This is the Pre-Published Version.
The following publication X. Lin, J. Xu, H. Hu and F. Zhe, "Reducing Uncertainty of Probabilistic Top-k Ranking via Pairwise Crowdsourcing," 2018 IEEE 34th International Conference on Data Engineering (ICDE), 2018, pp. 1757-1758 is available at https://doi.org/10.1109/ICDE.2018.00236

Reducing Uncertainty of Probabilistic Top-k Ranking via Pairwise Crowdsourcing

Xin Lin, Jianliang Xu, Haibo Hu and Zhe Fan

Abstract—Probabilistic top-k ranking is an important and well-studied query operator in uncertain databases. However, the quality of top-k results might be heavily affected by the ambiguity and uncertainty of the underlying data. Uncertainty reduction techniques have been proposed to improve the quality of top-k results by cleaning the original data. Unfortunately, most data cleaning models aim to probe the exact values of the objects individually and therefore do not work well for subjective data types, such as user ratings, which are inherently probabilistic. In this paper, we propose a novel pairwise crowdsourcing model to reduce the uncertainty of top-k ranking using a crowd of domain experts. Given a crowdsourcing task of limited budget, we propose efficient algorithms to select the best object pairs for crowdsourcing that will bring in the highest quality improvement. Extensive experiments show that our proposed solutions outperform a random selection method by up to 30 times in terms of quality improvement of probabilistic top-k ranking queries. In terms of efficiency, our proposed solutions can reduce the elapsed time of a brute-force algorithm from several days to one minute.

1 Introduction

Top-k ranking is an important query operator in many real-life applications such as decision support [40], data cleaning [11], and recommendation systems [41]. However, the data obtained from these applications are often uncertain due to various factors such as subjective nature of data, unreliable data sources, and limited equipment/machine precision. For example, the user ratings of a point-of-interest (e.g., restaurant or hotel) are diversified in nature; the features of a multimedia object extracted by some (fast and inexpensive) automated algorithm may not be 100% accurate. As such, processing top-k ranking queries on uncertain data, termed as probabilistic top-k queries, has drawn intensive research attention in recent years [15], [29], [42]. A probabilistic top-k query returns the set of highest-ranking objects with a probability associated with each query answer.

Possible World	Probability	Top-2 Objects
$W_1 = \{i_{11}, i_{21}, i_{31}\}$	0.024	$\{o_1, o_2\}$
$W_2 = \{i_{11}, i_{21}, i_{32}\}$	0.016	$\{o_1, o_2\}$
$W_3 = \{i_{11}, i_{22}, i_{31}\}$	0.096	$\{o_1, o_3\}$
$W_4 = \{i_{11}, i_{22}, i_{32}\}$	0.064	$\{o_1, o_2\}$
$W_5 = \{i_{12}, i_{21}, i_{31}\}$	0.096	$\{o_2, o_3\}$
$W_6 = \{i_{12}, i_{21}, i_{32}\}$	0.064	$\{o_2, o_1\}$
$W_7 = \{i_{12}, i_{22}, i_{31}\}$	0.384	$\{o_3, o_1\}$
$W_8 = \{i_{12}, i_{22}, i_{32}\}$	0.256	$\{o_1, o_2\}$

TABLE 1
Possible Worlds of Fig. 1

- *Xin Lin is with the Shanghai Key Laboratory of Multidimensional Information Processing, East China Normal University, Shanghai 200241, China, and with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong. E-mail: xlin@cs.ecnu.edu.cn.*
- *Jianliang Xu and Zhe Fan are with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong. E-mail: xujl@comp.hkbu.edu.hk and zhe.fan@foxmail.com. Zhe Fan is the corresponding author.*
- *Haibo Hu are with the Department of Electronic and Information Engineering, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. E-mail: haibo.hu@polyu.edu.hk.*

ID	Value	Prob
i_{11}	50	0.2
i_{12}	53	0.8

ID	Value	Prob
i_{21}	51	0.2
i_{22}	54	0.8

ID	Value	Prob
i_{31}	52	0.6
i_{32}	55	0.4

Fig. 1. Example of Uncertain Objects

Example: *Fig. 1 shows a toy example of probabilistic database, where three photos of Steve Jobs, regarded as probabilistic objects, are labeled by o_1, o_2, and o_3. Steve's ages might be estimated by using machine learning techniques (e.g., How-Old.net). Since such techniques cannot achieve 100% accuracy, the estimated ages might be provided as a list of probabilistic instances (see the tables in the lower part of Fig. 1). The instances of the same uncertain object are mutually exclusive and the sum of their probabilities is 1. Such a probabilistic database can be viewed as a set of possible worlds whose probabilities are the products of the probability of each instance. For example, the probability of the possible world $W_1 = \{i_{11}, i_{21}, i_{31}\}$ is $0.2 \times 0.2 \times 0.6 = 0.024$. Table 1 lists all possible worlds and their corresponding probabilities. To find the top-2 photos with the youngest ages, we combine all possible worlds with the same top-2 objects. For example, since $\{o_1, o_3\}$ is the top-2 photos for possible worlds W_3 and W_7 only, the probability that $\{o_1, o_3\}$ is a set of top-2 objects is $P(W_3) + P(W_7) = 0.48$.*

Although possible worlds resolve the issue of quantifying uncertainties over probabilistic data, the ambiguity brought by the uncertain data degrades the confidence and quality of top-k query results. In the above example, the highest probability of a top-2 object set, *i.e.*, $\{o_1, o_3\}$, is no more than 50%. This means that it has a higher chance of not being the genuine top-2 result than that of being so. It has been proved in the previous work

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Correctness of Photo Comparison	Correctness of Age Guessing (Deviation $\leq x$)					
	= 0	≤ 1	≤ 2	≤ 3	≤ 4	≤ 5
94%	6%	17%	28%	38%	47%	55%

TABLE 2
Pairwise Photo Comparison V.S. Age Guessing

[5] that partially reducing the uncertainty of the original data could improve the quality of final query results. In [22], data cleaning techniques are used to reduce the uncertainty of top-k query results. It adopts a *singleton cleaning* model, which probes the exact values of the objects individually. More sophisticated cleaning solutions harness additional resources, such as using redundant sensor devices [22]. However, these techniques cannot be applied to applications where exact values of individual objects are difficult to obtain, particularly in those domains where such values are usually subjective (*e.g.*, personal opinions). With the increasing popularity of crowdsourcing, an attractive alternative solution is to ask for help from the crowd or experts. By incorporating their feedbacks, the confidence level of top-k ranking would be improved with fewer possible worlds and reduced uncertainty.

