Comparative performance of methods for large top-k lists



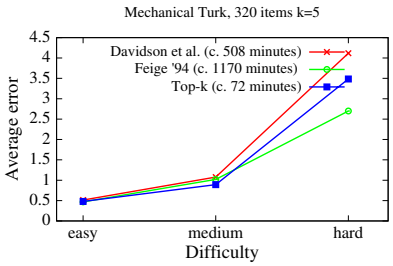
(c) Efficiency of the budgeting method



(d) Performance of randomized variant for increasing risk threshold



(e) Decreasing budget requirements with increasing risk threshold



(f) Comparative performance using Mechanical Turk crowd

Figure 2: Results of experimental study

Figure 2(d) demonstrates the error for the randomized variant as the risk increases. 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 of the endgame 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. We need significantly lower budget to achieve comparable or even superior quality. We can see the use of budget, expressed in required amount of work, in the Figure 2(e). We achieve a budget save that approaches 57% for a very low risk of $3 \cdot 10^{-4}$, without noticeable change in the quality of results.

4.2 Mechanical Turk Workers

For the experiments on Mechanical Turk, the task of workers is to rank polygons according to their size. We have three sets of polygons with different levels of difficulty (smaller differences in the area of the polygons makes the task of ranking more difficult), which we consider as easy, medium and hard. 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 1300 pairwise comparison HITs. The results in Figure 2(f) demonstrate a comparable performance of the three tested methods in terms of quality of results for all three levels of difficulty. The difference in latencies is significant. 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 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.

Acknowledgements

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

References

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

[Ciceri *et al.*, 2016] Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Crowdsourcing for top-k query processing over uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):41–53, 2016.

[Davidson *et al.*, 2013] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

[de Alfaro *et al.*, 2016] Luca de Alfaro, Vassilis Polychronopoulos, and Neoklis Polyzotis. Efficient techniques for crowdsourced top-k lists. In *Proceedings of the 4th AAAI Conference on Human Computation and Crowdsourcing (HCOMP)*, pages 22–31. AAAI, 2016.

[Fagin *et al.*, 2003] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, pages 301–312, New York, NY, USA, 2003. ACM.

[Fagin *et al.*, 2004] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *PODS*, pages 47–58. ACM, 2004.

[Feige *et al.*, 1994] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, October 1994.

[Guo *et al.*, 2012] Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. So who won?: Dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396, 2012.

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

[Marcus *et al.*, 2011] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. *VLDB*, 5(1):13–24, September 2011.

[Polychronopoulos *et al.*, 2013] Vassilis Polychronopoulos, Luca de Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. Human-powered top-k lists. In *WebDB*, pages 25–30, 2013.

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

[Venetis and Garcia-Molina, 2012a] Petros Venetis and Hector Garcia-Molina. Dynamic max algorithms in crowdsourcing environments. Technical report, Stanford University, August 2012.

[Venetis and Garcia-Molina, 2012b] Petros Venetis and Hector Garcia-Molina. Quality control for comparison microtasks. In *CrowdKDD*, pages 15–21. ACM, 2012.

[Wauthier *et al.*, 2013] Fabian L. Wauthier, Michael I. Jordan, and Nebojsa Jojic. Efficient ranking from pairwise comparisons. In *ICML (3)*, pages 109–117, 2013.