A General Early-Stopping Module for Crowdsourced Ranking

Caihua Shan[1], Leong Hou U[2], Nikos Mamoulis[3], Reynold Cheng[1], and Xiang Li[1]

[1] Department of Computer Science, University of Hong Kong, Hong Kong
[2] State Key Laboratory of Internet of Things for Smart City, Department of Computer and Information Science, University of Macau, Macau
[3] Department of Computer Science, University of Ioannina, Ioannina, Epirus, Greece
{chshan,ckcheng,xli2}@cs.hku.hk ryanlhu@umac.mo nikos@cs.uoi.gr

Abstract. Crowdsourcing can be used to determine a total order for an object set (e.g., the top-10 NBA players) based on crowd opinions. This ranking problem is often decomposed into a set of microtasks (e.g., pairwise comparisons). These microtasks are passed to a large number of workers and their answers are aggregated to infer the ranking. The number of microtasks depends on the budget allocated for the problem. Intuitively, the higher the number of microtask answers, the more accurate the ranking becomes. However, it is often hard to decide the budget required for an accurate ranking. We study how a ranking process can be terminated early, and yet achieve a high-quality ranking and great savings in the budget. We use statistical tools to estimate the quality of the ranking result at any stage of the crowdsourcing process, and terminate the process as soon as the desired quality is achieved. Our proposed early-stopping module can be seamlessly integrated with most existing inference algorithms and task assignment methods. We conduct extensive experiments and show that our early-stopping module is better than other existing general stopping criteria.

1 Introduction

Crowdsourcing has been used to address a variety of problems, such as entity matching [28], image labeling [14], and object ranking [12,16]. These problems, which are typically hard for computers to solve, can be easier for humans. In this paper, we study the use of crowdsourcing on *ranking* objects. This approach, which has received a lot of attention from different research communities [21,6,16], is particularly helpful when ranking cannot be done objectively. For example, to determine the greatest athletes of all times or the best pictures of a landmark, we could solicit opinions from the crowd and aggregate them to a ranking that maximizes the consensus. In addition, crowdsourced ranking can be used to filter data for subsequent machine learning tasks. For instance, ranking answers to a question posted in a forum and selecting only the top ones can ease the burden of natural language processing.

To conduct crowdsourced ranking, existing solutions typically decompose the ranking process into a set of small and easy-to-answer microtasks, such as pairwise comparisons [31]. The microtasks are then distributed via crowdsourcing platforms, such as Amazon Mechanical Turk (AMT) [1] and FigureEight [2], to crowd workers by offering incentives, e.g., money, reputation, etc. The final ranking is computed by an inference algorithm based on the answers collected from the crowd. Naturally, the ranking accuracy is proportional to the number of collected answers to microtasks, i.e., the *total budget* paid by the requester.

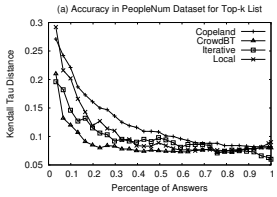
Recent studies [8,12,6] attempt to improve the *inference algorithm* $\mathcal{I}$ and fine-tune the *task assignment* $\mathcal{T}$ (i.e., by dispatching tasks to suitable workers), in order to spend the budget more effectively. Typically, the microtask answers are collected in batches. Let A_i be the ith batch of answers; Inference algorithm module $\mathcal{I}$ infers the *interim ranking* from $A_1 \cup ... \cup A_i$; Task assignment module $\mathcal{T}$ is used to determine the next batch of microtasks and assign them to crowdsourcing platforms.

According to a recent experimental survey on crowdsourced ranking [31], there is no single winner method that outperforms all others in all performance factors (accuracy, convergence rate, efficiency, scalability). In addition, most approaches require the budget to be set in advance, but they offer no guideline on how to set this value. Hence, it is expected that the requester sets a large enough budget, hoping that the ranking process will converge to a stable ranking. This raises an interesting question: *can we spend less and achieve approximately the same ranking, as if we had spent all the budget?*

To answer this question, we first investigate how much budget could be saved when some representative inference algorithms are applied, i.e., Copeland [22], CrowdBT [8], Iterative [12], and Local [12]. Details about these methods are given in Sec. 4.1. We carry out the top-10 query tasks on two public datasets, namely peopleAge [31] and peopleNum [15]. Fig. 1 shows how the *accuracy* of these algorithms varies as the budget increases. As an accuracy measure, we utilize Kendall's tau distance between the rankings progressively inferred and the ground truth ranking. All methods converge to a *stable state*[4], where the change of the distance induced by the inferred ranking is very small. In Fig. 1(a) and (b), CrowdBT reaches a stable state after using just 40% of the budget, whereas all other methods converge when 60% of the budget is used. Obviously, we can *stop early* the crowdsourcing process when we reach a stable state. We now face the following challenge: *how do we know if the ranking process has reached a stable state?*

To tackle this challenge, we develop a novel Early-Stopping (ES) module that attempts to predict the next batch of answers by probabilistic analysis. We then use Monte Carlo simulation [19], based on the prediction model, to construct the distribution of the final answer and, in turn, derive the *expected accuracy* of the final state. This helps us to assess when the ranking process reaches its stable state, subject to a budget B. To early-stop the process, our ES module requires an *accuracy tolerance* θ parameter, i.e., the acceptable accuracy that we

[4] A formal definition of the stable state is provided in the Sec. 2.2.



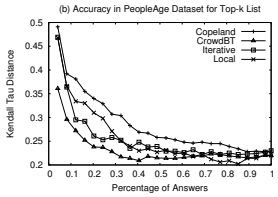


Fig. 1: Accuracy vs. Budget

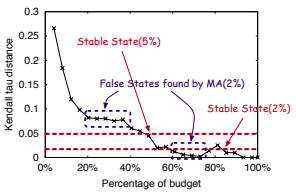


Fig. 2: Examples of Stable State

can afford to lose when compared to the ranking that will be obtained if all the budget is used up.