Humans make more reliable decisions on comparing object pairs than rating individual objects [25]. Take Fig. 1 for example. Users might feel easier to verify the claim that "o_3 is younger than o_1" than to precisely guess their ages. As a showcase, we collected 600 photos in the AgeGuessing website [1] together with their ground-truth ages. We randomly crowdsourced 50 pairs of photos for age comparison. The accuracy of the crowd is 94%. Meanwhile, we crowdsourced the photos for direct age guessing. As shown in Table 2, only 6% answers exactly match the ground truth. Even if we allow a deviation of 5 years, only 55% answers match the ground truth. As a result, the accuracy of age comparison based on such data is only 78%. This suggests that the singleton cleaning model suffers from more noise and hence cannot work well in such applications. As such, we propose a novel *pairwise crowdsourcing* model to reduce the uncertainty of top-k query results by crowdsourcing pairs of objects to domain experts for comparison.[1] Let us revisit the example in Fig. 1 and Table 1. Suppose we post o_1 and o_3 to the crowdsourcing platform and get "$o_3 < o_1$", then only the possible worlds W_5 and W_7 will still hold. Consequently, the probability of $\{o_1, o_3\}$ being the top-2 result is increased to 0.8, which significantly improves the quality of the query result.

We argue that the pairwise crowdsourcing model can be used for many uncertain top-k applications. Some of them are listed below:

- **Point-of-Interest (POI) Ranking**: In POI databases, POIs (e.g., hotels, museums, and restaurants) are rated by users. As such, the scores assigned to each POI are probabilistic instances of this POI (e.g., 65% rate 5 and 35% rate 4). To improve the quality of top-k ranking results, we can crowdsource some pairs of POIs to experts for a comparison of their goodness.
- **Product Ranking**: In Web product reviews, product scores are computed from user comments by automated text mining methods. A product may be assigned with several scores, each with a different confidence level. Similar to POI ranking, the quality of the top-k ranking can be improved by asking the crowd whether one product is better than another.
- **Complex Decision Making**: In business management and social science, mathematical synthesis is adopted to model a judgement (e.g., choosing a leader from several candidates based on their experience, education, and charisma) as a complex system. Some state-of-the-art techniques (e.g., analytic hierarchy process [27]) use domain knowledge and heuristics to initialize the weights in the system, and then use pairwise comparisons to evaluate and fine-tune them.

In this paper, we adopt the notion of *entropy* [22] [5] [43] to quantify the quality of query results. Since crowdsourcing tasks are costly in terms of money and time, there is a budget limit for the number of object pairs to be crowdsourced. As such, the objective of this paper is to carefully select object pairs for crowdsourcing in order to maximize the quality improvement of query results under limited budgets. There are two main challenges in this problem. First, the candidate object pairs are quadratic to the number of objects. Therefore, an issue is how to efficiently select the best object pairs from all the candidate pairs. Second, it is very costly to compute the exact quality improvement gained by pairwise crowdsourcing, because the number of possible top-k object sets is huge when k is large. To address these challenges, we devise an index called Probabilistic B-tree (PB-tree for short) to accelerate the sorting of candidate object pairs. Moreover, we propose an efficient algorithm to obtain the lower and upper bound of the quality improvement gained by crowdsourcing and hence significantly accelerate the selection procedure.

Our contributions made in this paper are summarized as follows:

- We present a pairwise crowdsourcing model to improve the quality of probabilistic top-k ranking on uncertain objects. To the best of our knowledge, this is the first study of top-k ranking across possible worlds under a pairwise crowdsourcing model.
- For the single object-pair selection problem, we propose a PB-tree-based solution and an bound-based method to efficiently find the best object pair for crowdsourcing. Moreover, a branch-and-bound optimization is devised to prune the candidate node pairs. This solution works for both order-insensitive and order-sensitive settings.
- To minimize the latency of crowdsourcing, we extend the above solution to select object pairs in batch so that they can be simultaneously posted in the crowdsourcing system. To avoid enumerating all possible combinations of object pairs, we propose two heuristics techniques for efficient selection.
- We conduct extensive experiments to evaluate the performance of the proposed solutions. The results show that our proposed methods outperform ran-

1. In this paper, the term "crowdsourcing" is mainly referred to as "expert sourcing" (i.e., making use of a small crowd of domain experts).

dom selection by 30X in terms of result quality improvement. Meanwhile, in terms of efficiency, our proposed methods can finish the selection of object pairs within one minute, while the brute-force algorithm takes several days.

The rest of this paper is organized as follows. In Section 2, we present the background and related work. In Section 3, we define the problem and give a brief analysis of its complexity. In Section 4, we focus on selecting the best object pair which maximizes the quality improvement and then extend the solution to the multiple object-pair selection problem in Section 5. Experimental results are presented in Section 6. Finally, we conclude this paper in Section 7.

2 Related Work

2.1 Crowdsourcing

Crowdsourcing has been a hot topic in the database community in recent years. Crowd-powered database platforms (e.g., [13] [25]) have been built to combine human intelligence with machine computing for challenging problems. Quality control is a fundamental issue in crowdsourcing systems. [36] and [44] separately addressed the diversity and noise issues of the results from the crowd. [16] and [3] studied how to evaluate the performance of workers in crowdsourcing platforms. A number of studies have focused on resolving specific queries on these platforms. As one of the most important query operators in database systems, crowdsourced join queries have been studied in [9] [37] [25]. Other query operators, including sorting [25], enumeration [31], and max [14] [35], have also been investigated on crowdsourcing platforms.

