Crowdsourced Top-k Queries by Confidence-Aware Pairwise Judgments

Ngai Meng Kou[#1] Yan Li[#2] Hao Wang[*3] Leong Hou U[#4] Zhiguo Gong[#5]

[#]Department of Computer and Information Science, University of Macau, Macau SAR
[1]yb27406@umac.mo [2]yb57411@umac.mo [4]ryanlhu@umac.mo [5]fstzgg@umac.mo

[*]State Key Laboratory for Novel Software Technology, Nanjing University, China
[3]wanghao@nju.edu.cn

ABSTRACT

Crowdsourced query processing is an emerging processing technique that tackles computationally challenging problems by human intelligence. The basic idea is to decompose a computationally challenging problem into a set of human friendly microtasks (e.g., pairwise comparisons) that are distributed to and answered by the crowd. The solution of the problem is then computed (e.g., by aggregation) based on the crowdsourced answers to the microtasks. In this work, we attempt to revisit the crowdsourced processing of the top-k queries, aiming at (1) securing the quality of crowdsourced comparisons by a certain confidence level and (2) minimizing the total monetary cost. To secure the quality of each paired comparison, we employ two statistical tools, Student's t-distribution estimation and Stein's estimation, to estimate the confidence interval of the underlying mean value, which is then used to draw a conclusion to the comparison. Based on the pairwise comparison process, we attempt to minimize the monetary cost of the top-k processing within a Select-Partition-Rank framework. Our experiments, conducted on four real datasets, demonstrate that our stochastic method outperforms other existing top-k processing techniques by a visible difference.

1. INTRODUCTION

Recently crowdsourcing is employed to process a variety of database queries, including `MAX` queries [12, 13, 21, 26, 38], `JOIN` queries [29, 40], and `top-`k queries [12, 13, 33]. In this work, we focus on the crowdsourced top-k queries for a collection of data items, where humans are involved in deciding the ordering of items. The crowdsourced top-k queries are particularly helpful in ranking *computer hostile* but *human friendly* items. For instance, finding the best translations of a sentence for training is an emerging requirement in machine translation. It is a difficult task for computer since the judgment relies on advanced natural language skills. However, humans can easily point out the better translation of two candidates if they have corresponding knowledge. Thus Google Translate [1] filters the translations via crowdsourcing. Zaidan et al. [43] asked the crowd to rank the translations. The ranking result was then used to improve the automatic translator. Many other applications like Duolingo [2], Facebook [3], and Twitter [4] also collect top translations by crowdsourcing. Top-k ranking for some annual events is another typical application. For instance, finding the best 3 soccer players of the year is the ever interesting topic for soccer fans. The judgment may rely on multi-source data integration and perceptual cognizance, which is more suitable for asking human soccer fans than computing by machines. Crowdsourced top-k query can also be used in estimating adverse drug reactions (ADRs). Some ADRs can be life-threatening while the others may be allergic slightly which makes the severity gradation of ADRs. Gottlieb et al. [20] ranked ADRs according to the severity by crowdsourcing pairwise comparisons on Amazon Mechanical Turk and achieved good correlation between mortality rate associated with ADRs and their ranks. More potential applications of the crowdsourced top-k queries can be found in Appendix A.

To process a crowdsourced top-k query, one needs to judge between data items; there are several ways to perform such judgments. A straightforward way is to ask humans to *rank* all or part of the items and then return the best k items by aggregating received rankings [29, 33]. These methods need complex user interface and are inconvenient for the human workers. Another way is to ask humans to *grade* the items and then return the best k items in terms of their average ratings [29]. However, graded judgments are known to differ in scale across judges [39]. To make things worse for crowdsourcing, graded judgments are hard to calibrate (e.g., normalizing the scores of each worker for fairness) as a worker may grade only a partial set of the items. Therefore, recent crowdsourced top-k query processing [12, 13, 20, 44] focus on *pairwise judgments*, i.e., comparing two items at a time. In contrast to other alternatives, pairwise judgments are easier to answer and less prone to human error, requiring only *relative* preference judgments for paired items.

Figure 1 is an example to show the general idea of how to process the top-k query by pairwise judgments in this work. The process (shown in the box) works in several rounds and in each round there are 2 phases. The upper phase is *ranking* and *scheduling* (cf. Section 5), which decides how to rank the items and which pairs should be compared in this round. To perform a *comparison* between a pair of items, a set of identical *microtasks* are sent to independent workers to collect

SIGMOD '17, May 14–19, 2017, Chicago, Illinois, USA.

 ISBN 978-1-4503-4197-4/17/05…$15.00

DOI: http://dx.doi.org/10.1145/3035918.3035953

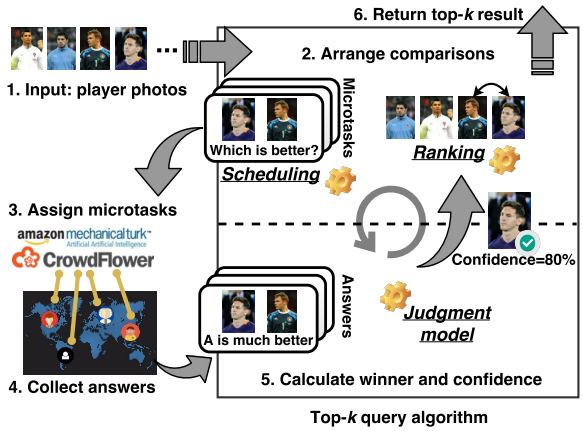


Figure 1: Query the top-k soccer players

the judgments. In the lower phase, the collected answers are fed into a *judgment model* (cf. Section 3) to decide the winner of a comparison as well as a confidence of the decision (i.e., the probability that the decision is correct). The two phases iterate until top-k items are found.

Determining the number of judgments (called *workload* hereinafter) needed for a pair of items is a hard problem. For instance, we probably need a large workload (e.g., 1,000 microtasks) to decide the better football player of Lionel Messi and Cristiano Ronaldo. In contrast, the workload needed for Lionel Messi and Anthony Martial should be substantially smaller (e.g., 30 microtasks).[1] If a fixed workload is used for both cases, it is either unthrifty or insufficient. To address this problem, Busa-Fekete et al. [8] proposed to estimate the workload from *pairwise binary judgments* (i.e., "yes or no" answers). However, their solution demands large workloads as the *confidence intervals*[2] [23] derived from binary samples are in general not tight enough. We will discuss and evaluate the estimation models in Section 3.2.

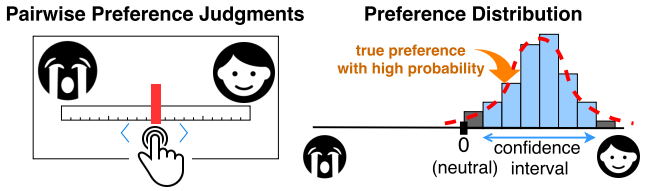


Figure 2: Pairwise preference judgments

In this work, we propose a new judgment model, entitled *pairwise preference judgment*, to tackle the problem discussed above. Our model can be viewed as a hybrid of graded judgments and pairwise binary judgments. Figure 2 demonstrates an example interface to collect pairwise preference judgments: a sliding bar is used to weigh the preference for a pair of items. Comparing to graded / pairwise binary judgments, pairwise preference judgments can derive tighter confidence intervals such that fewer workloads are needed to make judgments (Section 3.2). Table 1 briefly summarizes the features of three judgment models. Note that the relative order is easy to give (similar to that of the pairwise binary judgment), but the preference level is more difficult to grade (similar to that of the graded judgment). Therefore, the error of the pairwise preference judgment is moderate as compared with other two judgment models.

Table 1: Features of different judgment models

Model	Target	Pref.	Error	Workload per target
Graded	item	absolute	high	unknown
Binary	item pair	relative	low	large
Preference	item pair	relative	moderate	small

For ease of discussion, we assume normally distributed preferences[3] as in many existing solutions [6, 10, 34, 36]. Based on this assumption, we can readily derive the preference distribution between a pair of items from their judgments (see Figure 2). Given a set of judgments and a confidence level $1-\alpha$, the confidence interval of the preference mean can be estimated by statistical tools such as Student's t-distribution [23] and Stein's estimation [35]. To distinguish a pair, we simply check the lower and the upper bounds of the interval against a neutral value. For example, if the lower bound of the confidence interval with $\alpha = 0.05$ is larger than 0, then we are more than 95% confident that the right-hand side item is better than the left-hand side one (see Figure 2). In other words, we can stop comparing this pair if 95% confidence is adequate for our application.

Given the confidence-based judgment model, our objective is to return the top-k items such that the monetary cost (i.e., the total workload) is minimized. Given N items, the best known workload complexity of finding the top-k items is $O(Nw \log k)$ by heap sort and $O(Nw + kw \log N)$ by tournament tree, where w is the expected workload for a paired comparison. These approaches overlook a property that the workload needed for a pair of items should be inversely proportional to their distance in the (unknown) true total order. As an example, in Figure 3 the workload needed for (o_a, o_b) is likely less than (o_b, o_c).

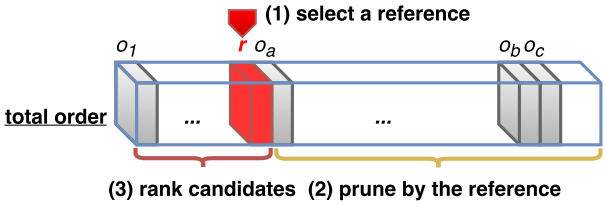


Figure 3: Select-Partition-Rank (SPR) top-k processing

With such considerations, our solution aims at avoiding the judgments between pairs that are close to each other in the underlying total ordering (i.e., pairs of items that are relatively difficult to distinguish). We develop a Select-Partition-Rank (SPR) framework that consists of (1) selecting a representative reference r, (2) partitioning items based on

[1] Martial is the most expensive teenager in football history.

[2] The confidence interval of an unobserved variable with confidence level $1-\alpha$ means that the variable falls into the interval with probability $1-\alpha$.

[3] For clarity, this assumption is used to derive tighter confidence interval when judging a pair of items. We can adopt other statistical models, e.g., Hoeffding's inequality, if the preference is not normally distributed.

their relative ordering with respect to r, and (3) ranking *some* of the items (related to the query) via sorting. With a good reference item r, the partitioning step is effective to avoid unpromising paired judgments. To the best of our knowledge, our work is the first to answer the crowdsourced top-k queries by an optimization (objective: minimizing the monetary cost) subject to a quality guarantee (constraint: confidence-aware pairwise comparison). The confidence-aware setting makes the optimization problem nontrivial since the workloads are varying on different pairs of items.

In the remainder of the paper, we first discuss related work in Section 2. We then introduce the pairwise preference judgment model and its superiority in Section 3. We give our objectives and analyze baseline solutions in Section 4. We discuss and analyze our Select-Partition-Rank framework in Section 5. Our experiments and conclusion can be found in Section 6 and Section 7. For ease of discussion, notation used thoroughly in this paper is listed in Table 2.