Our ES module can seamlessly be used by most ranking processes with minimal effort. The only requirement is that the process provides interfaces for the inference and task assignment modules, and accepts a programming call to terminate the crowdsourcing process, when our module determines that the expected accuracy already satisfies tolerance θ.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, we are the first to propose a general Early-Stopping (ES) module for crowdsourced ranking.
- Our ES module is orthogonal to any inference algorithm or task assignment method, and does not interfere with the flow of the crowdsourced ranking process.
- We thoroughly evaluate our ES module with subjective and objective tasks, different inference algorithms and task assignment methods, varying budgets and accuracy tolerances. Our module can save even half of the budget given to the ranking processes.

The rest of the paper is organized as follows. We formulate the problem and provide definitions and notations in Sec. 2. Our ES module is described in detail in Sec. 3. The experimental evaluations are shown in Sec. 4. We discuss related work in Sec. 5 and conclude in Sec. 6.

2 Preliminaries

We first define crowdsourced ranking and top-k queries as follows.

Definition 1 (Crowdsourced Ranking). *Given a set of n objects $\mathcal{O} = \{o_1, ..., o_n\}$, use human workers to decide a total order $\sigma = \{o_i \prec o_j \prec ...\}$.*

Definition 2 (Crowdsourced Top-k Query). *Given a set of n objects $\mathcal{O} = \{o_1, ..., o_n\}$, use human workers to find a ranked list $\sigma^k = \{o_i \prec o_j \prec ...\}$ of size k, such that for any $o_i \in \sigma^k \wedge o_l \notin \sigma^k$, $o_i \prec o_l$.*

Note that the operator $\prec$ is a conclusion drawn from the crowd's answers. For instance, given some replies to a question posted in a forum, we can ask the crowd to conduct pairwise comparisons between the replies, and then use existing inference algorithms to process the crowd's input and find the top-5 replies. Note that comparing two replies is not machine friendly since it not only requires strong natural language processing techniques but also a good understanding of the question, i.e., domain expertise.

2.1 Distance Between Rankings

In our solution, we need to measure the distance (i.e., difference) between the ranking inferred at an intermediate state and the ranked list at the final state. To measure the distance between two rankings, a common practice is to use *Kendall's tau distance*, i.e., the number of inverse pairs of objects.

We use the normalized Kendall's tau distance for complete rankings and top-k ranked lists as defined in Eq. 1 and Eq. 2, respectively:

$$\mathbb{D}(\sigma_1, \sigma_2) = \frac{\sum_{(o_i,o_j)\in O\times O, i<j} \mathbb{1}(o_i \prec o_j, \sigma_1) \times \mathbb{1}(o_i \succ o_j, \sigma_2)}{n(n-1)/2} \tag{1}$$

$$\mathbb{D}(\sigma_1^k, \sigma_2^k) = \frac{\sum_{(o_i,o_j)\in O\times O, i<j} \mathbb{1}(o_i \prec o_j, \sigma_1^k) \times \mathbb{1}(o_i \succ o_j, \sigma_2^k)}{k^2} \tag{2}$$

where $\mathbb{1}$ is the indicator function that equals to 1 when its predicate is true, or 0 otherwise. When σ_1 and σ_2 are reversed, the numerator of Eq. 1 takes its maximum possible value $n(n-1)/2$, and Eq. 1 reaches the highest value of 1. As for Eq. 2, the numerator takes its maximum value k^2 when objects in σ_1^k and σ_2^k have no intersection.

2.2 Stable State & Optimal Stopping Point

Publishing a batch of microtasks into the crowdsourced platform is a common strategy to accelerate the speed of collections. Let p_i be the state after collecting the ith batch of answers A_i and $\sigma_i = \mathcal{I}(A_1 \cup ... \cup A_i)$ is the ranked list at p_i. The stopping module should check whether to stop at each p_i. Without loss of generality, we assume that the budget B is the total number of microtasks we plan to publish and the number of microtasks, n_{batch}, is the same in each batch. B/n_{batch} is the total number of batches needed to collect all answers. We then give a formal definition of the stable state that we mentioned in the Introduction:

Definition 3 (Stable State). *Given the whole collection process* $\{A_1, A_2, \cdots, A_{B/n_{batch}}\}$ *and an accuracy tolerance* $\theta \in [0,1]$ *from the requester,* p_l *is called as a* stable state *of the process if:*

1. $\forall p_i, p_j \in [p_l, p_{final}],\ \mathbb{D}(\sigma_i, \sigma_j) \leq \theta$ *2.* $\nexists\ p_i < p_l,\ p_i$ *is a stable state where* $l \in [1, B/n_{batch}]$ *and* $p_{final} = B/n_{batch}$.

The first condition secures that the distances between the rankings at any two states (from p_l to the final) do not exceed θ. The second condition secures

the maximality that no earlier (better) stable state can be found in the entire process. It is obvious only one stable state exists in each collection process.

The stopping point p_{sc} is the moment decided by a stopping criterion (SC) to early stop the ranking process. Based on the stable state definition, we can say that

Corollary 1 (The Optimal Stopping Point). *The optimal point $p_{optimal}$ to early stop a ranking process is when the process turns into the stable state, i.e., $p_{optimal} = p_{\iota}$.*

The optimal stopping point guarantees the optimality because it saves up as much as possible the budget and ensures the distances from the ranked list at the stopping point to the final are always smaller than the accuracy tolerance θ.

For example, Fig. 2 shows the distance between the current and the final ranking at all states of the process. We show two optimal stopping points with $\theta = 5\%$ or $\theta = 2\%$. Basically we may save more budget with larger θ. Here we save 50% budget for $\theta = 5\%$, and 10% budget for $\theta = 2\%$.

One may wonder whether some simple method, e.g., Moving Average and Weighted Moving Average [3], can find p_{optimal}. We also show two kinds of intervals in Fig. 2. The first purple rectangle is an interval that tends to be stable during a certain time but descends gradually as more budget consumes. The second one also tends to be stable but the change of rankings is larger than θ as more budget consumes. Given a current point p_i, moving average uses the previous rank lists in a certain window size to represent the inferred rankings in the future. It is easy to drop out into these intervals and cause the process to stop earlier than it should. Besides, it is hard for users to set the best parameter values for them, such as the window size. Bad parameters lead to the worst stopping position. To avoid stopping at these intervals, we propose a novel ES module that attempts to discover the optimal stopping point.