As subjective tasks are best suited for crowdsourcing, it has been intensively studied how to resolve entity resolutions via crowdsourcing [37] [9] [33] [39] [38] [34]. [23] considers counting operations in crowdsourcing environments, while [26] uses crowdsourcing to facilitate rating and filtering. [20] adopts a crowdsourcing platform to help make traveling plans. [46] aids people to select the best path between a source and a destination. [24] leverages crowdsourcing in active learning. [32] proposes a human-powered top-k query processing solution, in which the system asks the crowd to rank the objects directly and aggregates them to get the final top-k results. We argue that it is more difficult for the crowd to rank a list of objects than to compare a pair of them. Moreover, it cannot handle uncertain top-k queries with prior rating knowledge. In [6], the authors proposed a human-powered uncertain top-k query cleaning solution. However, it used probabilistic distribution function (PDF) as the model of the uncertain data, whereas we adopt a more well-studied possible world model. In [10], an object comparison model has been suggested to select top-k objects. However, it only works for deterministic data and cannot be directly applied to probabilistic data.

While most of the above studies exploit machine-based techniques to reduce the cost of crowdsourcing, [43] uses crowdsourcing to reduce the uncertainty of semantic matching, which is the closest to our work. In particular, it aims to find out the crowdsourced entity pairs with the highest quality improvement, measured by entropy as in our work. [30] also adopts crowdsourcing techniques to clean multi-version data on the Web, while [8] uses crowdsourcing and knowledge base to clean relational data. However, to the best of our knowledge, there is no work leveraging crowdsourcing to reduce the uncertainty of top-k ranking, which is the focus of this paper.

2.2 Probabilistic Top-k Queries and Uncertainty Reduction

Probabilistic top-k queries have been extensively studied under various semantics, including U-Top$k$ [29], u-kRanks [29], PT-k [15], global-top$k$ [42], and expected rank queries [7]. Efficient query processing algorithms have been developed to evaluate the probabilistic top-k queries. However, the existing studies only return the most probable top-k results, although such results may contain high uncertainty.

On the other hand, data cleaning is a well-studied technique that aims to reduce the uncertainty of query results. [18] makes use of users' feedback to promote the quality of query results from diverse databases. [19] analyzes the sensitivity and explanation of query results in probabilistic query results. [5] defines the quality of max and range query results on uncertain data, based on which a mechanism is devised to promote the result quality with a limited cleaning budget. [45] aims at diagnosing faults in a system represented by a Bayesian network. [12] focuses on reducing uncertainty of pipeline. [17] selects the most informative subset of variables to reduce uncertainty in a graphical model. [21] aims to clean the edges in uncertain graphs.

The closest work to our problem is [22], which aims to reduce the uncertainty of probabilistic top-k queries. However, there are two major differences from our work. First, [22] assumes that the exact values of uncertain objects can be determined after the cleaning process, by which the uncertainty of top-k results will be reduced accordingly. Thus, it adopts a singleton model to clean the data. In contrast, we adopt a novel pairwise crowdsourcing model to filter out violated possible worlds for subjective rating tasks. Second, [22] defines top-k results at the instance level, which identifies top-k instances with the highest ratings. However, in many top-k applications, the top-k objects are preferred as query results. In the example of age guessing, the objective is to find the top-k photos, instead of top-k guessed ages. Hence, we focus on top-k queries at the object level. Unfortunately, the problem of processing object-level probabilistic top-k queries is more sophisticated than the instance-level problem. As such, we resort to crowdsourcing to reduce the uncertainty of object-level top-k ranking in this paper.

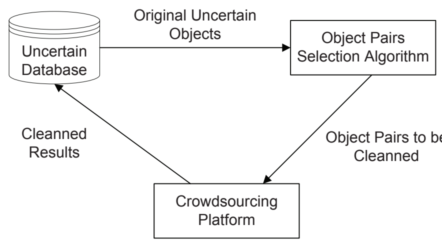
3 Problem Definition

3.1 Probabilistic Database

We adopt the possible world semantics to model uncertain objects [22]. For better readability, we summarize the frequently used notations in Table 3. Let a probabilistic database $\mathcal{D}$ contain m uncertain objects o_1, o_2,

Notation	Meanings
$P(W)$	The probability of the possible world W
$i_m(W)$	The top-m instance of the possible world W
$o_m(W)$	The top-m object of the possible world W
$P(k, S)$	The probability that S is the top-k result
$H(\mathcal{S}_k)$	The quality of the probabilistic top-k ranking
$EH(\mathcal{S}_k \mid (o_x, o_y))$	Excepted quality after crowdsourcing object pair (o_x, o_y)
$EI(\mathcal{S}_k \mid (o_x, o_y))$	Excepted quality improvement after crowdsourcing object pair (o_x, o_y)

TABLE 3
Frequently Used Notations


Uncertain Database
Original Uncertain Objects
Object Pairs Selection Algorithm
Cleanned Results
Object Pairs to be Cleanned
Crowdsourcing Platform
Fig. 2. Framework of Crowdsourcing Cleanning

$\ldots$, o_m. Each uncertain object is composed of several uncertain instances.[2] An uncertain instance i is in the form of $<oid, iid, v, p>$, where $i.oid$ is the identifier of the uncertain object containing i, $i.iid$ is the instance id of i, $i.v$ is the attribute value of i, and $i.p$ is the existential probability of i. In this paper, we use $i.v$ and i interchangeably for value comparison. For simplicity, we assume no pair of instances shares the same attribute value. The instances of the same uncertain object are mutually exclusive and the sum of their probabilities is 1. In addition, we assume the instances of an object are independent of other objects. Under the possible world semantics, the uncertain relations of objects are regarded as a set of possible worlds, each of which contains exactly one instance from each uncertain object. The probability of a possible world W is the product of those of the instances in it, *i.e.*,

$$P(W) = \prod_{i \in W} i.p.$$

Given two uncertain objects o_x and o_y, "$o_x > o_y$ ($o_x < o_y$)" means that the instance of o_x is larger (smaller) than the instance of o_y. The probability of $o_x > o_y$ can be computed by summing up the probabilities of the possible worlds where "$o_x > o_y$" holds. Formally, it can be computed by the following equation:

$$\begin{aligned} P(o_x > o_y) &= \textstyle\sum_{i_x \in o_x} i_x.p \cdot (\sum_{i_y \in o_y, i_x > i_y} i_y.p) \\ &= \textstyle\sum_{i_y \in o_y} i_y.p \cdot (\sum_{i_x \in o_x, i_x > i_y} i_x.p). \end{aligned} \tag{1}$$

According to Eq. (1), we have

$$P(o_x < o_y) = P(o_y > o_x) = 1 - P(o_x > o_y).$$

Take Fig. 1 and Table 1 for example, the probability of "$o_2 > o_1$" can be computed by summing up the probabilities of $W_1 - W_4$, W_7, and W_8, *i.e.*, $P(o_2 > o_1) = 0.84$, and $P(o_1 > o_2) = 1 - P(o_2 > o_1) = 0.16$. In the sequel, we use object and uncertain object interchangeably.

