Online Top-K Selection in Crowdsourcing Environments

Shenshen Liang
Technology and Information Management, UC Santa Cruz
Santa Cruz, California
sliang@soe.ucsc.edu

Luca de Alfaro
Computer Science, UC Santa Cruz
Santa Cruz, California
luca@ucsc.edu

ABSTRACT

Identifying the top-K items in a set of items is a problem that finds applications in many areas, such as recommender systems, social review platforms, online contests, web search, and more. Crowdsourcing provides an effective way to collect input for such tasks with low costs and has attracted significant attention.

We consider top-K problems in which the focus consists in selecting the set of top-K elements, regardless of their internal ordering. Past algorithms for top-K problems were generally based on a global sorting, which perform unnecessary work sorting elements that are all selected or all rejected. The exceptions are specialized crowdsourcing algorithms that have been proposed especially for top-K selection; these algorithms, however, require a fixed amount of work to be performed, and produce no useful intermediate answer.

We propose here a dynamic top-K selection algorithm that uses crowdsourced comparisons to progressively classify items into selected in top-K, and rejected. As the comparisons proceed, more and more elements are accepted; intermediate results can be provided at any time by returning the already-accepted items along with the best among the ones that are still unclassified. We show that the algorithm we develop is efficient and robust with respect to comparison noise. We illustrate the performance of algorithm both analytically and experimentally, and show that our algorithm can achieve comparable precision in the selection of top-K items with less crowdsourcing work than previous algorithms.

CCS CONCEPTS

• **Information systems → Learning to rank**;

KEYWORDS

top-K selection, Crowdsourcing, Online algorithm, Pairwise comparison.

1 INTRODUCTION

Selecting top-K items from large itemsets is a common task: it occurs in college admissions, candidate selections, social review platforms, online contests, and so on. Broadly, there are two approaches to the top-K selection problem. One approach consists in determining a global ranking, and then returning the top-K elements of such a ranking. This approach can avail itself of a long line of work on crowdsourced ranking, starting with now classical approaches such as the BTL model [5], ELO ranking [18], and Trueskill [22]. This approach can be inefficient for the task of top-K selection, as it determines a full ranking even among items that are all selected, and all rejected. Nevertheless, the approach often has the advantage of being able to offer partial results: as these algorithms maintain and refine a global ranking (in various ways), at any point during the crowdsourcing work, they can provide a best estimate of the ranking, and thus, of the top-K elements.

Approaches that focus on selecting the top-K elements, rather than building a global ranking, have been proposed in [12, 14, 16, 31, 35] among others. As expected, these can be far more efficient than global-ranking algorithms, as the crowdsourcing work is aimed at separating the top-K elements from the rest, rather than obtaining a full ranking. However, these specialized top-K algorithms require the completion of a fixed amount of work, and do not provide useful partial results if less work is performed. For instance, if the algorithms of [16, 31] are interrupted, one is left with multiple lists of "best-among-M" elements, for comparison batches of size M. Short of randomly selecting K of these best-among-M elements, there is no obvious way of compiling a global top-K result if only part of the required work for the algorithm is performed. Yet, partial results are often useful. For example, in candiate selection, one may wish to extend offers to candidates as soon as they are determined to be in the top-K set. If the crowdsourcing work takes longer than expected, if a deadline makes it impossible to carry it to the end, or if the funds for it are exhausted, having a usable partial result is the difference between failure and (partial) success.

Here, we propose a top-K selection algorithm that dynamically refines a classification of the items into accepted (in the top-K) and rejected (not in top-K). Initially all items are unclassified; for each item, the algorithm maintains an estimated quality, and an estimated uncertainty on the quality. Given an error tolerance $\epsilon > 0$, the algorithm creates a batch of comparisons, and as the results from the crowd come in, the algorithm uses them to refine the classification, promoting to accepting all the items that fall into the top-K with probability at least $1-\epsilon$, and demoting to rejecting all items that have probability at least $1-\epsilon$ of *not* belonging to the top-K elements. The process is repeated until the top-K elements are accepted. Partial results can be obtained at any time by returning the accepted elements, along with the unclassified elements that have largest estimated quality. We show that not only can our algorithm provide partial results; it is also more efficient than previous top-K algorithms. The efficiency stems from the fact that items are classified into accepted or rejected as soon as enough evidence is accumulated, rather than after a prescribed amount of comparisons are performed. Hence, the ability of our algorithm to provide partial results, and its superior efficiency, both stem from the same underlying mechanism.

The contributions of this paper can be summarized as follows:

- We present an online algorithm, named BetaTopK, for top-K item selection based on the dynamic classification of items

ICCDE '20, January 4–6, 2020, Sanya, China

ACM ISBN 978-1-4503-7673-0/20/01…$15.00
https://doi.org/10.1145/3379247.3379273

into top-K, and rejected, items. The algorithm can at any time provide a best-effort selection of the top-K elements.
- We show that, for a fixed error tolerance $\epsilon > 0$ and fixed crowd error probability, and for any top K/N percent, the expected number of required comparisons is $O(N \log N)$, where N is the total number of elements. The worst-case occurs for $K = N/2$.
- We propose several optimization techniques to improve BetaTopK.
- We evaluate the performance of our algorithm analytically and experimentally, from the perspectives of selection loss and number of required comparisons, and we show that it improves on the state of the art. Both global ranking and top-K selection approaches are used as benchmarks.

2 RELATED WORK

Identifying top-K items is a critical problem in many areas, such as recommender systems, social review platforms, online contests, candidate searches and so on. Crowdsourcing provides an efficient way to collect input with relatively low costs, and it has been applied extensively in learning tasks, such as ranking, classifying, labeling, and more [4, 17, 27].