3 Early-Stopping Module

3.1 Predicting the Next Answer Set

Consider a crowdsourcing rank process $\mathcal{R}$, based on an inference module $\mathcal{I}$ and a task assignment module $\mathcal{T}$, that has already collected the ith batch answer set ($A^c = A_1 \cup ... \cup A_i$). We predict the next batch of answers by a three-stage process, including (1) determining new tasks t_{new}, (2) predicting the answers a_{new} of t_{new}, and (3) estimating the influence of worker reliability.

Determining new tasks, t_{new} Recall that the microtasks of crowdsourced ranking are pairwise comparisons (o_i, o_j). Given the collected answer set A^c, the task assignment module $\mathcal{T}$ decides the importance of tasks. The most important n_{batch} tasks are distributed to crowdsourcing platforms as the next batch. We predict answers for these tasks in our subsequent prediction model.

Predicting the answer, a_{new} Given the collected answer set A^c and a chosen task $t_{new} = (o_i, o_j)$, we want to predict the answer to t_{new}. We assume that the workers are reliable since they have to obey the crowdsourcing platform policy, e.g., gain reputation via user feedback. Thereby, we can regard the answer a_{new} of the task $t_{new} = (o_i, o_j)$ as a Bernoulli distribution of the probability of $o_i \prec o_j$, denoted as P_{ij}. Formally, it can be written as $a_{new} \sim \text{Bernoulli}(P_{ij})$ where P_{ij} is the probability of $o_i \prec o_j$. Several models for P_{ij} has been suggested in previous crowdsourcing studies [18,5,26]. For instance, the Bradley-Terry (BT) model [5] defines $P_{ij} = \frac{e^{s_i}}{e^{s_i}+e^{s_j}}$, where s_i is the latent score of object o_i. The Thurstonian model [26] defines $P_{ij} = \Phi(s_i - s_j)$, where Φ is the normal cumulative distribution function. However, some inference modules [11,10] do not build on the latent scores of objects.

We attempt to design a new estimation model that is suitable for most inference modules. We estimate the probability P_{ij} independently, i.e., P_{ij} only based on the previous answer set of the task (o_i, o_j). Suppose that the current answer set is A^c; we build an observed matrix M, where M_{ij} is the number of answers reporting $o_i \prec o_j$ in A^c. P_{ij} depends on M_{ij} and M_{ji}.

We use maximum a posteriori probability (MAP) to calculate $\hat{P}_{ij}$:

$$\begin{aligned}\hat{P}_{\text{MAP}}(M) &= \arg\max_P Pr(P \mid M) = \arg\max_P \prod_{i,j|i<j} Pr(P_{ij} \mid M_{ij}, M_{ji}) \\ &= \arg\max_P \prod_{i,j|i<j} \frac{Pr(M_{ij}, M_{ji} \mid P_{ij})\, Pr(P_{ij})}{\int_0^1 Pr(M_{ij}, M_{ji} \mid p_{ij})\, Pr(p_{ij})\, dp_{ij}} \propto \arg\max_P \prod_{i,j|i<j} Pr(M_{ij}, M_{ji} \mid P_{ij})\, Pr(P_{ij})\end{aligned} \tag{3}$$

If we assume the prior distribution of P_{ij} as $Beta(1,1)$ which is the conjugate prior for the Bernoulli distribution, the posterior distribution of P_{ij} is $Pr(P_{ij} \mid M_{ij}, M_{ji}) \sim Beta(M_{ij}+1, M_{ji}+1)$. The reason behind using $Beta(1,1)$ is that we believe that we have equal probability to get either $o_i \prec o_j$ or $o_i \succ o_j$. It could also be interpreted as Laplace smoothing to avoid some undefined calculation, e.g., $Beta(0,0)$. The MAP of $\hat{P}_{ij}$ equals the mode of the posterior distribution, which is

$$\hat{P}_{ij} = \frac{M_{ij}+1}{M_{ij}+M_{ji}+2}. \tag{4}$$

Alternatively, we could also use maximum likelihood estimate (MLE) to calculate $\hat{P}_{ij}$:

$$\begin{aligned}\hat{P}_{\text{MLE}}(M) &= \arg\max_P Pr(M \mid P) = \arg\max_P \prod_{i,j|i<j} Pr(M_{ij}, M_{ji} \mid P_{ij}) \\ &= \prod_{i,j|i<j} \binom{M_{ij}+M_{ji}}{M_{ij}} {P_{ij}}^{M_{ij}} (1-P_{ij})^{M_{ji}} = \arg\max_P \prod_{i,j|i<j} {P_{ij}}^{M_{ij}} (1-P_{ij})^{M_{ji}}\end{aligned} \tag{5}$$

The MLE of $\hat{P}_{ij}$ equals to $\frac{M_{ij}}{M_{ij}+M_{ji}}$. Similarly, if we replace M_{ij} and M_{ji} by $M_{ij}+1$ and $M_{ji}+1$, respectively, by the Laplace smoothing, then the MLE equation will be identical to Eq. 4 (from MAP).

In summary, we estimate P_{ij} from M based on A^c and then sample an answer a_{new} by Bernoulli(P_{ij}) for the task (o_i, o_j).

Estimating the influence of worker reliability In this section, we discuss how worker reliability influences the predicting process of answer a_{new}. As mentioned in Sec. 3.1, the posterior distribution P_{ij} can be estimated based on the collected answers A^c. The estimation framework is built on our underlying assumption that *every worker is reliable.*

We attempt to add the effect of workers' reliability (i.e., the probability of answering correctly). Assume that we already know the average worker reliability rel in A^c, the probability of a new answer a_{new} should be revised as $P'_{ij} = P_{ij} \times rel + (1 - P_{ij}) \times (1 - rel)$. It means that workers give reliable answers with the probability rel while giving untrustworthy and opposite answers with the probability $1 - rel$. The worker reliability can be provided by the platforms and calculated based on workers' answer history in other projects. If the platforms do not provide this function, we could also set a lower bound of the required quality of workers or do the qualification test to filter bad workers before the actual assignments. This lower bound or qualified reliability is regarded as rel.