3.2 Quality of Probabilistic Top-k Ranking

Without loss of generality, for a top-k query, we say that the instance with a smaller value ranks higher than the one with a larger value. We then define the top-k result below, based on which we further define the quality of top-k query results.

Definition 1: **Object-Level Top-k Result of Possible Worlds**. Given a possible world W, let $i_m(W)$ denote the top-mth instance of W and $o_m(W)$ be the uncertain object in $i_m(W)$. The top-k result of W returns a set of uncertain objects $R(k, W)$ such that

$$R(k, W) = \{o_m(W) \mid m \leq k\}. \tag{2}$$

Let us revisit the example in Fig. 1 and Table 1, where we want to search the top-k smallest objects. In the possible world W_1, $R(2, W_1) = \{o_1, o_2\}$. The rightmost column of Table 1 lists the top-2 objects of each possible world, *i.e.*, $R(2, W_i)$ for each W_i.

Definition 2: **Probability of Object-Level Top-k Result**. Given a set of k objects, S, the probability that S is the top-k result is defined as follows:

$$P(k, S) = \sum_{R(k,W)=S} P(W). \tag{3}$$

Here the set equation in "$R(k, W) = s$" can be either order-insensitive or order-sensitive. In Table 1, if $S = \{o_1, o_3\}$ and it is order-sensitive, only $R(2, W_3) = S$; if S is order-insensitive, $R(2, W_3) = R(2, W_7) = S$. In this paper, we firstly focus on the order-insensitive problem, followed by the extension to the order-sensitive problem.

Due to the diversity of possible worlds, there can be many top-k results. We use notation $\mathcal{S}_k$ to denote the sets covering all possible top-k results. To quantify the quality of all results, we use entropy to measure the uncertainty of the ranking, based on the probability distribution of all results. Higher entropy means the returned top-k result is more uncertain and thus of lower quality. As in [5] [22] [28], we formally define the *quality of probabilistic top-k ranking* as follows:

$$H(\mathcal{S}_k) = \sum_{S \in \mathcal{S}_k} h(P(k, S)), \tag{4}$$

where the function $h(x) = -x \log x$ and $\log$ is the natural logarithm. As $\lim_{x \to 0} x \log x = 0$, we define $h(P(k, S))$ $= 0$ if $P(k, S) = 0$.

Consider our running example. If the top-k results are order-insensitive, $\mathcal{S}_k = \{\{o_1, o_2\}, \{o_1, o_3\}, \{o_2, o_3\}\}$. As such, $P(2, \{o_1, o_2\}) = P(W_1) + P(W_2) + P(W_4) + P(W_6) + P(W_8) = 0.424$, $P(2, \{o_1, o_3\}) = P(W_3) + P(W_7) = 0.48$, and $P(2,\ \{o_2, o_3\}) = P(W_5) = 0.096$. Hence, $H(\mathcal{S}_k) = -0.424 \log 0.424\ -0.48 \log 0.48$ - $0.096 \log 0.096 = 0.941$.

2. The concepts "object" and "instance" are equivalent to "x-tuple" and "tuple" in the x-tuple model [22].

3.3 Pairwise Comparison via Crowdsourcing

To improve the quality of probabilistic top-k ranking, we leverage crowdsourcing to reduce the uncertainty in possible worlds. As shown in Fig. 2, the server selects a limited number of object pairs by our proposed algorithm and crowdsources them to the domain experts for comparison. We assume that some conflict resolution mechanism (e.g., majority voting rule) is in place and the comparison result returned from the crowd is deterministic. Such results help the server to set the probabilities of those possible worlds that contradict with them to zero, and the probabilities of the remaining possible worlds to be proportionally enlarged so that they still sum up to 1. Assuming the server crowdsources a pair of objects o_x, o_y and the crowd returns that $o_x > o_y$, the new probability of a remaining possible world W is updated as follows:

$$P(W|o_x > o_y) = \begin{cases} \frac{P(W)}{P(o_x > o_y)}, & o_x > o_y \text{ in } W; \\ 0, & o_x < o_y \text{ in } W. \end{cases} \tag{5}$$

Since $P(W)$ is updated to $P(W|o_x > o_y)$, $P(k, S)$ for each possible top-k result set S is changed accordingly:

$$P(k, S|o_x > o_y) = \sum_{R(k,W)=S} P(W|o_x > o_y).$$

As such, the quality of the top-k ranking is also updated as follows:

$$H(\mathcal{S}_k|o_x > o_y) = \sum_{S \in \mathcal{S}_k} h(P(k, S|o_x > o_y)).$$

Thus, the posterior quality improvement of $o_x > o_y$ is $H(\mathcal{S}_k) - H(\mathcal{S}_k|o_x > o_y)$. Given a limited budget of crowdsourcing, our objective is to find the best pair(s) of objects which would bring in the largest quality improvement. However, the challenge lies in that we cannot tell apriori the comparison result before crowdsourcing an object pair. In the above running example, either $o_x > o_y$ or $o_y > o_x$ could be returned. Therefore, we need to consider both cases and use the notion *expected quality* (denoted by $EH(\mathcal{S}_k|(o_x, o_y))$) and *expected quality improvement* (denoted by $EI(\mathcal{S}_k|(o_x, o_y))$) for pair selection. They are formally defined in the following equations:

$$\begin{aligned} EH(\mathcal{S}_k|(o_x, o_y)) &= H(\mathcal{S}_k|o_x > o_y)P(o_x > o_y) \\ &+ H(\mathcal{S}_k|o_x < o_y)P(o_x < o_y), \end{aligned} \tag{6}$$

$$EI(\mathcal{S}_k|(o_x, o_y)) = H(\mathcal{S}_k) - EH(\mathcal{S}_k|(o_x, o_y)). \tag{7}$$