Extensive research has been done on obtaining a global ranking from crowdsourced data. Among the foundational work, we recall here the maximum likelihood estimation (MLE) model [20, 23], which is one of the most commonly adopted schemas for global ranking. Another approach is the Bradley-Terry-Luce (BTL) model [5, 26], which has provided the foundations of many crowdsourcing works, including the classical Elo rating [18]. For example, RankCentrality by Negahban et. al extended the BTL model and used random walk strategy on pairwise-choice Markov chain [30]. It is one of the most representative works in the family of spectral-based ranking paradigm, along with [28, 29, 32], to name a few. Another notable work stemming from BTL model is TrueSkill by Microsoft [22]. It improved Elo ranking [18] by constructing a factor graph and using approximate message passing, dramatically reducing the amount of input required to converge. Much work followed, including the CrowdBT model with BTL by Chen et al. [9], which took the quality of workers into consideration. Wauthier et. al [36] suggested a weighted counting algorithm on edges to achieve an approximate overall ranking with noisy input. The work in [1, 2, 6–8, 21, 24] considered the scenario in which input comparisons can be selected adaptively, and proposed different active learning strategies to improve a global ranking.

While comprehensive work exists for the problem of global ranking with crowd input, a direct application of those methods to a top-K problem increases the complexity unnecessarily, since determining a global ordering is more expensive than determining the set of top-K items.

Approaches that are tailored to top-K problem include [16, 31], which generalized the tournament approach into a multi-round ranking and selection process for determining the top-K elements. The algorithm by Davidson et al. [15] also adopted the tournament approach and utilized a heap-based method to get the top-K list. These three works can cope with errors in crowd comparisons, but they require the completion of a fixed amount of work, namely the ranking-and-selection rounds, before the final result becomes available. Eriksson proposed a top-K algorithm [19] by considering comparisons as graph edges. It performed a 'reverse' depth first search and eliminated the items with path length larger than K. However, this algorithm can return the items far from the top-K items if not all comparisons among the items are observed. Chen and Suh proposed a spectral MLE approach [11] that recovered the top-K items with high probability, with the strict assumption that the comparisons follow the BTL model. Jiang et al. [25] and Suh et al. [34] suggested top-K methods by extending the spectral MLE algorithm with different restrictions respectively, with the same BTL assumption. As the BTL model does not account for the possibility of users purposely lying, the BTL model and its variations fail to provide accurate estimations in practice [3, 13]. A Borda-count based algorithm was proposed in [33] to achieve the $O(N)$ bound of input comparisons, with the assumption of a Binomial distribution for the number of comparisons of each pair. Our algorithm is related, except it performs active learning, using a probability model to decide when to truncate the accumulation of Borda counts and discard or promote an item, while offering precise guarantees on the correctness probability of the top-K set. Chen et al. studied bounds for the amount of comparison input that is needed, under the setting in which comparisons can be requested uniformly for all items [10]. In contrast, our algorithm is an on-line one, where comparison requests are dynamically generated for the most needed items. In [12], active learning strategies for top-K problems were discussed. However, it assumed the probabilities of preferences between items are given, which is not realistic in practice.

3 ONLINE TOP-K SELECTION ALGORITHM

3.1 Definitions and Problem Settings

Our goal is to discover the top-K items from an itemset. We consider an itemset I, where each item $i \in I$ has an intrinsic quality r_i. For $K \leq |I|$, the top-K problem consists in selecting the subset $T_K \subseteq I$ of elements such that $|T_K| = K$, and such that $\sum_{i \in T_K} r_i$ is maximized. For a candidate top-K selection $U \subseteq I$, $|U| = K$, we do not measure the quality of U by its elementwise difference from T_K, as this would provide only a coarse measure of selection quality. Rather, we define the *selection loss* $Loss(U)$ as the fraction of quality we fail to capture by selecting U rather than T_k:

$$Loss(U) = \frac{\sum_{i \in T_k} r_i - \sum_{i \in U} r_i}{\sum_{i \in T_k} r_i} \tag{1}$$

This notion of loss encodes the general goal of top-K selection: with a constraint on the number of items that can be selected, the goal is to amass as large a total quality as possible. Such measurement finds its application in many real-world scenarios. For example, a graduate program plans to admit 10 students, but one of the top-10 applicants is not selected by mistake. In this case, selecting an applicant whose ranking is close to top-10 will be better than a bottom-ranked one. The loss defined in Equation (1) captures the degree of loss among different results, while an elementwise measurement fails to do so.

3.2 Algorithm foundations

The core idea of our algorithm consists in modeling the quality of an item via a Beta distribution, as the two parameters of Beta distribution account for the number of wins and losses of an item in comparing with others. If comparison outcomes are truthful, that is, if higher-ranked items win when they are compared against lower-ranked ones, a Beta distribution $\text{Beta}(\alpha, \beta)(x)$ represents the a-posterior probability that an item ranks at a fraction x of the total rank, given that $\alpha - 1$ comparisons were won, and $\beta - 1$ lost. Thus, a Beta distribution

accounts both for the estimated position of an item in the ranking, and for the uncertainty with which the rank is known. The algorithm accumulates evidence on items until the items can be accepted in the top-K set, or discarded, with a specified amount of confidence.

If there are no comparison errors, the fractional rank of an item after w wins and l losses has an a-posteriori distribution Beta$(w+1, l+1)(x)$. If crowd comparisons are only correct with an average probability of q, then the a-posteriori fractional rank distribution is Beta$(w+1, l+1)\Big(xq + (1-x)(1-q)\Big)$. Thus, let

$$c(w, l; q)(x) = CDF_{\text{Beta}}\Big(xq + (1-x)(1-q); w+1, l+1\Big),$$

where $CDF_{\text{Beta}}(x; \alpha, \beta)$ is the cumulative distribution function of Beta distribution, with parameters of α, β at x. Given a probability bound $\epsilon > 0$ of making a wrong decision, we can promote the item to the top-K set when

$$c(w, l; q)\left(\frac{N-K}{N}\right) < \epsilon, \tag{2}$$

and we can eliminate the item when

$$1 - c(w, l; q)\left(\frac{N-K}{N}\right) < \epsilon. \tag{3}$$

Equation (2) and (3) suggest that a non top-K item can be selected into top-K list with a probability of at most ϵ, and likewise a top-K item can be eliminated with the same probability bound. Also, with Equation (2), the probability of an item in top-K can be calculated any time before BetaTopK algorithm completes. As a result, the top-K list can be generated anytime by selecting the K items with highest probabilities. In this way, the online property on our algorithm is achieved.