Generating answers in the next batch A_{i+1} So far, we have discussed how to predict the next task answer a_{new} based on the collected answers A^c and worker reliability. To predict the answers A_{i+1} obtained in the next batch, we apply an iterative process that generates answers one after another. Algorithm 1 shows the pseudo code of the iterative process. We first estimate P_{ij} in line 2-6. Then we utilize the assignment module $\mathcal{T}$ to get the importance of tasks. We select the first n_{batch} important tasks, predict the answers respectively and add into A_{i+1} in line 7-12.

We can also predict a "complete" answer set A (obtained when we use up the budget B). Based on Algorithm 1, we predict A_{i+1} based on $A^c = A_1 \cup ... \cup A_i$. Similarly, A_{i+2} is predicted based on $A_1 \cup ... \cup A_{i+1}$, A_{i+3} is predicted based on $A_1 \cup ... \cup A_{i+2}$ and so on. Finally, we can predict $A = A^c \cup A_{i+1} \cup A_{i+2} \cup A_{i+3}...$ until the size of A is equal to the given budget B.

3.2 Calculating Deviation

In the last section, we showed how to predict a "complete" answer set A. In this section, we discuss how to judge whether the current point satisfies the definition of the optimal stopping point.

Expected distance between rankings Given a "complete" and deterministic answer set A, the inference module $\mathcal{I}$ can be used to compute each interim ranking $\sigma_i = \mathcal{I}(A_1 \cup ... \cup A_i)$ and the distance $\mathbb{D}(\sigma_i, \sigma_j)$ between each two interim rankings (cf. Eq. 1 and 2). However, the probabilistic process may create many possible worlds, i.e., many possible answer sets $\mathbb{A} = \{A^1, A^2...\}$. If we know the occurrence probability of each possible world $Pr(A')$ where $A' \in \mathbb{A}$, the expected distance between the ith and jth batches can be defined as $\mathrm{E}[\mathbb{D}_{ij}] = \sum_{A' \in \mathbb{A}} Pr(A') \times \mathbb{D}(\mathcal{I}(A'_1 \cup ... \cup A'_i), \mathcal{I}(A'_1 \cup ... \cup A'_j))$

Algorithm 1: *Predicting the Next Answer Set*

Input: Current answer set A^c, Inference module $\mathcal{I}$, Task assignment module $\mathcal{T}$, the number of tasks in a batch n_{batch}

Initialize $A_{i+1} = \varnothing$

// Step 1: Build the matrix M and P

Built matrix M based on the answer in A^c

for *all possible* (i, j) **do**

  Estimate $P_{ij} = \frac{M_{ij}+1}{M_{ij}+M_{ji}+2}$ by Eq. 4

  Calculate P'_{ij} by P_{ij} with worker reliability rel

// Step 2: Getting the next n_{batch} important tasks from $\mathcal{T}$

$T = \mathcal{T}(A^c)$

for *each* t_{new} *in* T **do**

  // Step 3: Predict the answer of $t_{new} = (o_i, o_j)$

  Sample $a \sim \text{Bernoulli}(P'_{ij})$

  $A_{i+1} = A_{i+1} \cup \{a\}$

return A_{i+1};

However, it is difficult to calculate the occurrence probability because it is impossible to conduct a brute-force search for all possible worlds. To tackle this problem, we apply the Monte Carlo method, that allows an estimation of the sampling distribution of almost any statistic using random sampling method. The Monte Carlo method helps to generate a list of possible worlds, i.e., "complete" answer sets $\{A^1, A^2, ..., A^s, ...|s \in [1, n_{\text{sample}}]\}$. Given a pair (i, j), we are able to compute a list of pairs of rankings (σ_i^s, σ_j^s) and the corresponding distances $\mathbb{D}_{ij}^s$. By the law of large numbers, the expected distance $\mathrm{E}[\mathbb{D}_{ij}]$ can be approximated by taking the sample mean $\overline{\mathbb{D}}_{ij} = \frac{1}{n_{\text{sample}}} \sum_{s=1}^{n_{\text{sample}}} \mathbb{D}(\mathcal{I}(A_1^s \cup ... \cup A_i^s), \mathcal{I}(A_1^s \cup ... \cup A_j^s))$. If p_{current} is the earliest point satisfying $\forall p_i, p_j \in [p_{\text{current}}, p_{\text{final}}]$, $\overline{\mathbb{D}}_{ij} \leq \theta$, p_{current} is the stopping point decided by our ES module.

The number of required samples In the Monte Carlo method, it is important to decide the number of required samples such that the quality is secured. Following common practice, we use Hoeffding's inequality[13] to decide it.

Hoeffding's Inequality. Let $X_1, ..., X_n$ be independent random variables bounded by the interval $[0, 1] : 0 \leq X_i \leq 1$. Define the mean of these variables as $\overline{X} = \frac{1}{n}(X_1 + ... + X_n)$. Then we have $Pr(\mathrm{E}[\overline{X}] - \overline{X} \geq t) \leq e^{-2nt^2}$ where $t \geq 0$.

We regard a possible world answer set A^s as a sample. The distance $\mathbb{D}_{ij}^s$ can be regarded as an independent random variable given (i, j). Based on Hoeffding's Inequality, we have $Pr(\mathrm{E}[\mathbb{D}_{ij}] - \overline{\mathbb{D}}_{ij} \geq t) \leq e^{-2nt^2}$. This inequality could be transformed into a confidence interval of $\mathrm{E}[\mathbb{D}_{ij}]$:

$$Pr(\mathrm{E}[\mathbb{D}_{ij}] \leq \overline{\mathbb{D}}_{ij} + t) > 1 - e^{-2nt^2}, \tag{6}$$

where $\overline{\mathbb{D}}_{ij}$ is computed using Eq. 3.2. We require at least $\frac{\ln(1/\alpha)}{2t^2}$ samples to acquire $(1 - \alpha)$-confidence interval for $\mathrm{E}[\mathbb{D}_{ij}] \leq \overline{\mathbb{D}}_{ij} + t$.