Table 2: Notation

Notation	Description
$o_i \in \mathcal{O}$	An item in the item set $\mathcal{O}$
$o_i^* \in \mathcal{O}$	The i-th item in the total order of $\mathcal{O}$
$\mathcal{O}^*$	Top-k item set
$s(o_i)$	Underlying score of item o_i
$v(o_i, o_j)$	Pairwise preference judgment between o_i and o_j
$V_{i,j}$	Bag of preferences $\{v_1(o_i, o_j), \ldots, v_{w_{i,j}}(o_i, o_j)\}$
$\textsc{Comp}\,(o_i, o_j)$	Comparison process based on $V_{i,j}$
$w_{i,j}$	Workload of $\textsc{Comp}\,(o_i, o_j)$ (i.e., size of $V_{i,j}$)
$1-\alpha$	Confidence level

2. RELATED WORK

Several existing studies involve crowdsourced ranking and top-k queries. Venetis et al. studied max queries (i.e., top-1 queries) over a set of items, with concerns on answer quality, monetary budget and time cost (i.e., the number of algorithmic steps to return the max item) [38]. In each judgment, a human worker was asked to identify the best out of several items. Strategies for partitioning items and recovering the max were then analyzed with respect to different human error models. Polychronopoulos et al. studied top-k queries over a set of data items [33]. Human workers were paid to rank small subsets of items, and then these ranked lists were used to determine the global top-k list via median-rank aggregation [16]. Instead of max or top-k queries, Marcus et al. aimed at sorting a given set of data items [29]. Two types of judgments were considered: one asked the workers to provide rankings of several items, and the other requested explicit ratings on items along, for example, a seven-point Likert scale [27]. Recently, Matsui et al. considered worker quality in such sorting problems [30]. They aggregated crowdsourced partial rankings based on Spearman's distance [14]. The above-mentioned studies are all different from our work, in that their methodologies are not based on pairwise comparisons, and therefore their solutions are in general not applicable to our problem.

Guo et al. investigated answering max queries by hiring human workers for pairwise comparisons [21]. Specifically, given a set of items and a budget, Guo et al. aimed at discovering the best item with the highest confidence out of the budget, to which end they took a marginal perspective toward the problem and managed to maximize the gain out of any additional budget. In their work, feedbacks from human workers were collected to form a vote matrix, and then maximum likelihood techniques (e.g., Kemeny ranking [25]) and graph-based heuristics (e.g., PageRank [32]) were employed to infer the best item. Davidson et al. as well considered such max queries [12, 13]. Pairwise votes from human workers, via a majority voting mechanism, were used to construct a tournament tree. With a random permutation of data items as leaves, the tree was tailored such that at the bottom levels each question was asked only once whereas at the top levels the number of questions for each comparison increased as the level approached the top. Under a proper error model, Davidson et al. showed that their solution might strike a balance between quality and cost. Gottlieb et al.'s [20] method outsourced random pairwise comparisons and inferred an initial order. By excluding ties and easy pairs, another set of random pairwise comparisons were asked. The global ranking was the one minimized the conflicts in the collected judgments.

Besides simple strategies such as majority voting, advanced models were developed to draw a conclusion from multiple votes for comparing a pair of data items. Thurstone proposed a model in which scores of items were assumed to be Gaussian random variables with known deviations but unknown means [36]. Given votes on two such items, Thurstone's model is able to estimate the mean difference of their scores, based on which one item can be inferred superior to the other. Bradley and Terry [7] and Luce [28] proposed what is now known as the Bradley-Terry-Luce (BTL) model. The model assumes that item scores are Gumbel, instead of Gaussian, random variables. Similar to Thurstone's model, the BTL model is also able to rank items based on votes.

Based on the BTL model, Chen et al. proposed solutions for aggregating votes on pairwise comparisons into a global ranking [9]. Similar to Guo et al. [21], they took a marginal perspective to solve the problem, aiming at finding (i) the best next pair of items to compare and (ii) the best human worker to do the comparison. In particular, Chen et al. considered crowdsourcing in an active learning scenario: reliability of each individual worker was estimated using known facts and was then incorporated into a maximum likelihood aggregation toward the gold-standard global ranking. Ye and Doermann considered both pairwise comparisons and absolute ratings for the sorting problem [41], where Thurstone's model was used to find the final winner from the votes for a paired comparison. Khan and Garcia-Molina [26] proposed a hybrid strategy that combines the graded and pairwise judgments to find the max item. Specifically, the graded judgment is used to filter out unpromising items and the pairwise judgment is used to rank the remained items. We share the intuition that the pairwise judgment is a better model to distinguish close items.

Busa-Feketes et al. [8] proposed to estimate the confidence interval of the mean score based on binary votes and processed the top-k queries by a preference-based racing algorithm. However, the objective of the algorithm was not to minimize the monetary cost.

These statistical models (e.g., the Thurstone's model, the BTL model and the confidence interval) secure the query answering quality but overlook a fact that the monetary cost is a major concern in processing a crowdsourced query. In this work we ask the human workers to express their

preferences in pairwise comparisons (instead of purchasing binary votes), which are much more informative in nature. Based on the judgment model, our work aims at not only securing the query answering quality but also minimizing the monetary cost.

Ciceri et al. [11] studied the crowdsourced top-k query over uncertain data with prior knowledge that the score of each data item is uniformly distributed in a narrow numerical domain. By enumerating the possible orderings of items in a tree, the item pair that most significantly reduces the uncertainty is outsourced in each iteration. The scenario is different from ours that zero prior knowledge exists. It is intractable to find the appropriate item pair in the tree of exponential size.

There are a bunch of other studies that can be considered related to our work in general. Their focus were, however, clearly different from ours. For example, Venetis and Garcia-Molina [37] investigated the role of task difficulty in comparison tasks. Yi et al. [42] studied crowdsourced ranking in a recommender system scenario. Fan et al. [17] studied the task assignment problem in crowdsourcing and proposed an *i*Crowd framework which could adaptively assign microtasks to high quality worker by estimating the accuracy of her previous completed tasks. Amsterdamer et al. [6] proposed a general crowd mining framework to discover association rules from human workers. They approximated the sample distribution as a normal distribution and sampled from a fixed numerical domain. Moreover, Amsterdamer et al. assumed the worker answers are independent and increased the confidence of data pattern (rule) by collecting more samples.

3. JUDGMENT MODELS

3.1 Comparison Process and Workloads

For a human worker to compare items o_i and o_j, she provides a preference $v(o_i, o_j) \in [-1, 1]$, of which the sign indicates her judgment intention and the absolute value describes her strength towards that intention. The comparison process $\textsc{Comp}(o_i, o_j)$ for items o_i and o_j is to draw a conclusion from a bag of $w_{i,j}$ (the workload) human preference values $V_{i,j} = \{v_1(o_i, o_j), \ldots, v_{w_{i,j}}(o_i, o_j)\}$. Since the unit cost of collecting a human preference is fixed, the goal is to minimize the workload to save monetary cost.

We may assume that every data item o_j is associated with a hidden score $s(o_j)$ (higher is better), and a human preference $v(o_i, o_j)$ is in principle monotonically proportional to the difference of the scores, $\Delta s_{i,j} = s(o_i) - s(o_j)$. Although such $\Delta s_{i,j}$ is unknown, we trust the human workers in that their preference values truly reflect $\Delta s_{i,j}$. Put in another way, the actual (unknown) difference of scores, $\Delta s_{i,j}$, determines a Gaussian distribution of human preference values between data items o_i and o_j, i.e., $v(o_i, o_j) \sim \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$, where $\mu_{i,j}$ is proportional to $\Delta s_{i,j}$ and the variance $\sigma^2_{i,j}$ describes how difficult it is to choose between o_i and o_j.

Since the mean $\mu_{i,j}$ is a monotonically increasing function of $\Delta s_{i,j}$, to conclude $s(o_i) < s(o_j)$ (i.e., $o_i \prec o_j$) or $s(o_i) > s(o_j)$ (i.e., $o_i \succ o_j$) we only need to test $\mu_{i,j}$ against 0. In particular, during the comparison process we aim at minimizing the number of samples required and estimating $\mu_{i,j}$ with a predefined confidence level of $1 - \alpha$.

Intuitively, for more difficult comparisons, more samples are required to reach the confidence level. Statistical tools

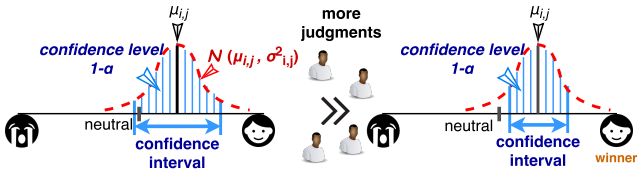


Figure 4: Comparison process

for *parameter estimation* can thus be applied on the bag of samples $V_{i,j}$ to find the confidence interval of $\mu_{i,j}$ with $1 - \alpha$ confidence. When the interval excludes 0, we are $1 - \alpha$ confident to make a judgment between a pair of items. Figure 4 illustrates a concrete example.

For ease of presentation, we adopt a closed symmetrical interval for the comparison process. However, the cumulative probability of half-closed confidence interval (excludes 0) can be larger than $1 - \alpha$ which improves the confidence of comparison. Our strategy can also extend to half-closed interval theoretically.

In this work, we use a statistical tool, Student's t-distribution estimation, to estimate the confidence interval of $\mu_{i,j}$ (with confidence level $1 - \alpha$). An alternative tool, Stein's estimation, is described in Appendix E.

Student's t-distribution estimation [23]. Let $n = w_{i,j}$ (number of samples), and $\bar{\mu}_n$ and S_n be the sample mean and sample standard deviation of $V_{i,j}$ respectively,

$$\bar{\mu}_n = \frac{1}{n}\sum_{\ell=1}^{n} v_\ell(o_i, o_j) \text{ and } S_n = \sqrt{\frac{\sum_{\ell=1}^{n}(v_\ell(o_i, o_j) - \bar{\mu}_n)^2}{n-1}}.$$

Then, a $1 - \alpha$ confidence interval is given by

$$\mu_{i,j} \in \left[\bar{\mu}_n - t_{\frac{\alpha}{2},n-1} \cdot \frac{S_n}{\sqrt{n}},\quad \bar{\mu}_n + t_{\frac{\alpha}{2},n-1} \cdot \frac{S_n}{\sqrt{n}}\right].$$

where $t_{\frac{\alpha}{2},n-1}$ indicates the right-tail probability of size $\frac{\alpha}{2}$ of the t-distribution with $n - 1$ degrees of freedom. Once the above interval does not contain 0 (the neutral value) then some conclusion can be made. For example, if $\bar{\mu}_n - t_{\frac{\alpha}{2},n-1} \cdot \frac{S_n}{\sqrt{n}} > 0$ then it is safe to conclude that $\mu_{i,j} > 0$ (and as a consequence $o_i \succ o_j$).

Algorithm 1 $\textsc{StudentComp}(o_i, o_j)$

B: budget for the pairwise comparison; I: minimum workload
Publish I microtasks and get $V \leftarrow \{v_1(o_i, o_j), \ldots, v_I(o_i, o_j)\}$
for $w = I + 1, I + 2, \cdots, B$ **do**
  Publish one more microtask and get $v_w(o_i, o_j)$
  $V \leftarrow V \cup \{v_w(o_i, o_j)\}$
  $\bar{\mu}_w \leftarrow$ sample mean of V
  $S_w \leftarrow$ sample standard deviation of V
  if $\bar{\mu}_w - t_{\frac{\alpha}{2},w-1} \cdot \frac{S_w}{\sqrt{w}} > 0$ **then return** $o_i \succ o_j$
  if $\bar{\mu}_w + t_{\frac{\alpha}{2},w-1} \cdot \frac{S_w}{\sqrt{w}} < 0$ **then return** $o_i \prec o_j$
return $o_i \sim o_j$ ▷ *indistinguishable under budget* B

As a general trend, with a larger workload we obtain tighter bounds on $\mu_{i,j}$. Therefore, in our crowdsourcing setting we take a progressive methodology: one additional feedback is retrieved from the crowd if current workload is insufficient to make any comparative judgment. Algorithm 1 shows the pseudo-code of the estimation process. In the pseudo-code, we add two additional parameters: B the bud-

get limiting the total workload, and I the minimum initial workload to overcome cold start. The value of I should be at least 30 according to common practice in statistics [23].

3.2 Why Preference Judgment?

To demonstrate the effect of different judgment models, we empirically study the number of microtasks needed to achieve certain confidence levels. Generally speaking, the confidence level can be achieved by fewer microtasks if the judgment model is more informative.