We propose an initial algorithm, which determines the top-K list by eliminating items until only K items remain. Later, we will propose an optimized algorithm that performs item promotion and elimination concurrently.

3.3 The BetaTopK Algorithm

The BetaTopK algorithm is detailed in Algorithm 1. In this algorithm, the top-K list is achieved by eliminating unlikely candidates in iterations, and returning the remaining itemset. The notations of the algorithm are:

- I: the itemset, in which the number of remaining items reduces with the progress of algorithm.
- K: the number of top items to be selected.
- h: the number of comparisons an item receives in one iteration.
- ϵ: the probability of eliminating a top-K item erroneously.
- E_K: the top-K result set.
- T: an itemset that holds the candidates for elimination.

At the beginning of an iteration, each item starts with zero wins and losses. It is compared with h random opponents, and the number of wins and losses are recorded, as illustrated in the PerformComparisons procedure. The algorithm then eliminates the items with probabilities of being top-K smaller than ϵ. Such iteration repeats until K items remain. The remaining items are returned as the output of the algorithm.

At any time before the BetaTopK algorithm completes, a best-effort top-K list can be obtained via Algorithm 2; this algorithm calculates the probability of being top-K of each item in the remaining itemset, and returns the K items with the largest probabilities.

Algorithm 1 BetaTopK Algorithm

Input: I, K, h, ϵ
Output: E_K

while $|I| > K$ **do**
  PerformComparisons(I, h)
  $e_i \leftarrow c(w_i, l_i; q)\left(\frac{|I|-K}{|I|}\right)$
  // Demotion: identify items to be eliminated.
  $T \leftarrow \{i \in I \mid 1 - e_i < \epsilon\}$ // See Equation (3).
  if $|T| > |I| - K$ **then**
    $T \leftarrow |I| - K$ items in T with largest e_i
  $I \leftarrow I \setminus T$
$E_K \leftarrow I$

procedure PerformComparisons(I, h):
  for all $i \in I$ **do**
    $w_i \leftarrow 0, l_i \leftarrow 0$
  $c \leftarrow 0$
  while $c < \frac{h}{2}$ **do**
    Shuffle I
    for $i = 1$ to $|I|$ **do**
      Get items u, v at positions i and $(i+1)\%|I|$
      Ask the crowd to compare u, v
      Update w_u, l_u, w_v, l_v
    $c \leftarrow c + 1$

Algorithm 2 Online retrieval of top-K

Input: I, K
Output: E_K

$e_i \leftarrow c(w_i, l_i; q)\left(\frac{|I|-K}{|I|}\right)$
$E_K \leftarrow K$ items with smallest e_i

Theorem 3.1. *To select any percentage of top items, the expected number of comparisons required in BetaTopK for an itemset of N items is $O(N \log N)$.*

Proof. We use the following notations in the proof:

- q: the average probability of correct comparisons by crowd.
- i: the true rank of an item.
- t: the number of wins an item has to receive minimally, in order to not be eliminated, i.e., to survive.
- P_i^d: the probability of item i being eliminated in one iteration.
- p_i : the probability of item i winning a randomly selected opponent.
- x_i: the total number of wins item i receives in one iteration.

In particular, the relationship between p_i and q is:

$$p_i = \frac{(N-i)q + (i-1)(1-q)}{N-1} \tag{4}$$

because for an item ranks at position i, $i-1$ items are superior to it and $N-i$ items are inferior to it. With crowd's average probability of comparing correctly being q, an item wins with an expectation of $(i-1)(1-q)$ times when being compared with the $i-1$ items that are superior to it, and wins an expectation of $(N-i)q$ times when comparing to the $N-i$ items that are inferior to it.

Based on Equation (4), we can derive the relationship between p_i and $p_{(N+1-i)}$ as follows:

$p_i + p_{(N+1-i)} = \frac{(N-i)q+(i-1)(1-q)}{N-1} + \frac{(N-(N+1-i))q+(N+1-i-1)(1-q)}{N-1}$
$= \frac{(N-1)q+(N-1)(1-q)}{N-1} = q + 1 - q = 1$

To sum up:

$$p_i + p_{(N+1-i)} = 1 \tag{5}$$

An item ranking at position i is eliminated in an iteration with a probability of P_i^d. We can express this probability as a Bernoulli variable since the outcome of elimination is binary. By characteristics of Bernoulli distribution, the expected number of items being eliminated in one iteration is:

$$\sum_{i=1}^{N} P_i^d \tag{6}$$

We can prove Theorem 3.1 if the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N. The rationale behind is that, the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ being independent of N means a fraction of N items is eliminated in one iteration, regardless of the value of N. Equivalently, it means to reduce the remaining number of items to a constant value, $\log N$ iterations are required. Note that in one iteration, all items receive h comparisons respectively, making the complexity of the number of comparisons in one iteration $O(N)$. Consequently, the expected number of comparisons required in BetaTopK is $O(N \log N)$.

We consider $K \leq \frac{N}{2}$ in the following proof. However, the proof remains valid when $K > \frac{N}{2}$, because in that situation, the top-K problem is equivalent to the problem of selecting the bottom-$(N-K)$ items, so the same proof applies.