Given the targeted accuracy tolerance θ, if we find that $\overline{\mathbb{D}}_{ij} \leq \theta - t$, we can also derive $Pr(\mathrm{E}[\mathbb{D}_{ij}] \leq \overline{\mathbb{D}}_{ij} + t \leq \theta) > 1 - e^{-2nt^2}$. We summarize it as the following theorem.

Theorem 1. *Given two points p_i and p_j, we secure that $\mathrm{E}[\mathbb{D}_{ij}] \leq \theta$ with confidence $(1-\alpha)$ after we random sample $\frac{ln(1/\alpha)}{2t^2}$ "complete" answer sets and find $\overline{\mathbb{D}}_{ij} \leq \theta - t$, for some $0 < t < \theta$.*

Here we set the confidence level $\alpha = 5\%$ and the estimation error t as an order of magnitude smaller than θ which secures enough samples to give a good estimation. We need to sample $n_{\text{sample}} \approx 10^4$ for $\theta = 0.1$, and $n_{\text{sample}} \approx 10^6$ when we set $\theta = 0.01$. The workload of sampling can be accelerated by multithreading or distributed computation.

We then analyze the number of samples to secure all $\mathrm{E}[\mathbb{D}_{ij}] \leq \theta$ with high probability from the current to the final state, i.e., judge whether the following formula holds: $\forall p_i, p_j \in [p_{\text{current}}, p_{\text{final}}],\ \mathrm{E}[\mathbb{D}_{ij}] \leq \theta$.

Assume that number of batches for remaining budget is $m = \frac{B - |A^c|}{n_{\text{batch}}}$, there are $(m+1)m/2$ different expected distances needed to compute and check. If we acquire confidence $(1-\alpha')$ for all the expected distances, the confidence $(1-\alpha)$ and the number of samples for each expected distance can be set as

$$\alpha = \frac{\alpha'}{(m+1)m/2} \quad \text{and} \quad n_{\text{sample}} = \frac{\ln((m+1)m/2) + \ln(1/\alpha')}{2t^2}. \tag{7}$$

We utilize the union bound to prove them. Let $\mathrm{E}[\mathbb{D}_{ij}] \leq \overline{\mathbb{D}}_{ij} + t$ for a pair (p_i, p_j) be an event. The confidence $(1-\alpha)$ means the probability that one event fails is α. Then based on the union bound, we derive that the probability that at least one of the events fails is no greater than the sum of the probabilities of the individual events, which is $\sum_{i=1}^{(m+1)m/2} \alpha = \alpha'$. In other words, the probability that no event fails is at least α', which satisfies our requirement.

Putting it all together. We put all of these techniques together to finalize the ES module, as shown in Algorithm 2.

4 Experimental Evaluation

In this section, we thoroughly evaluate our ES module on two real public datasets. Based on the different inference algorithms and task assignment approaches, we compare our ES module with some standard quality estimation methods.

4.1 Experimental Settings

Datasets. We use two real public datasets collected in AMT.

– *PeopleNum* [15] concerns 39 images taken in a mall, each of which includes multiple persons. The goal is to find the images with the most people in them. 6066 answers were collected from 197 workers. Each pair of images is answered by at least 5 workers.

Algorithm 2: *Early-Stopping module*

Input: Current answer set A^c, Inference module $\mathcal{I}$, Distance function $\mathbb{D}$, Budget B, accuracy tolerance θ, confidence interval α'

Calculate number of batches for remaining budget $m = \frac{B-|A^c|}{n_{\text{batch}}}$
Estimate the number of samples n_{sample} by Eq. 7
Initialize distance array d and $\overline{\mathbb{D}}$
for $1 \leq s \leq n_{sample}$ **do**
    Set a temporary answer set $A = A^c$
    Create a temporary array of ranked lists σ and set $\sigma[0] = \mathcal{I}(A^c)$
    for $1 \leq j \leq m$ **do**
        Predict a batch of answers A_j based on A by Alg. 1
        $A = A \cup A_j$
        $\sigma[j] = \mathcal{I}(A)$
    for $0 \leq i \leq m-1$ **do**
        for $i+1 \leq j \leq m$ **do**
            $d[s][i][j] = \mathbb{D}(\sigma[i], \sigma[j])$
for $0 \leq i \leq m-1$ **do**
    for $i+1 \leq j \leq m$ **do**
        $\overline{\mathbb{D}}[i][j] = \frac{1}{n_{\text{sample}}} \sum_{1 \leq s \leq n_{\text{sample}}} d[s][i][j]$
if $\overline{\mathbb{D}}[i][j] \leq \theta - t, \forall i, j$ **then**
    Invoke a programming call to terminate the rank process $\mathcal{R}$
else
    Continue collecting the next batch of answers

– *PeopleAge* [31] has 50 human photos with ages from 50 to 100. The goal is to find the photos that include the youngest person. There are 4930 answers from 150 workers. Each pair of photos is answered 3 times at least.

PeopleAge is hard because it is relatively subjective and different workers may have different opinions on age. The difficulty of *PeopleNum* is medium because it costs some time to count the persons.

Inference Modules $\mathcal{I}$. According to [31], we select some recommended inference algorithms and task assignment strategies to work with our ES module. For rank inference algorithms, we choose 4 methods:

– *Copeland* is a basic election approach where the objects are sorted by the times they win/lose in the comparisons.
– *Local* is a heuristic-based method based on a comparison graph, where nodes are objects and edges are built based on the pairwise comparisons. The score of an object is defined by the number of winning objects minus the number of losing objects in its 1-hop and 2-hop neighborhood.
– *Iterative* is an extended version of *local* supporting top-k queries. It keeps discarding the bottom half of the objects in the inference process and then re-computes the scores of the surviving objects. It repeats these two processes until k objects are left.
– *CrowdBT* is a representative method that uses the Bradley-Terry (BT) model to estimate the latent score s_i of the object o_i. It models the probability $o_i \prec o_j$

as $\frac{e^{s_i}}{e^{s_i}+e^{s_j}}$. Based on the crowdsourced comparisons A, it computes scores for the objects by maximizing $\sum_{o_i \prec o_j \in A} \log(\frac{e^{s_i}}{e^{s_i}+e^{s_j}})$.