Our experiments are on an IMDb dataset.[4] The statistics of the dataset can be found in Section 6. We randomly pick 30 popular movies ($> 100,000$ votes) from the dataset.[5] The ground truth total order Ω of the movies is decided based on their means of votes (see Section 6). To simulate a pairwise preference judgment for a pair of movies o_i and o_j, we sample the ratings $s(o_i)$ and $s(o_j)$ from corresponding voting histograms (i.e., preference distributions) and set $v(o_i, o_j) = s(o_i) - s(o_j)$. To simulate a pairwise binary judgment, we set $v(o_i, o_j) = 1$ when $s(o_i) - s(o_j) > 0$ and $v(o_i, o_j) = -1$ when $s(o_i) - s(o_j) < 0$. If $s(o_i) = s(o_j)$, then this judgment is dropped since it is not identifiable.

Given the bag $V_{i,j} = \{v_1(o_i, o_j), \ldots, v_n(o_i, o_j)\}$, we evaluate the effect of three statistical tools on the comparison process $\textsc{Comp}(o_i, o_j)$ by varying the confidence level $1 - \alpha$. $\textsc{Comp}(o_i, o_j)$ stops when the estimated confidence reaches the predefined level. We record the actual accuracy and the workload for each pair of the movies (435 pairs in total), then report the average accuracy and workload. We say a comparison result (e.g., $o_i \prec o_j$ or $o_i \succ o_j$) is accurate if it follows the order in Ω. For each $\textsc{Comp}(o_i, o_j)$, we run 100 times and use $B = \infty$ and $I = 30$ for all estimation models.

Table 3: Accuracy and Workload of Different Judgment Models

Model	Est. by	435 pairs	Confidence level, $1-\alpha$ 0.95	0.98	0.99
Binary	Hoeffding	Work.	6,029.7	8,713.8	10,847.1
		Acc.	0.989	0.990	0.990
Preference	Student	Work.	639.2	1,510.6	1,987.0
		Acc.	0.992	0.996	0.998
	Stein	Work.	557.4	1,250.6	2,029.8
		Acc.	0.992	0.996	0.998
Model	**-**	**30 items**	**Workload 100**	**1,000**	**10,000**
Graded	-	Acc.	0.965	0.991	0.998

*Work. and Acc. are the average workload (number of microtasks) and the average accuracy of comparisons.

Table 3 reports the average performance of the comparison processes $\textsc{Comp}(o_i, o_j)$ on 435 pairs of movies. We first compare the pairwise preference judgment to the pairwise binary judgment considered in [8]. To achieve the same confidence level, the workload needed for the preference judgment is 5.34 to 10.81 times lower than that of the pairwise binary judgment since more information leads to tighter confidence interval estimations. This indicates that the preference model is a better model to return high quality ranking results with relatively low monetary cost. The performance of the Student's t-distribution estimation and the Stein's estimation is very similar so that any of them can be used to process $\textsc{Comp}(o_i, o_j)$ for the bag of preferences $V_{i,j}$. This advantage is further verified theoretically in Appendix D.

In addition, we show the performance of the graded judgment. It is hard to determine the workload of an item judgment, so we report the performance by varying the workloads in Table 3. The graded judgment model is not recommended since (1) the accuracy is not guaranteed and (2) a fixed budget is applied to every item.

Besides, the unit monetary cost of these judgments is more or less the same according to our empirical study on a real crowdsourcing platform CrowdFlower (cf. Appendix B). In this work, we recommend the pairwise preference judgment for comparing pairs of items since it not only secures the quality (the confidence level) but also requires lower monetary cost (fewer microtasks).

4. CROWDSOURCED TOP-K QUERIES

Given a set of N items $\mathcal{O} = \{o_1, o_2, \cdots, o_N\}$, we want to find the k best items $\mathcal{O}^* = \{o_1^*, o_2^*, \cdots, o_k^*\} \subset \mathcal{O}$ via pairwise comparisons, where o_i^* is the i-th best item. Assuming a unified cost per human worker per microtask, the *total monetary cost* (TMC) for finding the top-k items $\mathcal{O}^*$ depends on (i) the set of pairs to compare, $\mathcal{C}$, and (ii) the workload for each paired comparison in $\mathcal{C}$:

$$\text{TMC} = \sum_{(o_i, o_j) \in \mathcal{C}} w_{i,j}.$$

Our goal is to minimize TMC while maintaining a certain quality of $\mathcal{O}^*$. We carry out two objectives to that goal:

1. **Microtask-level cost minimization with confidence guarantee**, in which we manage to determine the workload $w_{i,j}$ of each comparison $\textsc{Comp}(o_i, o_j)$ subject to a confidence guarantee; and

2. **Query-level cost minimization**, in which we target towards minimizing the number of paired comparisons for finding $\mathcal{O}^*$.

We achieve the first objective via statistical estimations discussed in Section 3.1. As a matter of fact, the strength of a preference and the confidence level $1 - \alpha$ suggest the difficulty in distinguishing items o_i and o_j, and intuitively the workload needed should be proportional to the underlying difficulty of the corresponding comparison. The variance of effort in distinguishing different pairs of items makes the second objective hard to achieve. Traditional top-k algorithms assume that the comparison process $\textsc{Comp}(o_i, o_j)$ has unified cost in any pair of items but this assumption is no longer held in crowdsourced top-k queries. Thus simply minimizing the number of comparisons does not necessarily mean minimizing TMC. Note that, we assume that the answers from the same worker are independent over different comparisons. The dependency may exist in practice since a high quality worker should have a consistent personal standard. However this information is not considered in this work for simplicity. In the following, we briefly review traditional top-k query processing techniques and then discuss the infimum cost to answer a crowdsourced top-k query.

[4] http://www.imdb.com/interfaces

[5] The mean of graded judgments can be viewed as the ground truth score of an item when the volume is large enough.

4.1 Baseline Solution: Tournament Tree

Tournament trees TourTree are widely adopted in crowdsourced top-k query processing [12, 13]. TourTree first randomly groups items into $N/2$ pairs. Winners of paired comparisons are promoted to upper levels of the tournament tree, until the best item reaches the root. The 2nd best item can be identified by building a tournament tree over the items that ever directly lost to the best item. All k items can be found in a similar way progressively.

Let w be the expected workload of a comparison. The total workload of TourTree is $O(Nw + kw \log N)$. Note that, the expected workload w is very sensitive to the initial grouping of the items. For instance, the workload may become very large when all initial $N/2$ pairs of items are adjacent in the true total order.

4.2 Baseline Solution: Heap Sort

Another common technique to answer top-k queries is to initialize a min-heap with k random items (i.e., top-k *candidates*) and then sequentially test every other item against the top of the heap. Whenever an item is found better than the worst candidate in the heap, it expels the worst candidate from the heap and becomes a new top-k candidate. Essentially, this is a crowd-based heap sort over the items.

Let w be the expected workload of a comparison. The total workload of the above solution, HeapSort, is $O(Nw \log k)$. The performance of HeapSort is sensitive to the choice of initial top-k candidates.

4.3 Baseline Solution: Quick Selection

Quick selection [22] returns the k-th element in an unordered list, of which the execution process is similar to that of quick sort. The difference is that, after data partitioning, quick selection recurses into only one side, instead of both sides, of the data items. For example, to find the 5th item within a set of 100 items, quick selection may randomly pick a reference item and compare it with every other item. If after all the comparisons 30 items are found to be better than the reference, then quick selection recurses into these 30 items (discarding the rest 70) for the 5th item.

It is straightforward to adapt the quick selection algorithm to solve our crowdsourced top-k problem. The total workload of this method, QuickSelect, is $O\left(N^2w + kw \log k\right)$ in the worst case but $O(Nw + kw \log k)$ in the average case.

4.4 Infimum Cost: Linear Scan

None of the above baseline solutions (TourTree, HeapSort, and QuickSelect) is aware of the fact that paired comparisons may differ largely in difficulty. For example, the references selected in each recursion of QuickSelect may cause a large number of difficult comparisons, leading to very large TMC. This raises an interesting question: *What is the infimum cost (i.e., the minimum possible cost) to answer a crowdsourced top-k query?*

Let $\mathcal{W}(o_i, o_j)$ be the expected workload to judge between data items o_i and o_j, satisfying

$$\mathcal{W}(o_i, o_j) \propto \frac{1}{|s(o_i) - s(o_j)|}.$$

The following lemma calculates the infimum cost to find the top-k list.

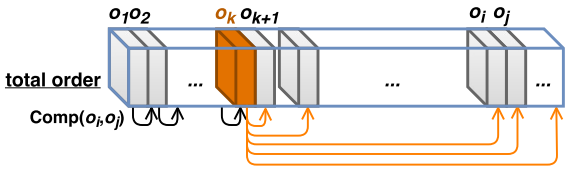


Figure 5: Infimum cost

Lemma 1. *Given a set of N items $\mathcal{O}$, the infimum cost to find the top-k list is*

$$TMC_{inf} = \sum_{j=1}^{k-1} \mathcal{W}(o_j^*, o_{j+1}^*) + \sum_{j=k+1}^{N} \mathcal{W}(o_j^*, o_k^*).$$

Proof. To find the top-k list it is sufficient and necessary to *confirm* (i) $o_1^* \succ o_2^* \succ \cdots \succ o_k^*$ and (ii) $o_k^* \succ o_j^*$ for $j > k$. In the best case, the former costs $\sum_{j=1}^{k-1} \mathcal{W}(o_j^*, o_{j+1}^*)$, comparing each adjacent pair of the top-k items.

To establish the latter, every o_j^* must be compared to at least one item $o \in \mathcal{O}$. The lemma is trivially true when every o_j^* is directly compared to $o = o_k^*$. Hence, we only need to consider the case when $o_k^* \succ o_j^*$ is established by *inference*.

Assume without loss of generality that $o = o_i^*$ $(k < i < j)$, and $o_k^* \succ o_i^*$ has already been confirmed. Confirming $o_i^* \succ o_j^*$ (thus inferring $o_k^* \succ o_j^*$) takes a cost of $\mathcal{W}(o_j^*, o_i^*) \geq \mathcal{W}(o_j^*, o_k^*)$. The inequality is due to the fact that $|s(o_j^*) - s(o_k^*)| \geq |s(o_j^*) - s(o_i^*)|$ as o_i^* is in between o_k^* and o_j^*. □

Figure 5 illustrates the query processing with the infimum cost. Note that the infimum cost in Lemma 1 is theoretically achievable: We could be lucky to pick the right reference o_k^*, thus after $N - k$ comparisons we prune $N - k$ non-result items. Then, when sorting the remaining k items, we may find that the items are already sorted.

5. SELECT-PARTITION-RANK (SPR)

Inspired by our discussion in Section 4.4, towards an efficient algorithm any non-result item $o \in \mathcal{O} \setminus \mathcal{O}^*$ should be excluded as early as possible by comparing o with o_k^*. Without any prior knowledge on o_k^*, this is of course only if we are very lucky to pick o_k^* by a wild guess. Fortunately, as will be shown later in Section 5.2, items succeeding but not far away from o_k^* may also have strong pruning power. We define the *sweet spot* as the set of items $\{o_k^*, o_{k+1}^*, \cdots, o_{\lceil ck \rceil}^*\}$ where $c > 1$ controls the size of the sweet spot ($ck \ll N$).