For the example in Fig. 1 and Table 1, if we crowdsource object pair (o_1, o_2) and the crowd returns $o_2 < o_1$, then the possible worlds W_1-W_4, W_7, and W_8 will be removed from the possible world space. According to Eq. (5), the probabilities of the remaining possible worlds W_5 and W_6 are enlarged to 0.6 and 0.4, respectively. Hence, the quality is updated to $H(\mathcal{S}_k|o_2 < o_1) = -0.6 \log 0.6 - 0.4 \log 0.4 = 0.67$. Otherwise, the crowd returns $o_1 < o_2$ and the quality is updated to 0.683. According to Eq. (7), the expected quality improvement is $EI(\mathcal{S}_k|(o_1, o_2)) = 0.941 - (0.683 \times 0.84 + 0.67 \times 0.16) = 0.26$.

The above definitions can be directly generalized to multiple pairs of crowdsourced objects. Assuming the quota of crowdsourced object pairs is n, we can select n object pairs to improve the quality of top-k ranking. These pairs are denoted by $\mathcal{P}_n = ((o_x^1, o_y^1), (o_x^2, o_y^2), \ldots, (o_x^n, o_y^n))$. There are 2^n possible results of the comparison between o_x^i and o_y^i ($1 \le i \le n$). Let $A(\mathcal{P}_n)$ denote the global set of such combinations. The *expected quality* (denoted by $EH(\mathcal{S}_k|\mathcal{P}_n)$) and *expected quality improvement* (denoted by $EI(\mathcal{S}_k|\mathcal{P}_n)$) are defined as follows:

$$EH(\mathcal{S}_k|\mathcal{P}_n) = \sum_{e \in A(\mathcal{P}_n)} H(\mathcal{S}_k|e)P(e), \tag{8}$$

$$EI(\mathcal{S}_k|\mathcal{P}_n) = H(\mathcal{S}_k) - EH(\mathcal{S}_k|\mathcal{P}_n), \tag{9}$$

where e is a combination of the comparison relationships between o_x^i and o_y^i. For example, $n = 2$, $\mathcal{P}_2 = ((o_1, o_2), (o_1, o_3))$, $A(\mathcal{P}_n)$ is a set with four possible cases, *i.e.*, {"$o_1 > o_2$ and $o_1 > o_3$" , "$o_2 > o_1$ and $o_1 > o_3$", "$o_1 > o_2$ and $o_3 > o_1$", "$o_2 > o_1$ and $o_3 > o_1$"}.

Definition 3: **Pairwise Crowdsourcing Problem**. Given an uncertain object set O and the quota of cleaned object pairs n, the pairwise crowdsourcing problem is to find n object pair combinations $\mathcal{P}_n$ which maximize $EI(\mathcal{S}_k|\mathcal{P}_n)$.

3.4 Problem Analysis

In fact, if we consider $\mathcal{S}_k$ and $A(\mathcal{P}_n)$ as two different random variables, $EH(\mathcal{S}_k|\mathcal{P}_n)$ can be regarded as the conditional entropy of $\mathcal{S}_k$ on $A(\mathcal{P}_n)$. Let $H(A(\mathcal{P}_n))$ denote the entropy on random variable $A(\mathcal{P}_n)$, *i.e.*, $H(A(\mathcal{P}_n)) = \sum_{e \in A(\mathcal{P}_n)} h(P(e))$. According to the property of conditional entropy, we obtain:

$$EH(\mathcal{S}_k|\mathcal{P}_n) = H(\mathcal{S}_k, A(\mathcal{P}_n)) - H(A(\mathcal{P}_n)), \tag{10}$$

$$\begin{aligned} EI(\mathcal{S}_k|\mathcal{P}_n) &= H(A(\mathcal{P}_n)) - (H(\mathcal{S}_k, A(\mathcal{P}_n)) - H(\mathcal{S}_k)) \\ &= H(A(\mathcal{P}_n)) - \Delta(A(\mathcal{P}_n)), \end{aligned} \tag{11}$$

where $H(\mathcal{S}_k, A(\mathcal{P}_n))$ is the joint entropy of $\mathcal{S}_k$ and $A(\mathcal{P}_n)$, and $\Delta(A(\mathcal{P}_n))$ denotes $H(\mathcal{S}_k, A(\mathcal{P}_n)) - H(\mathcal{S}_k)$. It is easy to prove that both $\Delta(A(\mathcal{P}_n))$ and $EI(\mathcal{S}_k|\mathcal{P}_n)$ are no less than zero.

Thus, we derive $EI(\mathcal{S}_k|\mathcal{P}_n)$ by treating the two factors $H(A(\mathcal{P}_n))$ and $\Delta(A(\mathcal{P}_n))$ separately at first and then combining them. The key problems are to efficiently derive $H(A(\mathcal{P}_n))$ and $\Delta(A(\mathcal{P}_n))$, and to rank the differences of them. As such, the following two sections will discuss how to achieve such computation and ranking for single-quota and multi-quota settings, respectively.

4 Single-Quota Selection Algorithm

We first focus on the object selection problem with a single quota, *i.e*, only one pair of objects is crowdsourced to improve the quality of top-k ranking. For simplicity, we use $\mathcal{P}_1$ and $\mathcal{P}_n$ interchangeably in this section. The basic algorithm to solve this problem is to examine all object pairs and compare their expected quality improvement. However, the cost might be prohibitively high given a large number of objects. In this section, we propose an index-based technique to prune the search space as well

Algorithm 1 Basic Solution to Find Object Pairs with Maximum Quality Improvement

```
INPUT: $\mathcal{L}$: Uncertain Object List
OUTPUT: $\mathcal{P}_1^h$ : Object Pair with Maximum Quality Improvement
1: initialization($\mathcal{L}$)
2: $EI_{max} \leftarrow 0$
3: while (($o_1^h, o_2^h$) = getNextPair()) $\neq$ null do
4:     $\mathcal{P}_1 = (< o_1^h, o_2^h >)$
5:     if $H(A(\mathcal{P}_1))$ - $\Delta(A(\mathcal{P}_1)) > EI_{max}$ then
6:         $EI_{max} \leftarrow H(A(\mathcal{P}_1))$ -$\Delta(A(\mathcal{P}_1))$
7:         $\mathcal{P}_1^r \leftarrow \mathcal{P}_1$
8:     if $H(A(\mathcal{P}_1)) < EI_{max}$ then
9:         return $\mathcal{P}_1^r$
```

as an approximate algorithm to efficiently compute the expected quality improvement.