The proof focuses on showing that the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N herein. For an item that ranks at posision i, P_i^d can be calculated as: $P_i^d = Pr(x_i < t)$. Note that any comparison that item i receives is a Bernoulli trial with a winning probability of p_i. As a result, the total number of wins that item i receives in one iteration with h comparisons, i.e., x_i, is a random variable that follows a Binomial distribution with parameters h and p_i. With x_i expressed as a Binomial random variable, $\sum_{i=1}^{N} P_i^d$ can be calculated by:

$$\begin{aligned}\sum_{i=1}^{N} P_i^d = \sum_{i=1}^{N} Pr(x_i < t) &= \sum_{i=1}^{N}\sum_{j=0}^{t-1} Pr(x_i = j) \\ &= \sum_{i=1}^{N}\sum_{j=0}^{t-1} Binomial(j; h, p_i)\end{aligned} \tag{7}$$

where $Binomial(k; n, p)$ is the probability mass function of Binomial distribution: $Binomial(k; n, p) = Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$.

By the nature of Binomial distribution, since $p_i + p_{(N+1-i)} = 1$, we have:

$$Binomial(j; h, p_{(N+1-i)}) = Binomial(h - j; h, p_i) \tag{8}$$

With Equation (8), Equation (7) can be calculated with the top half of all items:

$$\begin{aligned}&\sum_{i=1}^{N}\sum_{j=0}^{t-1} Binomial(j; h, p_i) = \\ &\sum_{i=1}^{\frac{N}{2}}\sum_{j=0}^{t-1} \Big\{ Binomial(j; h, p_i) + Binomial(h - j; h, p_i) \Big\}\end{aligned} \tag{9}$$

By characteristics of Bernoulli distribution, $\sum_{j=0}^{t-1} \Big\{ Binomial(j; h, p_i) + Binomial(h-j; h, p_i) \Big\}$ is the total area of two sides of a Binomial probability mass function, and its value ranges from 0 to 1, i.e.:

$$0 \leq \sum_{j=0}^{t-1} \Big\{ Binomial(j; h, p_i) + Binomial(h - j; h, p_i) \Big\} \leq 1 \tag{10}$$

Based on information theory, $\sum_{j=0}^{t-1} \Big\{ Binomial(j; h, p_i) + Binomial(h-j; h, p_i) \Big\}$ is minimized when $p_i = 0.5$. Consequently, Equation (9) has a lower bound as follows:

$$\begin{aligned}&\sum_{i=1}^{\frac{N}{2}}\sum_{j=0}^{t-1} \Big\{ Binomial(j; h, p_i) + Binomial(h - j; h, p_i) \Big\} \geq \\ &\frac{N}{2} \cdot \sum_{j=0}^{t-1} \Big\{ Binomial(j; h, 0.5) + Binomial(h - j; h, 0.5) \Big\}\end{aligned} \tag{11}$$

With Equation (10) and (11), we can derive the lower bound of the expected number of items being eliminated in one iteration, i.e. $\sum_{i=1}^{N} P_i^d$ below:

$$\sum_{i=1}^{N} P_i^d \geq N \cdot \frac{1}{2} \cdot \sum_{j=0}^{t-1} \Big\{ Binomial(j; h, 0.5) + Binomial(h - j; h, 0.5) \Big\} \tag{12}$$

where $\frac{1}{2} \cdot \sum_{j=0}^{t-1} \Big\{ Binomial(j; h, 0.5) + Binomial(h-j; h, 0.5) \Big\}$ is a value within the range of $[0, \frac{1}{2}]$.

If we can show that the variable t is independent of N, by Equation (12), we can assert that the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N.

As aforementioned, t is the number of minimal wins an item has to receive to survive, and an item will be eliminated when the area of Beta distribution to the right of threshold, i.e., $\Big(1 - \frac{K}{N}\Big)q + \frac{K}{N}(1-q)$, is smaller than ϵ. In other words, an item will be eliminated if the area of Beta distribution to the left of the threshold is greater than $1 - \epsilon$. With h comparisons, t can be calculated as:

$$t = \min_t \left\{ CDF_{Beta}\left(\Big(1 - \frac{K}{N}\Big)q + \frac{K}{N}(1-q); t+1, h-t+1 \right) \leq 1 - \epsilon \right\} \tag{13}$$

and $t >= 0$.

From above equation, it is apparent that the value of t depends on the percentage of K out of N, i.e., $\frac{K}{N}$, but is independent of N.

With Equation (13) showing that t is independent of N, we can conclude that the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N. As a result, Theorem 3.1 is proved.

■

3.4 Optimized BetaTopK Algorithm

We present here an optimized version of the BetaTopK Algorithm 1, which improves the initial BetaTopK algorithm in two aspects. First, it adds a promotion stage to each iteration, which selects items with probabilities of being in top-K larger than $1 - \epsilon$. As a result, the promotion stage can identify top-K items with a bounded error of ϵ. Second, instead of resetting all wins and losses at the beginning of each iteration, the optimized BetaTopK keeps the comparisons from

previous iterations as long as the comparisons are performed among the remaining items. In this way, the algorithm can converge faster due to better utilization of historical information.

The optimized algorithm is illustrated in Algorithm 3. In the algorithm, we use the previous notations with the following additions:

- K_{gap}: the number of top-K items yet to be discovered.
- T: an itemset that holds temporary results. It is used for different purposes at selection and elimination stages.

The reason for establishing a temporary itemset T is that, if the number of top-K candidates retrieved in an iteration exceeds what the algorithm expects, or if the number of elimination candidates exceeds what the algorithm targets at, the algorithm should only select or eliminate a subset of the items in T. Keeping the temporary itemset allows the algorithm to further calculate the targeted subset.