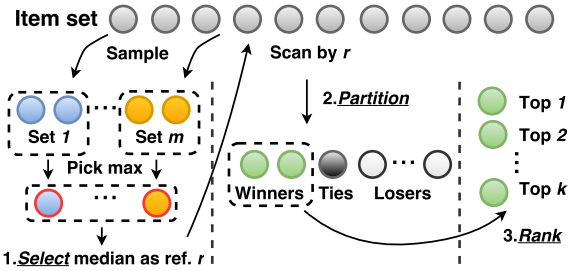


Figure 6: The framework of SPR

We develop a randomized algorithm, SPR, to solve our crowdsourced top-k problem. Figure 6 and Algorithm 2

Algorithm 2 SPR($\mathcal{O}, k$)

1: $R \leftarrow \emptyset$
2: Pick a reference item r ▷ *Section 5.1*
3: Compare every other item with r, partitioning $\mathcal{O}$ into winners (W_r), ties (T_r), and losers (L_r) ▷ *Section 5.2*
4: **if** $|W_r| < k$ **and** $|W_r \cup T_r| \geq k$ **then**
5: $R \leftarrow W_r$; Add random $k - |W_r|$ items of T_r into R
6: **return** $\mathcal{O}^* \leftarrow$ Sort(R) ▷ *Section 5.3*
7: **if** $|W_r \cup T_r| < k$ **then**
8: $R \leftarrow W_r \cup T_r \cup$ SPR $(L_r, k - |W_r| - |T_r|)$
9: **return** $\mathcal{O}^* \leftarrow$ Sort(R)
10: **return** $\mathcal{O}^* \leftarrow$ top-k items of Sort(W_r) ▷ $|W_r| \geq k$

sketch its general idea. SPR first manages to identify an item within the sweet spot with large probability via sampling (Figure 6:1; Line 2). By using that item as a reference, SPR then compares all the other items and prunes non-result ones. After the comparison processes, we get three different groups: *winners*, *ties*, and *losers*, holding those items that are superior to, indistinguishable from, and inferior to the reference, respectively (Figure 6:2; Line 3). Based on the partition, SPR efficiently finds the top-k results by sorting certain part of the partition (Figure 6:3; Lines 4-10).

5.1 Selecting a Reference

Given a set of N items, by a wild guess we have $\frac{j}{N}$ chance to hit an item r in the top-j set. This probability can be made much larger if r is the max item in a set of independent random samples $\mathcal{X} \subset \mathcal{O}$ (with putting back),

$$\Pr\{\, r \succeq o_j^* |\, x\} = 1 - \left(1 - \frac{j}{N}\right)^x. \tag{1}$$

Therefore the probability of r falling into or before the sweet spot is $\Pr\{o \succeq o_{ck}^* | x\}$. From both intuition and Equation (1) we know that this probability is monotonically increasing with respect to x, the total number of samples.

Recall that our goal is to find a reference item within the sweet spot. In other words, we want to find an item that can beat any item worse than o_{ck}^* but meanwhile is itself no better than o_k^*. That being the purpose, we take m independent sampling procedures, $\mathcal{X}_1, \mathcal{X}_2, \cdots, \mathcal{X}_m$, each with x samples, to get m independent max items $r_1, r_2, \cdots, r_m$. We argue that, when m is sufficiently large, the median of $r_1, r_2, \cdots, r_m$ stays in the sweet spot with high probability. Indeed, if r is the median of $r_1, r_2, \cdots, r_m$, then

$$\Pr\{\, o_k^* \succeq r \succeq o_{ck}^* |\, x, m\} = 1 - \sum_{i=\lceil \frac{m}{2} \rceil}^{m} \binom{m}{i} p^i (1-p)^{m-i} - \sum_{i=\lceil \frac{m+1}{2} \rceil}^{m} \binom{m}{i} q^{m-i} (1-q)^i$$

where $p = \Pr\{\, r_i \succeq o_{k-1}^* |\, x\}$ and $q = \Pr\{\, r_i \succeq o_{ck}^* |\, x\}$.

The following lemma shows that, by carefully tuning m and x, the probability $\Pr\{\, o_k^* \succeq r \succeq o_{ck}^* |\, x, m\}$ can be arbitrarily close to 1, thus a good reference is always achievable.

Lemma 2. *For arbitrarily small constant $\delta \in (0,1)$, there exists a pair of integers x and m, such that*

$$Pr\{\, o_k^* \succeq r \succeq o_{ck}^* |\, x, m\} > 1 - \delta.$$

Proof. Consider $x = \frac{N}{k} \ln \frac{1}{\gamma}$ for some $\gamma \in (0,1)$. We get

$$p = \Pr\{\, r_i \succeq o_{k-1}^* |\, x\} < 1 - \left(1 - \frac{k}{N}\right)^{\frac{N}{k} \ln \frac{1}{\gamma}} < 1 - \gamma + o(1),$$

$$q = \Pr\{\, r_i \succeq o_{ck}^* |\, x\} = 1 - \left(1 - \frac{ck}{N}\right)^{\frac{N}{k} \ln \frac{1}{\gamma}} > 1 - \gamma^c - o(1).$$

There exists γ such that $p < \frac{1}{2} < q$ as $c > 1$. Note that,

$$\sum_{i=\lceil \frac{m}{2} \rceil}^{m} \binom{m}{i} p^i (1-p)^{m-i} = \sum_{i=0}^{\lfloor \frac{m}{2} \rfloor} \binom{m}{i} (1-p)^i p^{m-i}.$$

Since $m(1-p) \geq \lfloor \frac{m}{2} \rfloor$, the Hoeffding inequality is applicable:

$$\sum_{i=0}^{\lfloor \frac{m}{2} \rfloor} \binom{m}{i} (1-p)^i p^{m-i} \leq \exp\left(-2 \frac{\left(m(1-p) - \lfloor \frac{m}{2} \rfloor\right)^2}{m}\right),$$

indicating $\sum_{i=\lceil \frac{m}{2} \rceil}^{m} \binom{m}{i} p^i (1-p)^{m-i} \to 0$ when $m \to +\infty$. Analogously, it can be shown that $\sum_{i=\lceil \frac{m+1}{2} \rceil}^{m} \binom{m}{i} q^{m-i} q^i \to 0$ when $m \to +\infty$. Hence, for any preset δ,

$$\Pr\{o_k^* \succeq r \succeq o_{ck}^* |x, m\} > 1 - \delta$$

if $x = \frac{N}{k} \ln \frac{1}{\gamma}$ for some proper γ and m is sufficiently large. □

Intuition of choosing the median. When $x = 1$, the expected rank of their median is $N/2$ (uniformly picked). When $x = N$ (i.e., all items are sampled), the maxima are always o_1^*. This implies that the expected rank will ascend from $N/2$ to 1 when x increases from 1 to N, thus with a proper x the median can approach the sweet spot.

Balancing quality and cost. In practice, as one may imagine, there should be a balance between the *quality of the reference* and the *cost of sampling*. In general, we would like to restrict the cost such that it does not dominate the cost of the entire SPR algorithm.

The sampling process takes $m(x-1)$ comparisons to find $r_1, r_2, \cdots, r_m$ and then any sorting or selection algorithm can be applied to find the median r. Since it takes $O(N)$ comparisons to partition the set of items (Line 3 of Algorithm 2), we solve the following optimization problem for a good choice of integers x and m:

$$\begin{aligned} \max_{x,m} \quad & \Pr\{\, o_k^* \succeq r \succeq o_{ck}^* |\, x, m\} \\ \text{s.t.} \quad & m(x-1) + \mathcal{C}(\mathcal{A}, m) \leq O(N), \end{aligned} \tag{2}$$

where $\mathcal{C}(\mathcal{A}, m)$ is an upper bound of the number of comparisons if algorithm $\mathcal{A}$ is used to find the median of m numbers. For example, it can be shown that $\mathcal{C}(\mathcal{A}, m) = \frac{1}{8}(3m^2 + m - 2)$ if algorithm $\mathcal{A}$ refers to bubble sort (cf. Appendix C).

Algorithm 3 summarizes our discussion in this section.

Algorithm 3 SelectReference($\mathcal{O}$)

1: $R \leftarrow \emptyset$
2: Solve Problem (2) for x and m
3: **for** $i \in \{1, 2, \cdots, m\}$ **do**
4: Generate $\mathcal{X}_i$, a set of x random items
5: Insert the max item of $\mathcal{X}_i$ into R
6: **return** r, the median of R

Complexity analysis. Let w be the expected workload to judge between two items. It takes $O(Nw)$ microtasks to find a good reference.

5.2 Reference-based Partitioning

With a proper reference r, SPR sequentially compares r with every other item. As a result, items $\mathcal{O} \setminus \{r\}$ are divided into three groups: *winners* W_r, *ties* T_r, and *losers* L_r, consisting of the items that are superior to, indistinguishable from, and inferior to r, respectively. Ties are mainly due to practical considerations: two items are considered tying if their relative ordering cannot yet be determined.

In Lemma 1 we have established that if $r = o_k^*$ then the cost for finding the top-k set (i.e., $W_r \cup \{o_k^*\}$) is minimal. Let us suspend the discussion on ties for the moment, assuming that every pair of items o_i and o_j can be eventually separated via a workload of $\mathcal{W}(o_i, o_j)$. We now show that, when r is in the sweet spot, it is still efficient in pruning non-result items.

LEMMA 3. *If the set $\mathcal{O}$ of N items are to be partitioned using $r = o_\ell^*$ $(\ell > k)$, then finding the top-k list requires an infimum cost of*

$$TMC_{inf}(o_\ell^*) = \sum_{j=1}^{k-1} \mathcal{W}(o_j^*, o_{j+1}^*) + \sum_{j=k+1}^{\ell} \mathcal{W}(o_j^*, o_k^*) + \sum_{j=\ell+1}^{N} \mathcal{W}(o_j^*, o_\ell^*).$$

Specifically, $TMC_{inf}(o_k^*) = TMC_{inf}$ as in Lemma 1.

PROOF. When using o_ℓ^* as a reference, to find the top-k list it is sufficient and necessary to confirm (i) $o_1^* \succ o_2^* \succ \cdots \succ o_k^*$, (ii) $o_k^* \succ o_j^*$ for $k < j \le \ell$, and (iii) $o_\ell^* \succ o_j^*$ for $j > \ell$. $TMC_{inf}(o_\ell^*)$ is then the sum of the minimal cost of each of the three confirmations. □

LEMMA 4. *When $k < \ell \ll N$, $TMC_{inf}(o_\ell^*)$ is monotonically increasing with respect to ℓ.*

PROOF. For any ℓ' such that $\ell < \ell' \ll N$, from Lemma 3 it is easy to infer that

$$\begin{aligned} & TMC_{inf}(o_\ell^*) - TMC_{inf}(o_{\ell'}^*) \\ = & \sum_{j=\ell'+1}^{N} \left(\mathcal{W}(o_j^*, o_\ell^*) - \mathcal{W}(o_j^*, o_{\ell'}^*)\right) \qquad (\le 0) \\ & + \sum_{j=\ell+1}^{\ell'} \left(\mathcal{W}(o_j^*, o_\ell^*) - \mathcal{W}(o_j^*, o_k^*)\right). \qquad (\ge 0) \end{aligned}$$

Since $\ell < \ell' \ll N$, the non-positive term clearly dominates the other, hence $TMC_{inf}(o_\ell^*) \le TMC_{inf}(o_{\ell'}^*)$. □

Lemmata 3 and 4 imply that a reference closer to o_k^* is preferable. Given an initial reference r, Algorithm 4 follows this implication to efficiently prune non-result items. Before any comparison we consider every pair of items (o, r) as a tie (Line 1). We then take an incremental approach to partition the set T_r of ties (Line 2-12). Despite special handling of cold starts, for every item in T_r we ask the crowd to provide *one* more preference feedback, if the budget still allows, and see whether any judgment can be made. As the iteration proceeds, W_r and L_r grow by accepting confirmed winners or losers (Lines 7-8).

Changing the reference. Algorithm 4 is designed to be *incremental* to defer difficult comparisons as much as possible. Such deferment is often beneficial because, as soon as the size of W_r reaches k, we have the opportunity to change for a better reference (Lines 9-12). Indeed, when $|W_r| = k$, the k-th best item in W_r (say r') satisfies $o_k^* \succeq r' \succ r$, thus according to Lemma 4 r' is a better reference than r. However, one should also notice that in this case previous efforts in comparing $o \in T_r$ with r could be wasted.