Task Assignment Modules $\mathcal{T}$. We implemented 4 task assignment strategies based on commercial systems and existing work.

- *Random* is the strategy used by Amazon Mturk; tasks are assigned to coming workers at random and all tasks are answered the same number of times.
- *Greedy* chooses the pair of objects with the highest product of scores as the next task.
- *Complete* finds the top-x objects with the highest scores, where x is the largest integer satisfying $\frac{x(x-1)}{2} \leq n_{\text{batch}}$, and sets their pairwise comparisons as the next tasks.
- *CrowdBT* is an active learning method which selects the pair of objects which maximizes the information gain based on the estimated scores.

Based on the characteristics of the inference algorithms and the task assignment strategies, we form and test 7 rank processes $\mathcal{R}$: *Copeland-Random*, *Iterative-Random*, *Local-Random*, *CrowdBT-Random*, *Local-Greedy*, *Local-Complete*, and *CrowdBT-CrowdBT*.

Competitors. In order to evaluate our ES module, we also investigate two alternative stopping criteria based on statistical analysis.

- *Moving Average (MA)* stops when the following equation is smaller than θ at the first time. We calculate the distances between all pairs of consecutive rankings or top-k lists, generated at the last w points before the current stage and average them. Suppose we already collected i batches of answers:

$$\text{MA}(i, w) = \frac{\sum_{j=1}^{w} \mathbb{D}(\mathcal{I}(A_1 \cup ... \cup A_{i-j}), \mathcal{I}(A_1 \cup ... \cup A_{i-j+1}))}{w} \tag{8}$$

- *Weighted Moving Average (WMA)* is similar to MA, except that we assign different weights to the distances based on how far away they are from the current stage. The distance between the latest two rankings has the largest weight w, the second latest $w-1$, etc, and so on.

$$\text{WMA}(i, w) = \frac{\sum_{j=1}^{w} (w-j+1)\mathbb{D}(\mathcal{I}(A_1 ... A_{i-j}), \mathcal{I}(A_1 ... A_{i-j+1}))}{w(w+1)/2} \tag{9}$$

Evaluation Metrics. We define the optimal stopping point p_{optimal} in the Sec. 2.2. To evaluate the effectiveness of different stopping criteria, we analyze the difference between p_{optimal} and the stopping point p_{sc} predicted by a stopping criterion. Mathematically, it can be written as $\Delta_{\text{sc}} = \frac{|p_{\text{optimal}} - p_{\text{sc}}|}{B/n_{\text{batch}}}$.

4.2 Experimental Results

Implementation details We compare our ES module with two competitors, MA and Weighted MA, on two datasets for ranking or top-k queries. The objective is to show the superiority and robustness of ES on top of different rank processes $\mathcal{R}$.

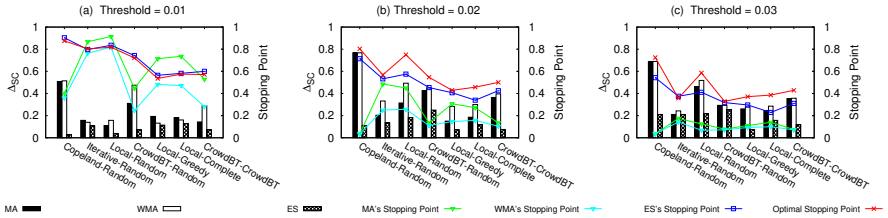


Fig. 3: Δ_{sc} & Stopping Points in PeopleNum Dataset for Top-10 Lists

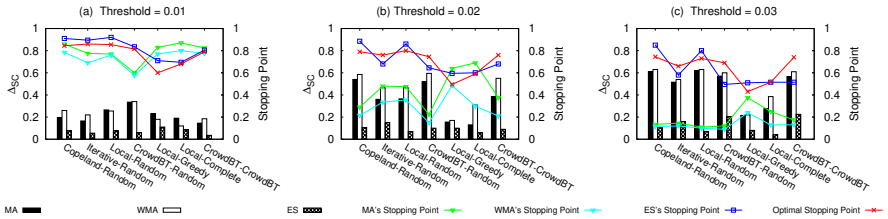


Fig. 4: Δ_{sc} & Stopping Points in PeopleAge Dataset for Top-10 Lists

The total budget is set to the number of answers in each original dataset. The number of microtasks in a batch is set to 200. To get an answer of a pairwise comparison (o_i, o_j), we randomly sample an answer from the answer set of (o_i, o_j) without replacement. If some pairs are running out of answers, we will simulate the next answer by a worker that has average reliability. To solve the cold-start problem of some task assignment strategies, we pre-generate an answer to every comparison. We also choose the best window size for MA and Weighted MA, which is 20 for PeopleNum dataset and 10 for PeopleAge dataset, respectively. Besides, a little change of initial answers for the cold-start problem will change the next sequence of microtasks. Thus, we run the collection process 10 times, and report the average performance.

We use two y-axes in Fig. 3 - 6. The left y-axis is Δ_{sc}, which is defined in Sec. 4.1. The right y-axis is the relative stopping point of MA, Weighted MA, ES and the optimal stopping point, which divided by the maximum possible stopping point, B/n_{batch}.

Top-k ranking Fig. 3 and 4 show Δ_{sc} and the stopping point of our ES module and the two alternative stopping criteria for top-k queries. Each stopping criterion is evaluated with seven rank processes (cf. Sec. 4.1). We set $k = 10$ by default. The accuracy tolerance θ is set to $\{0.01, 0.02, 0.03\}$. For instance, $\theta = 0.01$ means the possible number of inverse pairs between the current ranked list and the final state is smaller than $10^2 \times 0.01 = 1$.