Algorithm 3 Optimized BetaTopK Algorithm

Input: I, K, h, ϵ
Output: E_K

$E_K \leftarrow \emptyset, K_{gap} \leftarrow K$
while $|I| > K_{gap}$ and $|E_K| < K$ **do**
  PerformComparisonsCumulative(I, h)
  $e_i \leftarrow c(w_i, l_i; q)\left(\frac{|I|-K_{gap}}{|I|}\right)$
  // Promotion: identify top-K items.
  $T \leftarrow \{i \in I \mid e_i < \epsilon\}$ // See Equation (2).
  if $|T| > K_{gap}$ **then**
    $T \leftarrow K_{gap}$ items in T with smallest e_i
  $E_K \leftarrow E_K \cup T$
  // Demotion: identify items to be eliminated.
  if $|E_K| < K$ **then**
    $T \leftarrow \{i \in I \mid 1 - e_i < \epsilon\}$ // See Equation (3).
    if $|T| > |I| - K_{gap}$ **then**
      $T \leftarrow |I| - K_{gap}$ items in T with largest e_i
  $I \leftarrow I \setminus E_K, I \leftarrow I \setminus T, K_{gap} \leftarrow K - |E_K|$
if $|E_K| < K$ **then**
  $E_K \leftarrow E_K \cup I$

procedure PerformComparisonsCumulative(I, h):
  for all $i \in I$ **do**
    $w_i \leftarrow$ number of wins item i obtains, from the comparisons with other remaining items in I
    $l_i \leftarrow$ number of losses item i receives, from the comparisons with other remaining items in I
  $c \leftarrow 0$
  while $c < \frac{h}{2}$ **do**
    Shuffle I
    for $i = 1$ to $|I|$ **do**
      Get items u, v at positions i and $(i+1)\%|I|$
      Ask the crowd to compare u, v
      Update w_u, l_u, w_v, l_v
    $c \leftarrow c + 1$

4 EVALUATION

We evaluate the performance of BetaTopK from two aspects: selection loss, and number of input comparisons required to complete the selection task.

We conduct experiments with simulated crowdsourcing data as it can provide a precise evaluation of the loss, since we know the items' true qualities. The items' true qualities, or scores, are sampled from a Gaussian distribution. The crowd's perception on each item is expressed as a Gaussian distribution as well. The mean of the latter Gaussian distribution represents the crowd's average perception on the item, and it is set to the item's true quality. The crowd's uncertainty about the item is expressed as the variance of this Gaussian distribution, and is sampled uniformly.

Every time an algorithm requests a comparison from crowd, we simulate it via a two-step process. First, we determine the true winner, that is, the winner as perceived by a honest (or truthful) crowd worker. To this end, we sample the perception distributions of the two items; the item with the larger sample value is the true winner. Second, we return the true winner with probability q, and the true loser with probability $1 - q$; this models the crowd accuracy q. In real-world applications, q can be determined by the average rate of correct answers from crowd empirically.

In the evaluations, we repeat every experiment 20 times, and we report the average and confidence interval of the results.

4.1 Loss Evaluation

In this section, we evaluate our algorithm by comparing its loss with other existing algorithms. The loss defined in (1) is used for the comparison.

A experimental evaluation of top-K ranking and selection algorithms appeared in [37]; the work showed that TrueSkill, a global ranking algorithm, performs excellently in top-K problems among the existing algorithms. As a result, it is used to compare with our algorithm. TrueSkill estimates the qualities of items with any incoming comparisons, and uses the qualities to update the ranking constantly. It is an online algorithm and does not have a definite number of comparisons for termination. To compare our algorithms to TrueSkill, we feed to the algorithms itemsets with the same size and characteristics, and we record the loss at regular intervals along the working of the (online) algorithms. In our experiments, 100 items are compared against each other, with the goal of selecting the top 10 items. Each algorithm takes a total of 2000 comparisons, and the losses are recorded in every 200 comparisons. The hyper-parameter h of BetaTopK is set to 4 in the experiments.

Figure 1 shows the losses of algorithms in two separate settings. The left side of the figure shows the losses with the assumption of a perfectly truthful crowd, i.e., $q = 1.0$. The right side shows the losses with a more realistic assumption, where the crowd can make mistakes and judge incorrectly, with the average correctness q being 0.8. In Figure 1, the x-axis is the number of pairwise comparisons performed, and the y-axis is the average loss. The error bars indicate the 95% confidence intervals.

The results in Figure 1 demonstrate that BetaTopK achieves much lower selection loss than TrueSkill algorithm, when they consume the same number of comparisons. In both settings, the differences between two algorithms become evident after about 400 comparisons, revealing the fast converge characteristic of BetaTopK. It can be observed that the 95% confidence intervals between TrueSkill and BetaTopK rarely overlap, illustrating that our algorithm is able to consistently obtain substantial improvement. Overall, the results demonstrate that the proposed algorithm can achieve significant loss

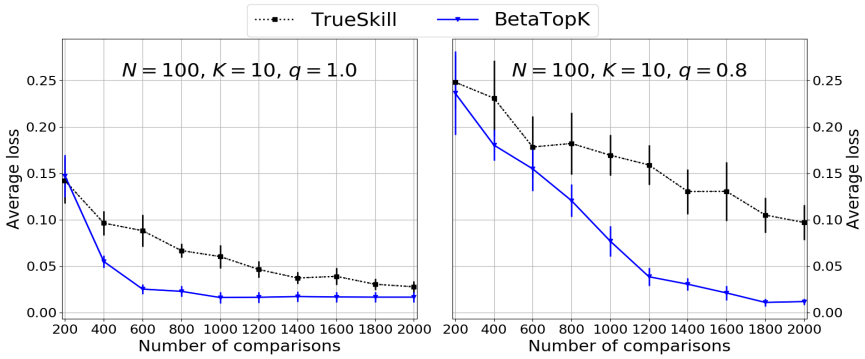

Figure 1: Loss comparisons

reduction and return high quality top-K results comparing to one of the best-performed top-K algorithms.