Algorithm 4 PARTITION$(\mathcal{O}, k, r)$

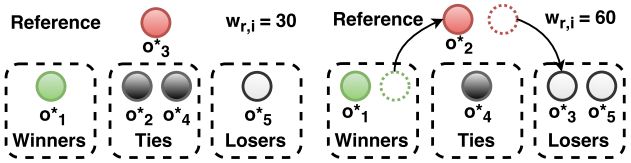
B: budget for the pairwise comparison; I: minimum workload
1: $W_r \leftarrow \emptyset$; $T_r \leftarrow \mathcal{O} \setminus \{r\}$; $L_r \leftarrow \emptyset$
2: **while** $\exists o \in T_r$ s.t. $|V_{o,r}| \ne B$ **do**
3: **for each** item $o \in T_r$ **do**
4: **if** $V_{o,r} = \emptyset$ **then** $\beta \leftarrow I$ **else** $\beta \leftarrow 1$
5: Generate β microtask(s) to compare o and r
6: Collect the β answer(s) into the bag $V_{o,r}$
7: **if** $o \succ r$ (resp. $o \prec r$) can be inferred from $V_{o,r}$ **then**
8: Remove o from T_r to W_r (resp. L_r)
9: **if** $|W_r| = k$ **then** ▷ *change reference (optional)*
10: $L_r \leftarrow L_r \cup \{r\}$
11: $r \leftarrow$ the k-th item in W_r
12: $W_r \leftarrow W_r \setminus \{r\}$
13: **if** $|W_r| < k$ **then** $W_r \leftarrow W_r \cup \{r\}$ ▷ *add the reference back*
14: **return** (W_r, T_r, L_r)

Table 4: Effect of changing the reference

Times	0	1	2	4	8	16
Work.	91310	88233	86498	**86372**	87718	88626

*Times and Work. are the maximum times of changing reference and the average workload (number of microtasks).

Table 4 shows effect of the changing the reference on IMDb dataset on default setting (cf. Section 6). In general, it is good to change the reference for several times but not too frequently. It is *not* the primary goal for Algorithm 4 to find o_k^*. Given the effort in picking the initial reference, there is a high probability for r to be in the sweet spot and thus is good already (Section 5.1). In addition, reference-based sorting over W_r may find the top-k list at a much lower cost (Section 5.3).



(a) Incremental partition (b) Reference refinement

Figure 7: Reference-based partitioning

Figure 7(a) is an example to demonstrate the idea of Algorithm 4 where this example contains 5 items and $k = 3$. Suppose o_3^* is picked as the initial reference. We compare all remained items to o_3^* by distributing microtasks to the crowd (Lines 5-6 of Algorithm 4). After collecting the bag of preferences from the crowd, we are able to partition these items into three sets, $W_r = \{o_1^*\}$, $T_r = \{o_2^*, o_4^*\}$ and $L_r = \{o_5^*\}$, based on the comparison process COMP(o_i, o_j) (Lines 7-8 of Algorithm 4). Since T_r is not empty, we keep distributing microtasks to the crowd until we add o_2^* to W_r. Now, the k-th item in W_r (i.e., o_2^*) is a better reference so that we can replace the reference by o_2^* (Figure 7(b)).

Complexity analysis. Reference-based partitioning compares r with all the other items, thus costs $O(Nw)$.

5.3 Reference-based Sorting

Given a small set of top-k candidates, a sorting procedure can find the top-k list. The key observation here is that, based on their pairwise comparisons with the reference item r, we may obtain a good initial order of those top-k candidates, hence any best-case linear sorting algorithm can find the top-k items efficiently.

In fact, for two top-k candidates o_i and o_j, we can compute the probability of o_i being better than o_j as

$$\begin{aligned}\Pr\{o_i \succ o_j\} &= \Pr\{s(o_i) > s(o_j)\} \\ &= \Pr\{s(o_i) - s(r) > s(o_j) - s(r)\}.\end{aligned}$$

Recall that, when comparing two items o and o', workers' feedbacks can be viewed as random samples drawn from a distribution, of which the mean $\mu_{o,o'}$ is proportional to $s(o) - s(o')$ and the standard deviation $\sigma_{o,o'}$ reflects the difficulty of the comparison. Hence,

$$\Pr\{s(o_i) - s(r) > s(o_j) - s(r)\} = \Pr\{\mu_{i,r} > \mu_{j,r}\}.$$

To compute this probability, we may use $V_{i,r}$ and $V_{j,r}$, the bags of preference values collected from the workers. For example, when the samples in $V_{i,r}$ and $V_{j,r}$ observe normal distributions, from Thurstone's calculation [36] we get

$$\Pr\{\mu_{i,r} > \mu_{j,r}\} \approx \Pr\{\hat{\mu}_{i,r} > \hat{\mu}_{j,r}\} = \Phi\left(\frac{\hat{\mu}_{i,r} - \hat{\mu}_{j,r}}{\hat{\sigma}_{i,j}}\right),$$

where $\hat{\sigma}^2_{i,j} = \hat{\sigma}^2_{i,r} + \hat{\sigma}^2_{j,r}$, assuming independence between the two distributions, and $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution.

Based on such probability calculations, an ordering between top-k candidates can be established, which is a rough estimation of the true ordering. Such an initial ordering may provide a good jumpstart to a best-case linear sorting algorithm. It is worth mentioning that most divide-and-conquer methods such as quick sort and merge sort are not good for this task, since they do not take any advantage of the fact that the input is almost sorted. In contrast, bubble sort could be a good choice. Given an almost sorted input, bubble sort takes near-linear time to adjust the ordering. In crowdsourcing scenarios, all human preference feedback can be stored and the results of comparisons are always *reusable*. Hence, although a pair of items could be compared multiple times during the execution of bubble sort, it is not a performance bottleneck when our goal is to save as much as possible the monetary cost.

Complexity analysis. Since r is in sweet spot (with high probability), the number of items to sort is fewer than ck. Therefore the monetary cost of sorting is $O((ck)^2 w)$ in the worst case but it is $O(ckw)$ in the best case. To sum up, the best case overall cost of SPR is $O(Nw + ckw)$.

5.4 System Accuracy Analysis

With the confidence of every pairwise judgment, we can analyze the system accuracy according to the procedure of the SPR framework. Incorrect judgments in selecting the reference will only affect the efficiency. However the top-k items may be falsely filtered from the result if the errors occur in the partitioning phase or the ranking phase. The probability of a top-k item o_i^* losing the comparison against the reference r is less than α. Therefore the expected number of top-k items is erroneously pruned in the partitioning phase is

$$\mathrm{E}[pruned] = \sum_{i=1}^{k} \Pr\{o_i^* \prec r\} \cdot 1 = \alpha k.$$

The remaining number of top-k items is $k(1-\alpha)$. Since there are ck items in total after partitioning, random k out of ck items can be returned, in which case the expected system precision is $\frac{1-\alpha}{c}$.

This is the lower bound of the expected system precision of the SPR framework since the ranking phase can be treated as a refinement step for returning higher quality result.

5.5 Latency Analysis

For different methods, the comparisons of different item pairs are outsourced in parallel only if they are independent, which follows the typical idea adopted in distributed algorithms.

Tournament tree. TourTree naturally supports parallel processing. To find the top-1 item o_1^*, each level of the tournament tree can be done in parallel, and there are in total $O(\log N)$ levels. To find the top-j item for $j \geq 2$, TourTree takes $O(\log \log N)$ rounds to find the best item within $O(\log N)$ items. The total latency of TourTree is thus $O(B \cdot (\log N + k \log \log N))$, where B is the maximum budget of a pairwise comparison.

Heap sort. It is difficult for HeapSort to run in parallel. Indeed, to build a min-heap of k items, every item in the initial binary tree should be moved down into one of its two subtrees. This can be done in a level-wise manner: To handle the j-th level of the tree, 2^{j-1} items can be moved down in parallel in $2j$ rounds, thus the latency of building the heap is $\sum_{j=1}^{\log k - 1} 2j = O\left(\log^2 k\right)$. After that, every other item should be evaluated *sequentially* against the heap, taking $(N-k)\log k$ rounds. Hence, the overall latency of HeapSort is $O\left(B \cdot \left(\log^2 k + (N-k)\log k\right)\right)$.

Quick selection. QuickSelect can be readily run in parallel. At each iteration, QuickSelect may compare the reference to all the other items in one batch. Therefore the expected latency of QuickSelect is $O(B \cdot (\log N))$.

SPR. SPR is parallelism-friendly. While selecting the reference, the m sampling processes (each sampling x items) can be done in parallel since they are independent (see Section 5.1). Then, $\log x$ rounds are required to find the best items, and another $\log m$ rounds are to sort the m candidates for selecting the median. The latency for selecting the reference is thus $\log x + \log m$. During partitioning, the reference can be compared with all the other items simultaneously, thus the latency is constant. Finally, all the items in W are sorted. Recall that $|W| \leq ck$. The total number of rounds for sorting is thus $O(ck)$ in the worst case and $O(1)$ in the best case (cf. best-case linear sorting in Section 5.3). To sum up, the best overall latency of SPR is $O(B \cdot (\log x + \log m))$.

Microtask-level batch processing. To compare a pair of items, there is often *latency* between the distribution of microtasks and the collection of user feedbacks. At one extreme, every time only 1 microtask is sent into the crowd. Assume that w microtasks are sufficient and necessary to make a judgment. In this way, the monetary cost is w, which is minimized, but there is also a latency of w, which is often undesirable. At the other extreme, we can simply send B microtasks into the crowd at once, minimizing the latency to 1 but increasing the monetary cost to B, which

is often wasteful. To strike a balance between the monetary cost and the latency, we distribute microtasks in *batches*. Specifically, if we distribute η microtasks at one time, then in the worst case we need B/η rounds to make a judgment. Then the constant B in the latency of all the methods can be reduced to B/η.

6. EXPERIMENTS

6.1 Datasets

Table 5: Dataset statistics

Dataset	#Items	Filtering criterion	Source
IMDb	1225	movies with $\geq$ 100,000 votes	Rating
Book	537	books with $\geq$ 50 votes	Rating
Jester	100	users voted all the jokes	Rating
Photo	200	N/A	Pairwise

IMDb. This dataset is collected from IMDb that contains 642,775 movies. Associated to each movie there is a histogram of the user votes and the *weighted rank* of a movie can be computed by

$$\text{weighted rank} = \frac{\#\text{votes}}{\#\text{votes} + K} \cdot \mu + \frac{K}{\#\text{votes} + K} \cdot C,$$

where K and C are the constants (where $K = 25,000$ and $C = 6.9$ based on the description in the IMDb dataset [5]) and μ is the mean of the votes. For each movie, we calculate its mean of the votes based on the weighted rank formula. The total order of the movies Ω is then determined by their mean values. To simulate a pairwise preference judgment for a movie pair o_i and o_j, we generate the ratings $s(o_i)$ and $s(o_j)$ based on their own voting histograms and set $v(o_i, o_j) = s(o_i) - s(o_j)$.

Book. This dataset is collected from Book Crossing, a free online book club used in [45]. Similar to what we do to IMDb, we simulate the judgment based on the histograms of books. The total order of the books Ω is determined by the mean of the histograms.

Jester. This dataset is collected from the Jester online joke recommender system developed by UC Berkeley [19]. We simulate a judgment result of any joke pair o_i and o_j by picking a random user from the dataset and setting $v(o_i, o_j) = s(o_i) - s(o_j)$ where $s(o_i)$ and $s(o_j)$ are the rates given by the picked user. The total order of the jokes Ω is determined by the mean of the joke ratings.