Since $H(\mathcal{S}_k)$ is a constant value, as indicated by Eq. (10), lower $H(\mathcal{S}_k, A(\mathcal{P}_1))$ and larger $H(A(\mathcal{P}_1))$ are preferred in the object pair selection. The derivation of $H(\mathcal{S}_k, A(\mathcal{P}_1))$ is more complex than that of $H(A(\mathcal{P}_1))$ since the former needs to enumerate all possible results in $\mathcal{S}_k$, which is prohibitively inefficient. On the other hand, since $A(\mathcal{P}_1)$ only contains two possible cases "$o_x > o_y$" and "$o_x < o_y$", $H(A(\mathcal{P}_1))$ can be derived by the following equation:

$$\begin{aligned} H(A(\mathcal{P}_1)) &= h(P(o_x > o_y)) + h(P(o_x < o_y)) \\ &= h(P(o_x > o_y)) + h(1 - P(o_x > o_y)). \end{aligned} \tag{12}$$

Algorithm 1 shows the basic algorithm to select the object pair with the maximum expected quality improvement. It accesses object pairs in descending order of $H(A(\mathcal{P}_1))$. For each accessed object pair, it computes $\Delta(A(\mathcal{P}_1))$ for this pair and get the expected quality improvement by Eq. (11). The algorithm stops when an object pair has an $H(A(\mathcal{P}_1))$ smaller than EI_{max}, the maximum expected quality improvement of the accessed pairs so far. In this situation, the remaining object pairs cannot achieve a higher quality improvement since it is bounded by $H(A(\mathcal{P}_1))$. *Initialization* and *getNextPair* are two functions to initialize data structures and to get the next object pair in descending order of $H(A(\mathcal{P}_1))$, which will be elaborated in Algorithms 2 and 3.

The above algorithm leaves us two key problems to solve: 1) how to access object pairs in descending order of $H(A(\mathcal{P}_1))$; and 2) how to efficiently compute $\Delta(A(\mathcal{P}_1))$ for each object pair. We will present the solutions in the following two sections.

4.1 Accessing Object Pairs in Descending Order of $H(A(\mathcal{P}_1))$

A brute-force solution is to compute the $H(A(\mathcal{P}_1))$ values of all pairs and then sort them. However, the computational complexity is $O(n^2)$, where n is the number of the objects. A key observation is that the $H(A(\mathcal{P}_1))$ values of many object pairs are too small and should be pruned at early time. Thus, we propose an index, named Probabilistic B-tree (or PB-tree in short), for this pruning. In this section, we first present how to access object pairs in sequence by a PB-tree, followed by the construction of PB-tree.

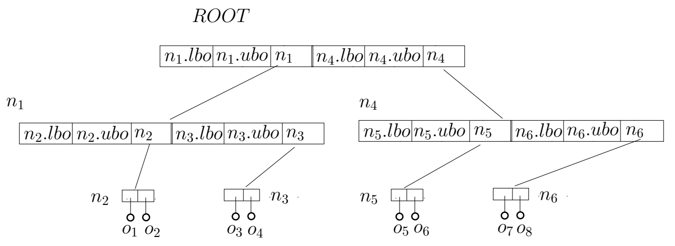

ROOT
$n_1.lbo$ $n_1.ubo$ n_1 $n_4.lbo$ $n_4.ubo$ n_4
n_1
$n_2.lbo$ $n_2.ubo$ n_2 $n_3.lbo$ $n_3.ubo$ n_3
n_4
$n_5.lbo$ $n_5.ubo$ n_5 $n_6.lbo$ $n_6.ubo$ n_6
n_2
n_3
n_5
n_6
o_1 o_2
o_3 o_4
o_5 o_6
o_7 o_8

Fig. 3. PB-tree

Fig. 3 shows the basic architecture of PB-tree. It is a variant of B-tree, which summarizes both attributes and probabilities of the objects. A node n is in the form of $(ptrs, lbo, ubo)$, where $ptrs$ is a list of pointers pointing to the children nodes, lbo and ubo are two pseudo-objects that satisfy $lbo \trianglelefteq o \trianglelefteq ubo$ for each object o under this node. We call these pseudo-objects as *bound objects* of the node. Here the notation $\trianglelefteq$ means a "dominance" relationship, which is defined as follows:

Definition 4: **Dominance**. Given two objects o_1 and o_2, we say o_1 dominates o_2 (denoted by $o_1 \trianglelefteq o_2$) iff for any arbitrary positive value d, it holds that

$$\sum_{i_1 \in o_1, i_1.v<d} i_1.p \geq \sum_{i_2 \in o_2, i_2.v<d} i_2.p,$$

and

$$\sum_{i_2 \in o_2, i_2.v>d} i_2.p \geq \sum_{i_1 \in o_1, i_1.v>d} i_1.p.$$

The basic idea of the concept is to ensure that $P(o_1 < o_2) = 1$ holds if $o_1 \trianglelefteq o_2$. For example, there are two objects o_1 and o_2, where o_1 is composed of two instances i_{11} <10, 0.6> and i_{12} <30, 0.4>, and o_2 is composed of two instances i_{21} <20, 0.5> and i_2 <40, 0.5>. We have $o_1 \trianglelefteq o_2$ since for any value d, it holds that the cumulative probability of the instances with a value less than d in o_1 is larger than that in o_2.

According to the definition, dominance is a transitive relationship. We regard $n.lbo$ and $n.ubo$ as two bounds of the objects under n. The following theorem shows an important property of these two bounds.