4.2 Evaluation: Number of Comparisons

We now analyze the number of comparisons required for algorithm termination, by comparing our algorithm with the one proposed in [16]. The algorithm in [16] is a recursive offline algorithm that selects the top-K items by dividing the ranking task into reduction and endgame phases. In the reduction phase, the itemset is partitioned into small sets that can be sent out for ranking by crowd workers; the items that end up in top position in such rankings are used to construct a reduced itemset. The process continues, recursively reducing the itemset size, until the complete set of items can be sorted. In the endgame phase, the complete sorting of the reduced itemset is used to reconstruct a top-K selection of the original itemset, following the recursion backwards. We name the algorithm in [16] RecurTopK, due to its recursive structure. RecurTopK needs a pre-established number of comparisons before the top-K set is constructed. Here, we compare the number of crowdsourced comparisons required by RecurTopK, with the number required by the BetaTopK algorithm we introduced in this paper.

4.2.1 Complexity Analysis. The number of comparisons required by RecurTopK can be derived analytically. Due the the space limit, we do not present the analytical details and summarize here the results. In the following, let:

- s: size of a ranking task in the reduction phase, i.e., the number of elements that RecurTopK can send to each crowd worker for comparison;
- η: probability of an item being incorrectly ranked in a recursive call.
- $CDF_{Binomial}(k; n, p)$: the cumulative distribution function of Binomial distribution. $CDF_{Binomial}(k; n, p) = Pr(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i}$, where X is a random variable.

In a perfect scenario where the crowd is perfectly truthful, i.e., $q = 1.0$, we have:

Reduction phase: $\frac{N}{s-1} \cdot s \log s$.

Endgame phase, where K < s : $\left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot \left(\frac{(K+1)\cdot K}{2}\right) \log\left(\frac{(K+1)\cdot K}{2}\right)$.

Endgame phase, where K ≥ s : $\left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot sK \log(sK)$.

In scenarios where the crowd is inaccurate, i.e. $q < 1.0$, we get:

Reduction phase: $R \cdot \frac{N}{s-1} \cdot s \log s$, where

$$R = \min_R \left\{ CDF_{Binomial}\left(\left\lfloor \frac{R}{2} \right\rfloor; R, q\right) \leq \frac{\eta}{\log s} \right\}.$$

Endgame phase, where K < s :

$$U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot \left(\frac{(K+1)\cdot K}{2}\right) \log\left(\frac{(K+1)\cdot K}{2}\right),$$

$$\text{where } U = \min_U \left\{ CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q\right) \leq \frac{\eta}{\log\left(\frac{(K+1)\cdot K}{2}\right)} \right\}.$$

Endgame phase, where K ≥ s : $U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot sK \log(sK)$,

$$\text{where } U = \min_U \left\{ CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q\right) \leq \frac{\eta}{\log(sK)} \right\}.$$

In the equations above, we relate the probability η of error in a recursive call to the error bound ϵ of BetaTopK via:

$$\eta = \frac{E\epsilon}{2(D+1)} \tag{14}$$

where $D = \lceil \log_s(\frac{N}{K}) \rceil - 1$ is the number of recursive calls required by the RecurTopK algorithm.

For BetaTopK, we perform the evaluation both analytically and experimentally. In the analytical evaluation, we calculate the expected number of comparisons required by BetaTopK to complete. In the experimental evaluation, we demonstrate the average number of comparisons required by BetaTopK.

4.2.2 Analytical Comparison. Table 1 shows the number of crowdsourced comparisons required by RecurTopK(denoted as Recur) and BetaTopK(denoted as Beta) in Algorithm 1 analytically, with various K, N and q settings. The algorithm used here does not adopt the optimization techniques described in Section 3.4. Also, the analytical results only use the lower bound of expected number of eliminated items for calculation. The value of ϵ is set at 0.01 for BetaTopK, and the same level of error is set for RecurTopK via (14).

We can see that in most settings, BetaTopK requires less comparisons than RecurTopK, and as K grows, the advantage of BetaTopK becomes more significant. This observation is in line with our expectation. The BetaTopK Algorithm 1 generates the top-K list by dynamically reducing candidates, promoting or discarding them. When K is large, many candidates can be easily promoted, or many easily

discarded. In contrast, the RecurTopK performs a more thorough analysis of the top-K elements; in particular, the number of recursive calls is independent of K. As a consequence, the algorithm does not exploit the ease with which some elements can be determined to be in the top-K set. As a result, the differences between BetaTopK and RecurTopK get larger, the larger the value of K.

Furthermore, Table 1 reveals that with the same setting of K, N and s, BetaTopK gains a larger advantage when the crowd is more prone to mistakes, i.e., when q is smaller. This demonstrates that the dynamic nature of BetaTopK enables the more efficient handling of crowd errors.

4.2.3 Experimental Comparison. In Table 2 we compare the analytical evaluation of the BetaTopK Algorithm 1 (denoted as A), with the experimental evaluation of the optimized BetaTopK Algorithm 3 (denoted as E). The comparison uses the same $\epsilon = 0.01$ as in the analytical results. This comparison shows the value of the optimizations, which in many cases reduce the number of crowdsourced comparisons by a factor of two.

In Figure 2, we plot the number of crowdsourced comparisons required by RecurTopK, BetaTopK Algorithm 1 analyzed analytically (termed BetaTopK Analytical), and the BetaTopK Algorithm 3 analyzed experimentally (termed BetaTopK Experimental) when $N = 10,000$. The x-axis is the average probability of correctness of the crowd, and y-axis is the number of comparisons. The results in Table 2 demonstrate the dramatic reduction in number of crowdsourced comparisons required by BetaTopK compared to RecurTopK. The results also show that the number of comparisons required by experiments is significantly smaller than that of analytical results.