Photo. This dataset contains a set of 200 campus photos (one photo per university) and we build a judgment database $\mathcal{D}$ by the workers on a real crowdsourcing system CrowdFlower. For each judgment record in $\mathcal{D}$, we ask a worker to compare two campus photos and to rate her preference on an eight-point Likert scale [27]. The monetary cost is 0.1 US cent per question. To secure the database quality, we collect at least 10 judgment records for each pair of campus photos. To simulate a judgment, we randomly sample one record from $\mathcal{D}$ of the corresponding pair.

6.2 Experiment Settings

All methods were implemented in Java compiled by JDK 1.8. The experiments were conducted on a machine with Intel 3.4GHz i7-2600 CPU and 3.17GB memory, running Windows 7 32-bit OS. Investigated parameters and their evaluated ranges are listed in Table 6. Unless otherwise specified, in each experiment we vary one parameter and set the remaining ones to their defaults (shown in bold).

Table 6: Experiment parameters

Parameter	Values
Number of items, N	25, 50, 100, 200, 400, 800, **All**
Query parameter, k	1, 5, **10**, 15, 20
Confidence level, $1-\alpha$	0.8, 0.85, 0.9, 0.95, **0.98**
Pairwise comparison budget, B	(I=)30, 100, 200, 500, **1000**, 2000, 4000
Sweet spot range, c	1.25, **1.50**, 1.75, 2.00

In the experiments, the performance factors we evaluated include (1) the total monetary cost (TMC), (2) the query latency, and (3) the accuracy of the top-k result. TMC is the total number of microtasks we need for a query. The query latency is the number of the batch iterations we need for a query. As explained in Section 5.5, the microtasks are distributed in batches. We set the batch size η to 30 in all experiments. The accuracy of the result is measured by Normalized Discounted Cumulative Gain (NDCG) [24] that is a general metric for the ranking quality. All reported results are averaged over 100 runs.

Table 7: TMC of confidence-aware methods

TMC	SPR	TourTree	HeapSort	QuickSelect	PBR
IMDb	**88,233**	177,231	114,190	334,938	1.6M
Book	**80,369**	175,280	115,382	319,498	2.2M
Jester	**35,371**	47,560	56,265	80,497	222,596
Photo	**30,989**	38,787	48,920	58,088	41,360

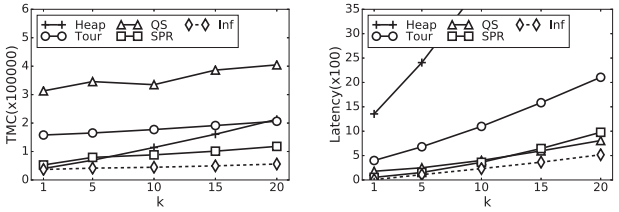
Table 7 shows the experiment results of the confidence-aware methods on default settings. SPR is the best method on all the datasets with respect to TMC. Preference-Based Racing algorithm (PBR) [8] considers a different scenario that the transitivity may not exist in the pairwise comparisons. This causes PBR to require much more microtasks so that we omit it from the remaining experiments.

In the following experiments, since the performance trends on all the datasets are similar, we only present the results on IMDb and Book. Other results can be found in Appendix F.

6.3 Scalability Experiments

Effect of k. Figure 8 shows the trend of TMC and query latency by varying k on IMDb and Book, respectively. SPR requires consistently less monetary cost than TourTree (Tour) and QuickSelect (QS). HeapSort (Heap) performs slightly better than SPR when k is small. Nevertheless, for HeapSort, the TMC increases significantly when k becomes larger and the query latency is much higher than that of the other methods. The query latency of SPR is less than that of TourTree and HeapSort because reference-based partitioning is parallelism-friendly. Benefiting from its binary partition, QuickSelect achieves roughly the same query latency with SPR, but its TMC is higher than the other methods on all the settings (except $k = 20$ on Jester).

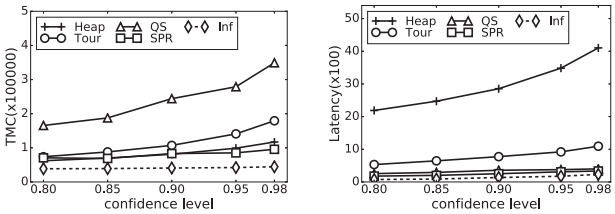
Effect of item cardinality. Figure 9 shows TMC and query latency by varying the number of items on IMDb and Book, respectively. QuickSelect, TourTree and HeapSort are more sensitive than SPR in terms of TMC. The trends of SPR on TMC and query latency are the closest



(a) TMC (IMDb) (b) Latency (IMDb)

(c) TMC (Book) (d) Latency (Book)

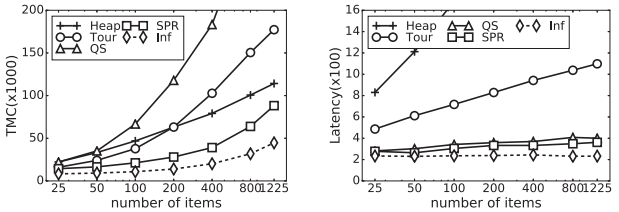
Figure 8: Effect of k



(a) TMC (IMDb) (b) Latency (IMDb)

(c) TMC (Book) (d) Latency (Book)

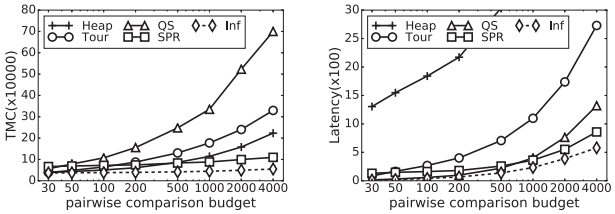
Figure 10: Effect of confidence level



(a) TMC (IMDb) (b) Latency (IMDb)

(c) TMC (Book) (d) Latency (Book)

Figure 9: Effect of item cardinality



(a) TMC (IMDb) (b) Latency (IMDb)

(c) TMC (Book) (d) Latency (Book)

Figure 11: Effect of B

to infimum (cf. Section 4.4), making SPR superior to its competitors.

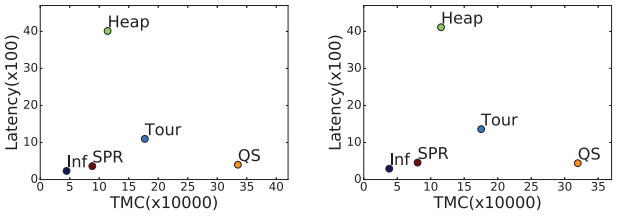
Effect of confidence level. Figure 10 shows that TMC and query latency of all the methods are increasing when the confidence level becomes higher. For each pair of items o_i and o_j, comparison process $\textsc{Comp}(o_i, o_j)$ needs more microtasks with a higher confidence level $(1-\alpha)$. SPR achieves less (vs. HeapSort and TourTree) or similar (vs. QuickSelect) query latency with lower cost in the experiments.

Effect of B. Figure 11 shows TMC and query latency by varying pairwise comparison budget on IMDb and Book. SPR is constantly close to infimum corresponding to TMC and query latency. As the monetary cost and query latency of all the methods monotonically increase with respect to the pairwise comparison budget B, TMC and execution rounds can be reduced by decreasing B. However, we show that a sufficient B is necessary to secure the *accuracy* in the next section.

Performance summary. Figure 12 summarizes the performance using the default settings on IMDb and Book. SPR is the only method that can approach infimum. Moreover, we conducted an interactive experiment on CrowdFlower to verify the performance of SPR in Appendix F.

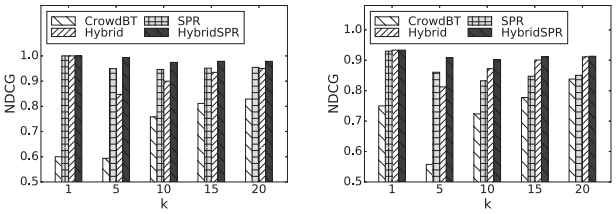
6.4 Accuracy

Figure 13 shows the accuracy by varying k, item cardinality, pairwise comparison budget and confidence level on IMDb. All the methods perform badly when $B \leq 100$, indicating that the accuracy can only be secured when B is sufficiently given. To balance the accuracy and cost, we set



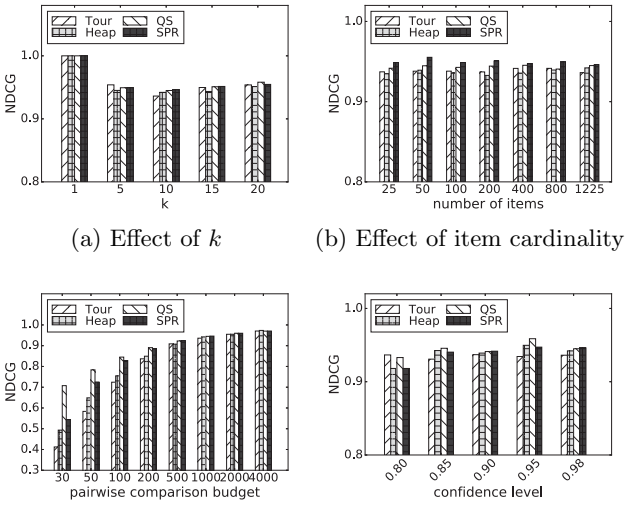
(a) IMDb (b) Book

Figure 12: Performance summary



(a) NDCG on IMDb dataset (b) NDCG on Book dataset

Figure 14: Non-confidence-aware methods



(a) Effect of k (b) Effect of item cardinality

(c) Effect of B (d) Effect of confidence level

Figure 13: Accuracy on IMDb dataset

$B = 1000$ by default. As a comparison among the methods, SPR has similar accuracy with its competitors but achieves lower TMC. Though QuickSelect achieves higher accuracy in some cases, its TMC is much higher.

6.5 Non-Confidence-Aware Methods

To further investigate the performance of SPR, we additionally compare SPR with two representative heuristic algorithms (with no confidence guarantee) of the crowdsourced top-k queries (recommended by a recent survey [44]). CrowdBT [9] decides the global ranking of items based on the pairwise binary judgment using the Bradley-Terry-Luce (BTL) model. Hybrid [26] is a hybrid method that employs both graded and pairwise binary judgment models. It first asks the crowd to grade the items and keeps the highest rating items (filtering phase). Then it outsources and compares every pair of the remained items (ranking phase). The scores of items are estimated based on both phases.

We compare SPR with Hybrid, CrowdBT, and an SPR based hybrid method HybridSPR on IMBb and Book. For fairness, Hybrid and CrowdBT are set to have the same budget to the TMC of SPR. The parameters are set following the suggestion of the original papers. The likelihood is optimized by BFGS [31] with 100 iterations. HybridSPR is a hybrid method that employs the filtering phase of Hybrid and ranks the remained items by SPR.

CrowdBT has relatively low NDCG since the given budget is not enough to distinguish a sufficient number of paired items. So the hidden item scores are not well estimated. The hybrid methods Hybrid and HybridSPR may have slightly better NDCG than SPR. This is because the item ratings are treated as the ground truth in this work so that the filtering phase can return a set of good candidates for the ranking phase. Our HybridSPR verifies the effectiveness of the confidence-aware technique as it consistently outperforms Hybrid. Besides, HybridSPR saves 10% monetary cost on average due to a more effective ranking phase.

7. CONCLUSION

In this work, we first discuss how the pairwise preference judgment helps to reduce the monetary cost of the comparison processes in crowdsourcing. According to our empirical study, it outperforms two popular judgment models as it achieves similar accuracy of comparisons by less number of microtasks. Based on the pairwise preference judgment, we develop a novel Select-Partition-Rank (SPR) framework by minimizing unnecessary tie pair comparisons. SPR not only reduces the cost but also secures the quality of the comparison processes. Our experiments show that SPR outperforms other competitors in terms of the monetary cost and the query latency.