ES outperforms the other two competitors for different rank processes and different datasets in all settings. When we set θ to a larger value (accepting higher accuracy loss), MA and Weighted MA tend to fall into the false states

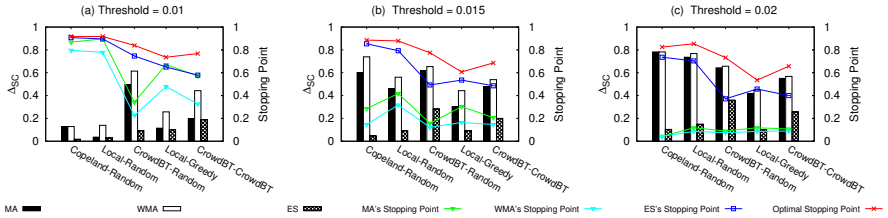


Fig. 5: Δ_{sc} & Stopping Points in PeopleNum Dataset for Rankings

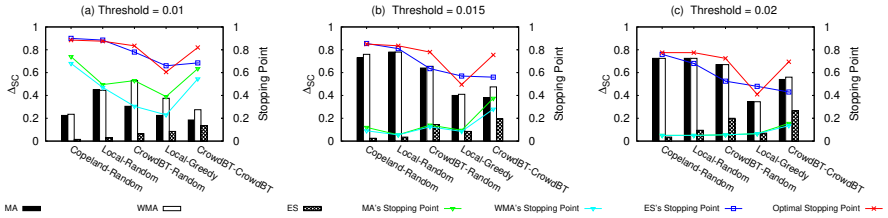


Fig. 6: Δ_{sc} & Stopping Points in PeopleAge Dataset for Rankings

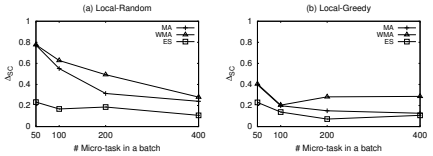


Fig. 7: Δ_{sc} in varied n_{batch}

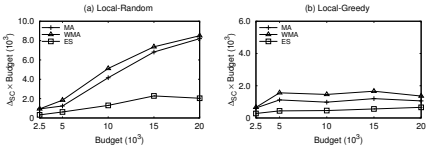


Fig. 8: Δ_{SC} in varied B

mentioned in the Sec. 2.2 and stop much earlier than the optimal stopping point, which results in high accuracy loss. According to the right y-axis, the position of p_{optimal} varies from 0.5 to 0.9. The stopping point of our ES module is very close to p_{optimal} when compared with the stopping points of MA and Weighted MA. This reveals that ES is effective in finding p_{optimal}.

Complete Ranking Fig. 5 and 6 show the performance for ranking queries. The accuracy tolerance θ is set to $\{0.01, 0.15, 0.02\}$. Note that we exclude two inference algorithms, *Iterative* and *Complete*, since they are designed for top-k queries. Similar to top-k queries, our ES module is much better than the other two competitions in terms of Δ_{sc}. The curve of ES's stopping point is very close to that of p_{optimal} compared with MA and Weighted MA.

4.3 Parameter Analysis

In this section, we test the effect of the total budget B and the number of microtasks in one batch n_{bach}. We evaluate these parameters with two rank processes, *Local-Random* and *Local-Greedy* on PeopleNum dataset.

Fig. 7 shows the effect of n_{batch}. In these experiments, we set the budget B equal to the total number of answers in the original dataset and set $\theta = 0.02$. ES is the clear winner since its Δ_{sc} is always less than or equal to 0.2 and outperforms MA and Weighted MA. In addition, MA and Weighted MA perform worse when n_{batch} becomes small (i.e., fewer microtasks in a batch), which means that the distance between two consecutive batches does not represent the distance between the current state and the final state.

Fig. 8 evaluates the effect of the budget B. Note that we use the absolute number of answers instead of a percentage in the y-axis. We set $\theta = 0.02$ and $n_{\text{batch}} = 200$ as default. We try $\{2.5, 5.0, 10.0, 20.0\} * 10^3$. Δ_{sc} of ES is always smaller than the corresponding Δ_{sc} of MA and Weighted MA. Particularly, errors of MA and Weighted MA increase dramatically when B increases in *Local-Random*. This is because increasing budget B improves the quality of the final result and the position of the optimal stopping point moves backwards. But the stopping points predicted by MA and Weighted MA do not change.

5 Related work

Crowdsourced ranking. The ranking problem has a long history and has been studied in the past several decades. Simple traditional ranking algorithms, e.g., BordaCount [4] and Copeland [22], rank objects by the times they win/lose in the comparisons. But these algorithms do not consider that the crowd may give incorrect answers. How to deal with noisy answers and control worker qualities is the key component in almost all crowdsourcing problems [17,24,25]. Several *inference algorithms* and *task assignment strategies* are proposed to solve it.

Inference algorithms in raking problems can be divided into two categories: heuristic-based solutions from the DB community approach the problem as a top-k operation in databases, and machine-learning algorithms formalize it as a leaning problem and maximize the likelihood of top-k objects. Heuristic score-based algorithms [12,20,29,27] rank objects by estimating the underlying score of objects. CrowdBT [8] and CrowdGauss [21] are ML algorithms, which set the objective function based on the assumption and use maximum likelihood to obtain the top-k object with the highest probability. Regarding task assignment strategies, Amazon MTurk follows a random assignment strategy; i.e., microtasks are randomly dispatched to each coming worker. Random assignment does not consider the difficulties of microtasks. Some heuristic assignment methods [12] aim at maximizing the probability of obtaining the top-k result, e.g., by selecting most promising object pairs (e.g., with the largest latent scores) to compare. [6] avoids some unnecessary comparisons by setting a bound for the object latent score. Active learning methods are also used in CrowdBT [8] and CrowdGauss [21] to compare objects with the largest expected information gain.

According to a recent experimental study [31], different inference and assignment methods have their own advantages and there does not exist a single best one. Machine-learning methods typically have high quality. Still, global inference heuristics that utilize global comparison results achieve comparable and even

higher quality than ML methods. Local inference heuristics have poor quality, but have higher efficiency and scalability. For task assignment, active-learning methods achieve higher quality than heuristics, but they have low efficiency.

Stopping Criteria. Stopping criteria have been defined for various crowdsourcing problems. [23] designs an early-stopping strategy for multiple-choice-question problems (e.g., choosing the opinion positive, neutral, or negative in a sentence). [30] uses Sequential Probability Ratio Test to decide when to stop for multi-labeling tasks (e.g., labeling pictures as a portrait or a landscape). Besides, [7] uses Chao92 estimator to evaluate the level of completion for entity collection (e.g., collecting a set of active NBA players). The settings of all these problems are quite different from crowdsourced ranking because microtasks are independent in these problems while correlated in the ranking problem.