5 CONCLUSION

In this paper, we propose an online top-K algorithm in crowdsourcing environments, which focuses on maximizing the total qualities of the selected top-K items, while is indifferent to their ordering. We prove that the expected number of comparisons required by this algorithm is $O(N \log N)$. Our analysis shows that the algorithm can achieve comparable selection outcome while require fewer comparisons than existing algorithms; the advantage is especially marked when the value K of selected items is a non-negligible proportion of the total number of items. We also show how the algorithm can be optimized, further improving its performance.

An interesting direction for future research is to study active learning strategies that would further accelerate convergence, and that would improve the quality of the partial results that are available before the algorithm completes. We believe that by using sampling strategies that focus on the items with less confidence in qualities, the algorithms may be further sped up.

REFERENCES

[1] Arpit Agarwal, Shivani Agarwal, Sepehr Assadi, and Sanjeev Khanna. 2017. Learning with Limited Rounds of Adaptivity: Coin Tossing, Multi-Armed Bandits, and Ranking from Pairwise Comparisons. In *Conference on Learning Theory*. 39–75.

[2] Nir Ailon. 2012. An Active Learning Algorithm for Ranking from Pairwise Preferences with an Almost Optimal Query Complexity. *Journal of Machine Learning Research* 13, Jan (2012), 137–164.

[3] T. Ballinger and Nathaniel Wilcox. 1997. Decisions, Error and Heterogeneity. *Economic Journal* 107, 443 (1997), 1090–1105.

[4] Daren C. Brabham. 2017. Crowdsourcing. In *The International Encyclopedia of Organizational Communication*. American Cancer Society, 1–6.

[5] Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika* 39, 3/4 (1952), 324–345.

[6] Mark Braverman, Jieming Mao, and S. Matthew Weinberg. 2016. Parallel Algorithms for Select and Partition with Noisy Comparisons. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing (STOC '16)*. ACM, New York, NY, USA, 851–862.

[7] Mark Braverman and Elchanan Mossel. 2008. Noisy Sorting Without Resampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 268–276.

[8] Robert Busa-Fekete, Balazs Szorenyi, Weiwei Cheng, Paul Weng, and Eyke Huellermeier. 2013. Top-k Selection based on Adaptive Sampling of Noisy Preferences. In *International Conference on Machine Learning*. 1094–1102.

[9] Xi Chen, Paul N. Bennett, Kevyn Collins-Thompson, and Eric Horvitz. 2013. Pairwise ranking aggregation in a crowdsourced setting. In *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 193–202.

[10] Xi Chen, Sivakanth Gopi, Jieming Mao, and Jon Schneider. 2017. Competitive analysis of the top-K ranking problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1245–1264.

[11] Yuxin Chen and Changho Suh. 2015. Spectral MLE: Top-K Rank Aggregation from Pairwise Comparisons. *arXiv:1504.07218 [cs, math, stat]* (April 2015). arXiv: 1504.07218.

[12] Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. 2016. Crowdsourcing for top-K query processing over uncertain data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 1452–1453.

[13] Donald Davidson. 1974. Experimental Tests of a Stochastic Decision Theory (1959). In *Economic Information, Decision, and Prediction: Selected Essays: Volume I Part I Economics of Decision*, Jacob Marschak (Ed.). Springer Netherlands, Dordrecht, 133–171.

[14] Susan B Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. 2013. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*. ACM, 225–236.

[15] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. 2013. Using the Crowd for Top-k and Group-by Queries. In *Proceedings of the 16th International Conference on Database Theory (ICDT '13)*. ACM, New York, NY, USA, 225–236.

[16] Luca de Alfaro, Vassilis Polychronopoulos, and Neoklis Polyzotis. 2016. Efficient Techniques for Crowdsourced Top-k Lists. In *Fourth AAAI Conference on Human Computation and Crowdsourcing*.

[17] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. 2011. Crowdsourcing Systems on the World-Wide Web. *Commun. ACM* 54, 4 (April 2011), 86–96.

[18] Arpad E. Elo. 1978. *The rating of chess players, past and present*. Arco Pub.

[19] Brian Eriksson. 2013. Learning to Top-K Search using Pairwise Comparisons. In *Artificial Intelligence and Statistics*. 265–273.

[20] Lester Randolph Ford. 1957. Solution of a Ranking Problem from Binary Comparisons. *The American Mathematical Monthly* 64, 8P2 (Oct. 1957), 28–33.

[21] Reinhard Heckel, Nihar B. Shah, Kannan Ramchandran, and Martin J. Wainwright. 2016. Active Ranking from Pairwise Comparisons and when Parametric Assumptions Don't Help. (June 2016).

[22] Ralf Herbrich, Tom Minka, and Thore Graepel. 2007. TrueSkill : A Bayesian Skill Rating System. In *Advances in Neural Information Processing Systems 19*. MIT Press, 569–576.

[23] David R. Hunter. 2004. MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics* 32, 1 (Feb. 2004), 384–406.

[24] Kevin G Jamieson and Robert Nowak. 2011. Active Ranking using Pairwise Comparisons. In *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., 2240–2248.

[25] Minje Jang, Sunghyun Kim, Changho Suh, and Sewoong Oh. 2016. Top-K Ranking from Pairwise Comparisons: When Spectral Ranking is Optimal. *arXiv e-prints* 1603 (March 2016), arXiv:1603.04153.

[26] R. Duncan Luce. 1959. *Individual choice behavior*. John Wiley, Oxford, England.

[27] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. 2017. A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software* 126 (April 2017), 57–84.

[28] Lucas Maystre and Matthias Grossglauser. 2015. Fast and Accurate Inference of Plackett–Luce Models. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 172–180.