In the future, we plan to further exploit the crowdsourced top-k queries by introducing other settings and optimization techniques. For instance, given some partial knowledge of the items [11], SPR could more effectively select a reference so that the overall cost can be further reduced. In addition, one possible optimization we left for future work is to outsource more judgments at microtask-level to achieve tighter intervals (e.g., we continue the comparison process even if the interval already excludes the neutral point), and infer the partial ranking based on the distinguishable intervals and their dependence.

We will also explore the statistical tools to enhance the performance of other crowdsourced queries, e.g., ranking, natural join, skyline, etc.

Acknowledgement

This work was supported by grants MYRG2016-00182-FST, MYRG2014-00106-FST, MYRG105- FST13-GZG, and MYRG 2015-00070-FST from UMAC RC, grants NSFC 61502548, 61432008, and 61503178 from NSF of China, grant BK20150587 from NSF of Jiangsu China, and grants FDCT/116/2013/A3 and FDCT/007/2016/AFJ from FDCT.

8. REFERENCES

[1] https://translate.google.com/.

[2] https://www.duolingo.com/.

[3] https://www.facebook.com/translations/.

[4] https://translate.twitter.com/.

[5] http://www.imdb.com/interfaces.

[6] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD*, pages 241–252, 2013.

[7] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[8] R. Busa-Fekete, B. Szörényi, W. Cheng, P. Weng, and E. Hüllermeier. Top-k selection based on adaptive sampling of noisy preferences. In *ICML*, pages 1094–1102, 2013.

[9] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, pages 193–202, 2013.

[10] W. Chu and Z. Ghahramani. Preference learning with gaussian processes. In *ICML*, pages 137–144, 2005.

[11] E. Ciceri, P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Crowdsourcing for top-k query processing over uncertain data. *IEEE Trans. Knowl. Data Eng.*, 28(1):41–53, 2016.

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

[13] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Top-k and clustering with noisy comparisons. *ACM Trans. Database Syst.*, 39(4):35:1–35:39, 2014.

[14] P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *JRSS: Series B*, pages 262–268, 1977.

[15] A. Doan, M. J. Franklin, D. Kossmann, and T. Kraska. Crowdsourcing applications and platforms: A data management perspective. *Proceedings of the VLDB Endowment*, 4(12):1508–1509, 2011.

[16] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001.

[17] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[18] B. Frei. Paid crowdsourcing. *Current State & Progress toward Mainstream Business Use, Smartsheet. com Report, Smartsheet. com*, 9, 2009.

[19] K. Y. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, 2001.

[20] A. Gottlieb, R. Hoehndorf, M. Dumontier, and R. B. Altman. Ranking adverse drug reactions with crowdsourcing. *Journal of medical Internet research*, 17(3):e80, 2015.

[21] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396, 2012.

[22] C. A. R. Hoare. Algorithm 65: Find. *Commun. ACM*, 4(7):321–322, July 1961.

[23] R. Hogg, E. Tanis, and D. Zimmerman. *Probability and Statistical Inference*. Pearson, 9 edition, 2013.

[24] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *TOIS*, 20(4):422–446, 2002.

[25] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

[26] A. Khan and H. Garcia-Molina. Hybrid strategies for finding the max with the crowd. Technical report, Technical report, 2014.

[27] R. Likert. A technique for the measurement of attitudes. *Arch. Psychol.*, 140:1–55, 1932.

[28] R. D. Luce. *Individual choice behavior : a theoretical analysis*. Wiley, 1959.

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

[30] T. Matsui, Y. Baba, T. Kamishima, and H. Kashima. Crowdordering. In *PAKDD*, pages 336–347, 2014.

[31] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[32] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: bringing order to the web. Technical Report 422, Stanford InfoLab, Stanford University, 1999.

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

[34] J. Snyder. Estimating the distribution of voter preferences using partially aggregated voting data. *Political Methodol.*, 13(1):2–5, 2005.

[35] C. Stein. A two-sample test for a linear hypothesis whose power is independent of the variance. *Ann. Math. Stat.*, 16(3):243–258, 1945.

[36] L. L. Thurstone. A law of comparative judgement. *Psychol. Rev.*, 34:273–286, 1927.

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

[38] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

[39] E. M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Inf. Process. Manage.*, 36(5):697–716, 2000.

[40] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.

[41] P. Ye and D. Doermann. Combining preference and absolute judgements in a crowd-sourced setting. In *Machine Learning Meets Crowdsourcing*, 2013.

[42] J. Yi, R. Jin, S. Jain, and A. K. Jain. Inferring users' preferences from crowdsourced pairwise comparisons: A matrix completion approach. In *HCOMP*, 2013.

[43] O. F. Zaidan and C. Callison-Burch. Crowdsourcing translation: Professional quality from non-professionals. In *ACL*, pages 1220–1229, 2011.

[44] X. Zhang, G. Li, and J. Feng. Crowdsourced top-k algorithms: An experimental evaluation. *Proceedings of the VLDB Endowment*, 9(8):612–623, 2016.

[45] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.

APPENDIX

A. POTENTIAL APPLICATIONS

Crowdsourced Q&A systems. The main challenge in the crowdsourced Q&A systems is to properly rank the answers from the crowd so that the top answers are valuable to the users. Yahoo! Answer and Stack Overflow are two popular crowdsourced Q&A systems for general users and professional programmers, respectively. In order to rank the crowdsourced answers, these systems ask users (who are interested in the question) to vote for their favourite answers. When looking for the best in rating list on Yahoo! Answer, we may find some very good answers with lower ratings. Actually, it is not strange because the crowd may just give comments without rating others' or they may just rate the several popular comments skipping over the rest. Consequently, the asker may miss the real best answer if she just browses the first few answers. To polish the ranking mechanism on those crowdsourced Q&A systems, our top-k algorithm can progressively refine the ranking by generating pairwise comparisons without leaving the good answers out.

Crowdsourced pattern recognition. Pareidolia[6] is an emerging topic in many social media web sites, e.g., FaceBook, Flickr, and Reddit, which asks users to post images of patterns that look like other things. The most common pattern type is to find things with faces[7]. Reddit collects its top image archive by users' votes, e.g., like or dislike. However, some good images may not get any votes because they are at the tail of a long list. Our proposed algorithm can right avoid this voting unfairness and find the top-k items with high accuracy guarantee.

Trip planner. Gogobot (https://www.gogobot.com/) is a crowdsourcing travel-planning platform that collects (suggests) the travel information from (to) the crowd. Given the set of user itineraries in a city, our crowdsourced top-k queries can be used to recommend the top-k most interesting travel plans for potential travelers.

Annual top-k ranking. Pairwise comparison can be particularly helpful in finding top items of some annual events. For instance, The Washington Post hosts the prize "Worst Year in Washington"[8] by asking people to compare paired candidates.

B. MICROTASK CATEGORIES AND UNIT MONETARY COST

Table 8: Crowdsourcing task categories

Type	Characteristics	Examples
Micro	Volume: very high Cost: very low	1. Label an image 2. Verify an address 3. Simple entity resolution
Macro	Volume: high Cost: low	1. Write a restaurant review 2. Test a new website feature 3. Identify a galaxy
Simple	Volume: low Cost: moderate	1. Design a logo 2. Write a term paper
Complex	Volume: single Cost: high	1. Build a website 2. Develop a software system

[6]https://en.wikipedia.org/wiki/Pareidolia
[7]http://www.thingswithfaces.net/
[8]http://www.washingtonpost.com/wp-srv/interactivity/worst-year-voting.html

The tasks in modern crowdsourcing platforms [15, 18] can be classified into four categories (shown in Table 8). To our understanding, both binary and preference judgments should be classified into the microtask category. Due to the task simplicity of this category, the monetary cost is relatively low (e.g., 0.1 US cent per question) and the popularity is high (in terms of volume) in the crowdsourcing platforms.

Table 9: Worker Satisfaction

Judgment	Overall	Ease of Job	Unit Time	#Microtasks
Binary	4.3/5	4.7/5	7.8s	300
Preference	4.3/5	4.2/5	10.3s	300

We empirically evaluate the difference between these two pairwise judgments in CrowdFlower[9]. We published 10 binary questions and 10 preference questions in CrowdFlower, each offering US$0.01. Each question was then answered by 30 workers. Table 9 shows the average worker satisfaction (5 is the highest) based on 17 (Binary) and 21 (Preference) worker feedback provided by CrowdFlower. In general, the workers are positive to both binary and preference questions. Although the binary judgments were reported to be easier, both types of questions can be answered in 11 seconds (on average) and the overall satisfaction was similar.

C. UPPER BOUND ANALYSIS FOR CHOOSING THE MEDIAN

In order to find the median of m numbers, a straightforward idea is to sort the entire set up to the median position. We can use many sorting algorithms to solve this problem. As an example, we give the upper bound analysis if $\mathcal{A}$ refers to bubble sort. The main idea is that we start from the last number and compare with its previous adjacent neighbour, swapping can only happen when it is superior to the neighbour. After $\lceil \frac{m}{2} \rceil$ iterations, we can finally obtain the median number.

$$\mathcal{C}(\mathcal{A}, m) = \sum_{i=1}^{\lceil \frac{m}{2} \rceil} (m-i) = (m-1+m-\lceil \frac{m}{2} \rceil) \cdot \lceil \frac{m}{2} \rceil \cdot \frac{1}{2}$$
$$\leqslant (m-1+\frac{m}{2}) \cdot \frac{m+1}{2} \cdot \frac{1}{2} = \frac{1}{8}(3m^2+m-2).$$

For other algorithms, we omit the details since similar analysis directly gives the upper bounds in Table 10.

Table 10: Upper bound of other sorting algorithms for choosing the medium

Algorithm	Upper Bound	Algorithm	Upper Bound
Selection	$\frac{1}{8}(3m^2+m-2)$	Heap	$m+2m\log\frac{m}{2}$
Merge	$3m\log m$	Quick	$\frac{1}{2}m(m-1)$

D. THEORETICAL ANALYSIS OF JUDGMENT MODELS

From the intuition and the experiments shown in Table 3, the pairwise preference judgment is likely a better judgment model than the pairwise binary judgment. We confirm this

[9]http://www.crowdflower.com

advantage by demonstrating the expected number of microtasks of the pairwise preference judgment until converge is less than that of the pairwise binary judgment.

Assume without loss of generality that the sample mean $\bar{\mu}_n > 0$. Following the assumptions in Section 3.1, we deduce the number of microtasks n of a given item pair o_i and o_j based on pairwise preference judgment model. We have the number of microtasks when

$$\bar{\mu}_n - t_{\frac{\alpha}{2},n-1} \cdot \frac{S_n}{\sqrt{n}} = 0 \Rightarrow n = \left(t_{\frac{\alpha}{2},n-1} \cdot \frac{S_n}{\bar{\mu}_n} \right)^2 .$$

We then consider the number of microtasks n_b based on the pairwise binary judgment model. In the pairwise binary judgment, a user preference $v(o_i, o_j)$ is classified into a binary choice $v^b(o_i, o_j) \in \{-1, 1\}$. Mathematically,

$$v^b(o_i, o_j) = \begin{cases} -1, & v(o_i, o_j) < 0 \\ 1, & \text{otherwise} \end{cases} .$$

Then the expected value of sample mean shifts from $\mu_{i,j}$ to $\tilde{\mu}_{i,j}$ where the shifted mean

$$\begin{aligned} \tilde{\mu}_{i,j} &= \mathrm{E}[v^b(o_i, o_j)] = 1 - 2\Pr\{v(o_i, o_j) < 0\} \\ &= 2\Phi\left(\frac{\mu_{i,j}}{\sigma_{i,j}}\right) - 1. \end{aligned}$$

Assuming the sample mean of the binary judgment is $\bar{\mu}_{n_b}$, $\Pr\{|\bar{\mu}_{n_b} - \tilde{\mu}_{i,j}| \le t\} \ge 1 - 2\exp\left(-\frac{n_b t^2}{2}\right)$ based on Hoeffding inequality. When $t < \tilde{\mu}_{i,j}$, we are $1 - 2\exp\left(-\frac{n_b t^2}{2}\right)$ confident to make the judgment between o_i and o_j. Then we replace t by $\tilde{\mu}_{i,j}$ and obtain

$$\begin{aligned} \Pr\{|\bar{\mu}_{n_b} - \tilde{\mu}_{i,j}| < \tilde{\mu}_{i,j}\} &> 1 - 2\exp\left(-\frac{n_b \tilde{\mu}_{i,j}^2}{2}\right) \\ &= 1 - \alpha. \end{aligned}$$

Thus, the number of microtasks n_b based on pairwise binary judgment model is

$$n_b = \frac{2}{\tilde{\mu}_{i,j}^2} \log \frac{2}{\alpha}. \tag{3}$$

Since there do not exist explicit formulation of $\Phi(\cdot)$ and $t_{\frac{\alpha}{2},n-1}$, we run a simulation of $n_b - n$ by Mathematica showing that $n_b > n$ on all values of $\mu_{i,j}$ and $\sigma_{i,j}$ in Figure 15.