Theorem 1: Given two PB-tree nodes n_1 and n_2, assuming o_1 and o_2 are two arbitrary objects under n_1 and n_2, respectively, we have:

$$P(n_1.ubo > n_2.lbo) \geq P(o_1 > o_2) \geq P(n_1.lbo > n_2.ubo).$$

Proof. Omitted due to space limitations. See Appendix A for detailed proof.

Since all objects of a child node n_c must also be under the parent node of n_c, the relationship between the bound objects of the parent node and those of a child node is shown in the following lemma:

Lemma 1: If node n_c is a child node of node n_p, we have $n_c.ubo \trianglelefteq n_p.ubo$ and $n_p.lbo \trianglelefteq n_c.lbo$.

By Theorem 1, we can derive the bounds of $P(o_1 > o_2)$ by only accessing the nodes containing them. Let $\check{P}(o_1 > o_2)$ and $\hat{P}(o_1 > o_2)$ denote the lower bound and upper bound of $P(o_1 > o_2)$, respectively, that is,

$$\check{P}(o_1 > o_2) = P(n_1.lbo > n_2.ubo), \tag{13}$$

Algorithm 2 Function: initialization

```
INPUT: $\mathcal{L}$: Uncertain Object List
1: $\mathcal{T} \leftarrow$ the PB-tree of $\mathcal{L}$
2: $\mathcal{NP} \leftarrow \emptyset$ // max heap storing PB-tree node pairs $(n_1, n_2)$'s in the descending order of $\hat{H}(n_1, n_2)$
3: $\mathcal{OP} \leftarrow \emptyset$ // max heap storing object pairs in the descending order of $H(A)$
4: insert ($\mathcal{T}.root$,$\mathcal{T}.root$) into $\mathcal{NP}$
```

Algorithm 3 Function: getNextPair

```
OUTPUT: ($\mathcal{P}_1$): the Next Object Pair in the Descending Ordered of $H(A)$
1: **if** $\mathcal{NP}$ is not empty **then**
2:   $(n_1, n_2) \leftarrow$ **Dequeue**($\mathcal{NP}$)
3:   **if** $\mathcal{OP}$ is not empty **then**
4:     $\mathcal{P}_1^h \leftarrow$ **Dequeue**($\mathcal{OP}$)
5:     **if** $H(A(\mathcal{P}_1^h)) > \hat{H}(n_1, n_2)$ **then**
6:       **return** $\mathcal{P}_1^h$
7:     **else**
8:       **Enqueue**($\mathcal{P}_1^h$, $\mathcal{OP}$)
9:   **if** $n_1$ and $n_2$ are leaf nodes **then**
10:    **for** each child object $o'_1$ of $n_1$ **do**
11:      **for** each child object $o'_2$ of $n_2$ **do**
12:        insert $(o'_1, o'_2)$ into $\mathcal{OP}$
13:  **else** // $n_1$ and $n_2$ are index nodes
14:    **for** each child node $c_1$ of $n_1$ **do**
15:      **for** each child node $c_2$ of $n_2$ **do**
16:        insert $(c_1, c_2)$ into $\mathcal{NP}$
17: **else** // $\mathcal{NP}$ is empty
18:   **if** $\mathcal{OP}$ is empty **then**
19:     **return** *null*
20:   **else**
21:     **return** **Dequeue**($\mathcal{OP}$)
```

$$\hat{P}(o_1 > o_2) = P(n_1.ubo > n_2.lbo). \tag{14}$$

Let us revisit the function $H(x) = h(x) + h(1-x)$, which is the basic form of Eq. (12). It has three properties: 1) it is monotonously increasing if $x \in [0, 0.5]$; 2) it is monotonously decreasing if $x \in [0.5, 1]$; and 3) it satisfies that $H(x) = H(1-x)$. Thus, given an object pair $\mathcal{P}_1 = (o_1, o_2)$, where o_1 and o_2 are respectively under PB-tree nodes n_1 and n_2, the lower bound and upper bound of $H(A(\mathcal{P}_1))$, denoted by $\check{H}(n_1, n_2)$ and $\hat{H}(n_1, n_2)$, can be derived as follows:

$$\begin{aligned}\check{H}(n_1, n_2) = \quad & H(0.5 - max(abs(\check{P}(o_1 > o_2) - 0.5), \\ & abs(\hat{P}(o_1 > o_2) - 0.5))),\end{aligned} \tag{15}$$

$$\begin{aligned}\hat{H}(n_1, n_2) = \quad & H(0.5 - min(abs(\check{P}(o_1 > o_2) - 0.5), \\ & abs(\hat{P}(o_1 > o_2) - 0.5))),\end{aligned} \tag{16}$$

where *abs*() is the absolute-value function.

Based on the above discussions, we obtain an efficient algorithm to access the object pairs in descending order of $H(A(\mathcal{P}_1))$. The psuedo-code is shown in Algorithms 2 and 3. In the *initialization* function (Algorithm 2), the PB-tree is first constructed. We also maintain two max heaps $\mathcal{NP}$ and $\mathcal{OP}$ storing the node pairs and object pairs, where $\mathcal{NP}$ is in desending order of $\hat{H}(n_1, n_2)$ of

Algorithm 4 Derivation of the Lower Bound

```
INPUT: $O$: A set covering all objects under a node $n$
OUTPUT: $lbo$: The lower bound of $n$
1: $\mathcal{I} \leftarrow$ min heap storing all instances in $O$ in the ascending order
2: $lbo \leftarrow$ an object with no instance
3: $tp \leftarrow 0$
4: **for** each object $o_x$ in $O$ **do**
5:   $o_x.rp \leftarrow 0$
6: **while** $\mathcal{I}$ is not empty **do**
7:   $i \leftarrow$ **Dequeue**($\mathcal{I}$)
8:   $o \leftarrow$ the object containing $i$
9:   **if** $o.rp \geq i.p$ **then**
10:    $o.rp \leftarrow o.rp - i.p$
11:    **continue**
12:  **else**
13:    $p_m \leftarrow i.p - o.rp$
14:    $i_l \leftarrow$ new instance with $i_l.v = i.v$ and $i_l.p = p_m$
15:    insert $i_l$ to the instance list of $lbo$
16:    $tp \leftarrow tp + p_m$
17:    $o.rp \leftarrow 0$
18:    **if** $tp = 1$ **then**
19:      **return** $lbo$
20:    **for** each object $o_x$ in $O$ which $o_x \neq o$ **do**
21:      $o_x.rp \leftarrow o_x.rp + p_m$
```