Some previous studies on crowdsourced ranking define their special stopping conditions. For instance, [9] assumes that each object has a latent score and answers to pairwise comparisons follow the Bradley-Terry model [5]. [16] asks the crowd to give a numerical answer in $[0, 1]$ for a pairwise comparison and calculates the confidence interval of the result. However, these approaches are based on special assumptions that cannot generalize to most situations.

6 Conclusion

In this paper, we proposed a general stopping criterion for crowdsourced ranking. We demonstrated the robustness of our method in different situations, including subjective or objective tasks, diverse inference modules or task assignment modules and different budget and tolerance parameter values.

Acknowledgement. Caihua Shan and Reynold Cheng supported by the Research Grants Council of HK (RGC Projects HKU 17229116, 106150091, and 17205115), HKU (Projects 104004572, 102009508, and 104004129), and the Innovation and Technology Commission of HK (ITF project MRP/029/18). Nikos Mamoulis has been co-financed by the European Regional Development Fund, Research–Create–Innovate project "Proximiot" (T1EDK-04810). Xiang Li is the corresponding author.

References

1. Amazon mechanical turk. `https://www.mturk.com`
2. Figure eight. `https://www.figure-eight.com`
3. Moving average. `https://en.wikipedia.org/wiki/Moving_average`
4. Adelsman, R.M., Whinston, A.B.: Sophisticated voting with information for two voting functions. J. of Economic Theory **15**(1), 145–159 (1977)
5. Bradley, R.A., Terry, M.E.: Rank analysis of incomplete block designs: I. the method of paired comparisons. Biometrika **39**(3/4), 324–345 (1952)
6. Busa-Fekete, R., Szorenyi, B., Cheng, W., Weng, P., Hüllermeier, E.: Top-k selection based on adaptive sampling of noisy preferences. In: ICML (2013)

7. Chai, C., Fan, J., Li, G.: Incentive-based entity collection using crowdsourcing. In: ICDE. pp. 341–352. IEEE (2018)
8. Chen, X., Bennett, P.N., Collins-Thompson, K., Horvitz, E.: Pairwise ranking aggregation in a crowdsourced setting. In: WSDM. pp. 193–202. ACM (2013)
9. Chen, X., Chen, Y., Li, X.: Asymptotically optimal sequential design for rank aggregation. arXiv preprint arXiv:1710.06056 (2017)
10. Davidson, S.B., Khanna, S., Milo, T., Roy, S.: Using the crowd for top-k and group-by queries. In: ICDT. pp. 225–236. ACM (2013)
11. Eriksson, B.: Learning to top-k search using pairwise comparisons. In: Artificial Intelligence and Statistics. pp. 265–273 (2013)
12. Guo, S., Parameswaran, A., Garcia-Molina, H.: So who won?: dynamic max discovery with the crowd. In: SIGMOD. pp. 385–396. ACM (2012)
13. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American statistical association **58**(301), 13–30 (1963)
14. Karger, D.R., Oh, S., Shah, D.: Efficient crowdsourcing for multi-class labeling. SIGMETRICS **41**(1), 81–92 (2013)
15. Khan, A.R., Garcia-Molina, H.: Hybrid strategies for finding the max with the crowd: technical report. Tech. rep., Stanford InfoLab (2014)
16. Kou, N.M., Li, Y., Wang, H., U, L.H., Gong, Z.: Crowdsourced top-k queries by confidence-aware pairwise judgments. In: SIGMOD. pp. 1415–1430. ACM (2017)
17. Li, G., Wang, J., Zheng, Y., Franklin, M.J.: Crowdsourced data management: A survey. TKDE **28**(9), 2296–2319 (2016)
18. Lu, T., Boutilier, C.: Learning mallows models with pairwise preferences. In: ICML. pp. 145–152 (2011)
19. Metropolis, N., Ulam, S.: The monte carlo method. Journal of the American statistical association **44**(247), 335–341 (1949)
20. Negahban, S., Oh, S., Shah, D.: Iterative ranking from pair-wise comparisons. In: Advances in neural information processing systems. pp. 2474–2482 (2012)
21. Pfeiffer, T., Gao, X.A., Chen, Y., Mao, A., Rand, D.G.: Adaptive polling for information aggregation. In: AAAI (2012)
22. Pomerol, J.C., Barba-Romero, S.: Multicriterion decision in management: principles and practice, vol. 25. Springer Science & Business Media (2012)
23. Raykar, V., Agrawal, P.: Sequential crowdsourced labeling as an epsilon-greedy exploration in a markov decision process. In: Artificial intelligence and statistics. pp. 832–840 (2014)
24. Shan, C., Mamoulis, N., Li, G., Cheng, R., Huang, Z., Zheng, Y.: T-crowd: Effective crowdsourcing for tabular data. In: ICDE. pp. 1316–1319 (2018)
25. Shan, C., Mamoulis, N., Li, G., Cheng, R., Huang, Z., Zheng, Y.: A crowdsourcing framework for collecting tabular data. TKDE (2019)
26. Thurstone, L.L.: A law of comparative judgment. Psychological review **34**(4), 273 (1927)
27. Venetis, P., Garcia-Molina, H., Huang, K., Polyzotis, N.: Max algorithms in crowdsourcing environments. In: WWW. pp. 989–998. ACM (2012)
28. Wang, J., Kraska, T., Franklin, M.J., Feng, J.: Crowder: Crowdsourcing entity resolution. PVLDB **5**(11), 1483–1494 (2012)
29. Wauthier, F., Jordan, M., Jojic, N.: Efficient ranking from pairwise comparisons. In: ICML. pp. 109–117 (2013)
30. Welinder, P., Perona, P.: Online crowdsourcing: rating annotators and obtaining cost-effective labels. In: CVPRW. pp. 25–32. IEEE (2010)
31. Zhang, X., Li, G., Feng, J.: Crowdsourced top-k algorithms: An experimental evaluation. PVLDB **9**(8), 612–623 (2016)