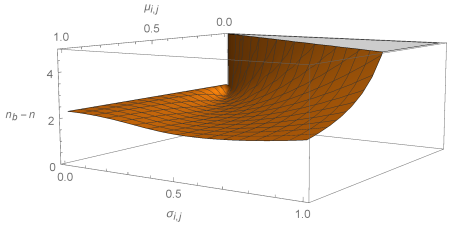


Figure 15: Analysis of $n_b - n$ by Mathematica

E. ADDITIONAL ESTIMATION TOOL

Stein's estimation [35]. Given a predefined interval width $2L$, Stein proposed a method to estimate the number of samples n needed to conclude $\mu_{i,j} \in [\bar{\mu}_n - L, \bar{\mu}_n + L]$ with confidence level $1 - \alpha$. In its original form, the estimation takes two stages:

1. Get y (Gaussian) samples and calculate S_y, the sample standard deviation. S_y is thus a rough estimation of the true standard deviation $\sigma_{i,j}$;
2. Compute $y' = S_y^2 \cdot L^{-2} \cdot t^2_{1-\frac{\alpha}{2},y-1}$. Then, $n = \max\{y, y'\}$ is the number of samples necessary to conclude $\mu_{i,j} \in [\bar{\mu}_n - L, \bar{\mu}_n + L]$ with confidence level $1 - \alpha$.

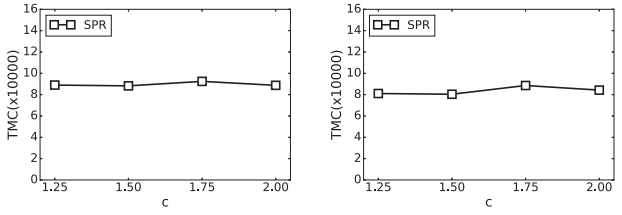
Recall that in our problem we simply want to find an interval $[\bar{\mu}_n - L, \bar{\mu}_n + L]$ that excludes 0 with as few samples as possible. That being the purpose, we transform Stein's original method into a progressive estimation on $\mu_{i,j}$.

Algorithm 5 SteinComp(o_i, o_j)

```
   B: budget for the pairwise comparison; I: minimum workload
1: Publish I microtasks and get V ← {v_1(o_i,o_j), ..., v_I(o_i,o_j)}
2: for w = I+1, I+2, ··· , B do
3:     Publish one more microtask and get v_w(o_i,o_j)
4:     V ← V ∪ {v_w(o_i,o_j)}
5:     μ̄_w ← sample mean of V
6:     S_w ← sample standard deviation of V
7:     L ← |μ̄_w| − ε                    ▷ ε is a small positive value
8:     if S_w^2 · L^{-2} · t^2_{1−α/2,w−1} > w then continue
9:     if μ̄_w < 0 then return o_i ≺ o_j
10:    if μ̄_w > 0 then return o_i ≻ o_j
11: return o_i ∼ o_j                   ▷ indistinguishable under budget B
```

As shown in Algorithm 5, for comparing a pair of data items o_i and o_j, we progressively retrieve human preference values $v_1(o_i, o_j), v_2(o_i, o_j), \cdots, v_w(o_i, o_j)$ until the number of samples is sufficient to make a judgment. During such a process, we dynamically change the width of the confidence interval, L (Line 7). Specifically, based on the sample mean $\bar{\mu}$ we always make L slightly smaller than $|\bar{\mu}|$ such that the interval $[\bar{\mu} - L, \bar{\mu} + L]$ is always "adjacent" to 0.[10] With this invariance, as soon as the workload is sufficient (Line 8), the algorithm terminates with a judgment (Lines 9-10).

F. ADDITIONAL EXPERIMENTS



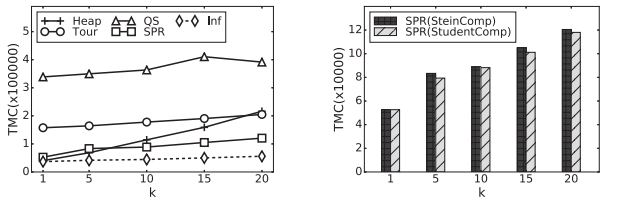
(a) TMC (IMDb) (b) TMC (Book)

Figure 16: Sweet spot range

Sweet spot range. As we mentioned in Section 5, sweet spot consists of items $o_k^*, o_{k+1}^*, \cdots, o_{ck}^*$ where c controls the range ($ck \ll N$). Figure 16 shows the TMC of SPR as a function of c. The monetary cost of SPR is stable when we vary c. Therefore we simply set $c = 1.5$ in the experiments.

Confidence estimation. We compare SteinComp and StudentComp using IMDb by varying k. Figure 17(a) is

[10] A small positive ε guarantees that the interval excludes 0.



(a) Effect of k (Stein) (b) Stein vs. Student (SPR)

Figure 17: Stein and Student comparison (TMC)

a reproduction of Figure 8(a) under the same settings but replacing StudentComp with SteinComp. Figure 17(b) demonstrates the difference between SteinComp and StudentComp. The results show that the performance of SteinComp and StudentComp are analogous. And the results are consistent on the other datasets. Therefore we simply select StudentComp as our default estimation method.

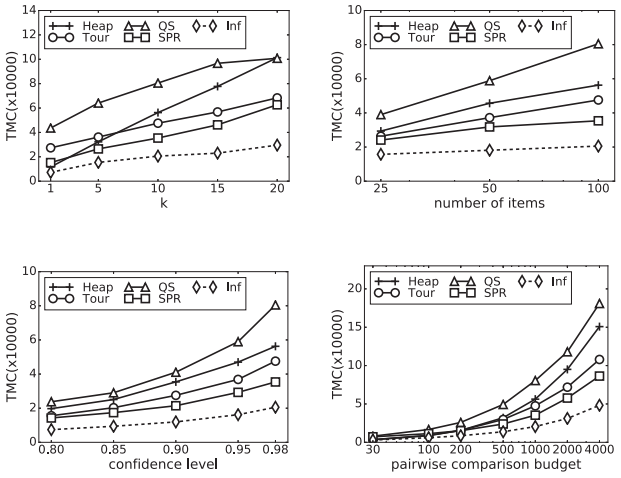


Figure 18: TMC experiments on Jester

Additional scalability experiments. Figures 18-21 show the TMC and query latency of the methods as a function of k, number of items N, confidence level $1-\alpha$ and pairwise comparison budget B, respectively, on Jester and Photo datasets. The overall trend is similar to the results on IMDb and Book (cf. Section 6.3).

Interactive Experiment. We also conducted an interactive experiment on another dataset **PeopleAge**[11]. We pick this dataset since it offers ground truth for the accuracy evaluation. PeopleAge contains a set of 100 different women photos from 1 year old to 100 years old. The query is to find the 10 youngest women. To save our monetary cost, we set $1-\alpha = 0.90$ and $B = 100$. SPR takes 6 hours 55 minutes to get the final result and costs 10.56 US dollars on CrowdFlower. The NDCG is 0.917. We also run the simulation on PeopleAge with the same settings. The TMC and the NDCG by simulation are 9570 (i.e. 9.57 US dollars) and 0.905, respectively. This experiment confirms that our simulations can reflect the real performance of our method.

[11]http://edouardjanssens.com/art/1-to-100-years/women/

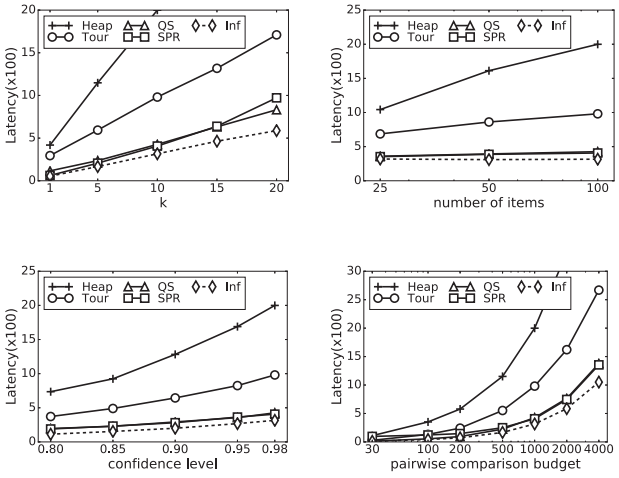


Figure 19: Latency experiments on Jester

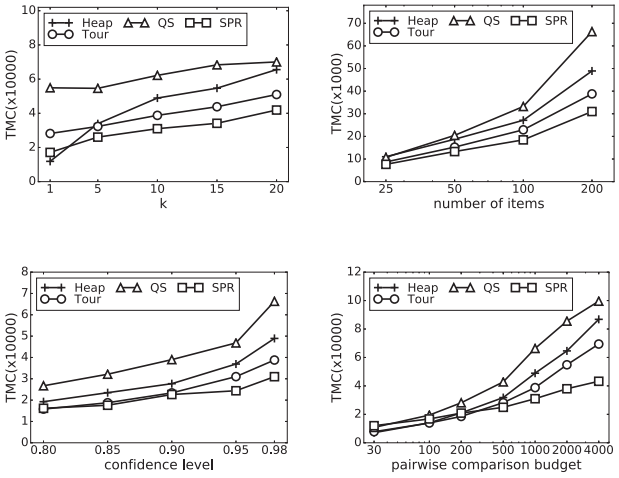


Figure 20: TMC experiments on Photo

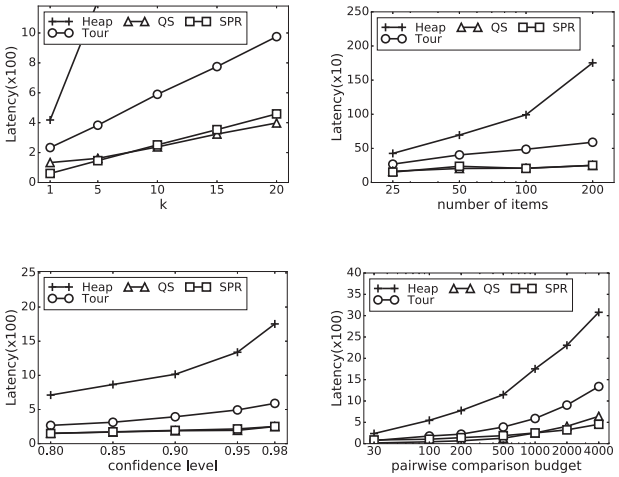


Figure 21: Latency experiments on Photo