[29] Sahand Negahban, Sewoong Oh, and Devavrat Shah. 2012. Iterative Ranking from Pair-wise Comparisons. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12)*. Curran Associates Inc., USA, 2474–2482.

[30] Sahand Negahban, Sewoong Oh, and Devavrat Shah. 2017. Rank Centrality: Ranking from Pairwise Comparisons. *Operations Research* 65, 1 (2017), 266–287.

[31] Vassilis Polychronopoulos, Luca De Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. 2013. Human-Powered Top-k Lists.. In *WebDB*. 25–30.

[32] Arun Rajkumar and Shivani Agarwal. 2014. A Statistical Convergence Perspective of Algorithms for Rank Aggregation from Pairwise Data. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. JMLR.org, Beijing, China, I–118–I–126.

[33] Nihar B. Shah and Martin J. Wainwright. 2018. Simple, Robust and Optimal Ranking from Pairwise Comparisons. *Journal of Machine Learning Research* 18, 199 (2018), 1–38.

[34] Changho Suh, Vincent Y. F. Tan, and Renbo Zhao. 2017. Adversarial Top- K Ranking. *IEEE Transactions on Information Theory* 63, 4 (April 2017), 2201–2225.

Table 1: Number of comparisons required by RecurTopK and analytical BetaTopK

Parameters		$q = 0.6$		$q = 0.7$		$q = 0.8$		$q = 0.9$		$q = 1.0$	
		Recur	**Beta**	**Recur**	**Beta**	**Recur**	**Beta**	**Recur**	**Beta**	**Recur**	**Beta**
$K < s$	$N = 100, K = 10, s = 15$	21,392	20,533	6,427	13,759	3,682	8,496	2,209	5,073	736	4,231
	$N = 100, K = 50, s = 60$	1,430,950	11,632	367,752	10,371	150,092	8,406	68,769	7,554	13,753	6,780
	N=10,000, $K = 50, s = 60$	3,376,557	2,264,560	1,400,874	596,339	657,249	179,726	431,882	146,932	86,376	75,750
$K \geq s$	$N = 100, K = 10, s = 5$	42,844	20,533	12,206	13,759	6,530	8,496	3,692	5,073	854	4,231
	N=10,000, $K = 50, s = 30$	3,048,114	2,264,560	1,261,284	596,339	640,196	179,726	310,544	146,932	82,413	75,750
	N=10,000, $K = 500, s = 30$	13,444,818	2,162,972	3,994,380	1,003,450	2,126,615	604,319	1,192,733	337,860	258,851	219,106
	N=10,000, K=5,000, $s = 30$	331,364,256	1,163,242	80,919,361	1,037,168	33,986,324	840,666	18,308,137	755,406	2,629,951	370,000

Table 2: Comparisons: analytical and experimental results of BetaTopK

Parameters	$q = 0.6$		$q = 0.7$		$q = 0.8$		$q = 0.9$		$q = 1.0$	
	A	**E**	**A**	**E**	**A**	**E**	**A**	**E**	**A**	**E**
$N = 100, K = 10$	20,533	18,500	13,759	6,220	8,496	3,440	5,073	1,840	4,231	1,520
$N = 100, K = 50$	11,632	10,460	10,371	9,420	8,406	8,200	7,554	5,760	6,780	3,600
$N = 10,000, K = 50$	2,264,560	580,000	596,339	204,000	179,726	118,000	146,932	100,000	75,750	74,000
$N = 10,000, K = 500$	2,162,972	2,130,000	1,003,450	634,000	604,319	320,000	337,860	186,000	219,106	120,000
$N = 10,000, K = 5,000$	1,163,242	1,158,000	1,037,168	968,000	840,666	832,000	755,406	748,000	678,031	458,000

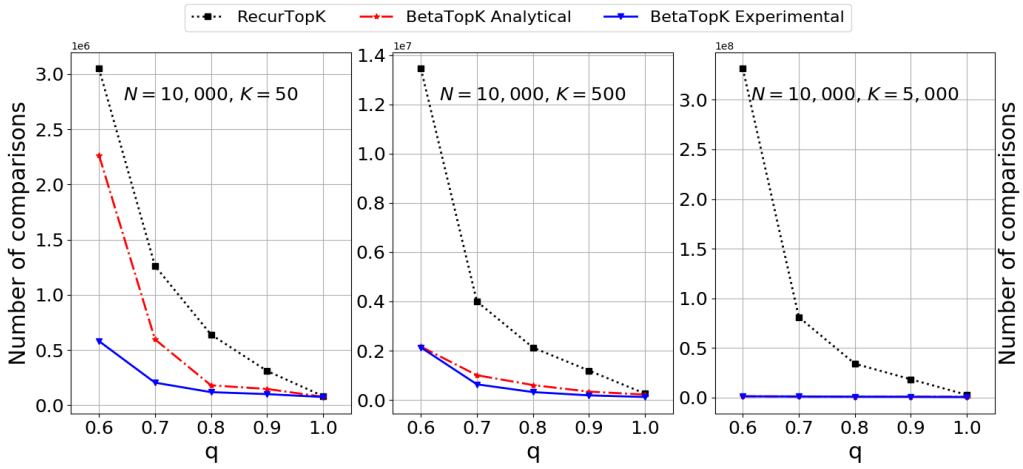

Figure 2: Number of comparisons required by each algorithm

[35] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. 2012. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*. ACM, 989–998.

[36] Fabian Wauthier, Michael Jordan, and Nebojsa Jojic. 2013. Efficient Ranking from Pairwise Comparisons. In *PMLR*. 109–117.

[37] Xiaohang Zhang, Guoliang Li, and Jianhua Feng. 2016. Crowdsourced Top-k Algorithms: An Experimental Evaluation. *Proceedings of the VLDB Endowment* 9, 8 (April 2016), 612–623.