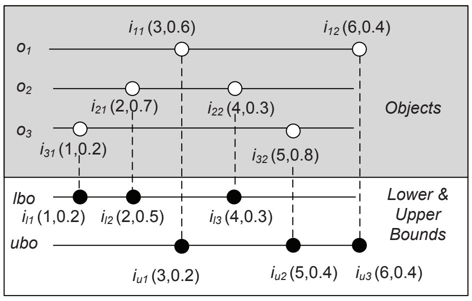


Fig. 4. An Example of Bound Objects Construction

node pairs and $\mathcal{OP}$ is in descending order of $H(o_1, o_2)$ of object pairs. As the final initialization step, a node pair ($\mathcal{T}.root$,$\mathcal{T}.root$) is inserted into $\mathcal{NP}$.

In the *getNextPair* function (Algorithm 3), we first check whether the two heaps, $\mathcal{NP}$ and $\mathcal{OP}$, are empty. If both heaps are empty, *null* is returned (Line 19). If only $\mathcal{NP}$ is empty, we return $\mathcal{P}_1^h$, the first element of $\mathcal{OP}$. If only $\mathcal{OP}$ is empty, we find the children of n_1 and n_2, the top pair in $\mathcal{NP}$, and insert them pairwise into $\mathcal{OP}$ or $\mathcal{NP}$, depending on whether they are index or leaf nodes (Lines 9-16). If neither heap is empty, we compare $H(A(\mathcal{P}_1^h))$ with $\hat{H}(n_1, n_2)$. If the former is larger, it means $\mathcal{P}_1^h$ must be the object pair with the maximum $H(A(\mathcal{P}_1))$ and will be returned. Otherwise, we insert the children pairs of n_1 and n_2 into $\mathcal{OP}$ or $\mathcal{NP}$.

Next, we discuss the construction of PB-tree. It is similar to B-tree, except for the metric of object clustering and bound object generation. Starting with an empty tree, the objects are sequentially inserted into this tree according to a clustering metric. We define a new metric $D(lbo, ubo)$ to measure the distance between bound objects lbo and ubo:

$$D(lbo, ubo) = \sum_{i_u \in ubo} i_u.v \cdot i_u.p - \sum_{i_l \in lbo} i_l.v \cdot i_l.p. \quad (17)$$

For object insertion and node splitting during PB-tree construction, smaller $D(lbo, ubo)$ is preferred to get tighter bounds of $\check{P}(o_1 > o_2)$ and $\hat{P}(o_1 > o_2)$ in Eq. (13) and (14).

For bound object generation, Algorithm 4 shows how to compute the lower bound for a set of objects, and a similar algorithm can compute the upper bound. Fig. 4 shows an example, in which we want to construct the lbo and ubo of o_1, o_2, and o_3. We first sort the instances of these objects by their values and get an instance list $\mathcal{I}$: $\{i_{31}, i_{21}, i_{11}, i_{22}, i_{32}, i_{12}\}$. To compute lbo, we access the list in the ascending order. During the construction process, we maintain a global variable tp to record the sum of probabilities of the constructed instances in lbo, as well as a variable rp for each object o_x to record the difference between tp and the sum of probabilities of the accessed instances of o_x. All $o_x.rp$ and tp are initialized as 0 (Lines 3-5). The first accessed instance is i_{31}. Since $o_3.rp = 0 < i_{31}.p$ (Lines 9 and 12), the first instance of lbo, *i.e.*, i_{l1}, is constructed, where $i_{l1}.v = i_{31}.v = 1$ and $i_{l1}.p = i_{31}.p = 0.2$. Meanwhile, tp, $o_1.rp$, and $o_2.rp$ are all updated to 0.2 (Lines 16, 20, 21). The second accessed instance is i_{21}. Since $o_2.rp = 0.2 < i_{21}.p$, we construct the second instance i_{l2} for lbo, where $i_{l2}.v = i_{21}.v = 2$ and $i_{l2}.p = i_{21}.p - o_2.rp = 0.5$. tp and $o_1.rp$, $o_3.rp$ are then increased by 0.5 and become 0.7, 0.7, and 0.5, while $o_2.rp$ is set to 0. The third accessed instance is i_{11}. Since $o_1.rp = 0.7 > i_{11}.p$, nothing happens except $o_1.rp$ is decreased by $i_{11}.p$ and updated to 0.1. The next accessed instance is i_{22}. Here $o_2.rp = 0 < i_{22}.p = 0.3$ and we construct the third instance i_{l3} for lbo, where $i_{l3}.v = i_{22}.v = 4$ and $i_{l3}.p = 0.3 - 0 = 0.3$. tp is increased by 0.3 and becomes 1 now. After that, the construction process is terminated and lbo is returned (Line 19). The construction of ubo is similar except that we access the instances in descending order of their values.

In fact, the bounds constructed by Algorithm 4 are the tightest bounds of a node, where "tightest" is defined as follows:

Definition 5: **The Tightest Bounds.** Given two psedo-objects lbo and ubo, and a PB-tree node n, lbo and ubo are the *tightest lower and upper bounds* of n iff: 1) lbo and ubo are bound objects of n; 2) for any bound objects lbo', ubo' of n, it holds that $lbo' \trianglelefteq lbo$, $ubo \trianglelefteq ubo'$.

The following theorem proves that our algorithm achieves the tightest bounds.

Theorem 2: Alogrithm 4 returns the tightest lower bounds of a node.

Proof. Omitted due to space limitations. See Appendix B for detailed proof.

As a side-effect, the tightness of our bound also ensures the tightness of the bound of $\check{P}(o_1 > o_2)$ and $\hat{P}(o_1 > o_2)$. This guarantees the efficiency and correctness of pruning effect of PB-tree.

4.2 Derivation of $\Delta(A(\mathcal{P}_1))$

A brute-force solution to deriving $\Delta(A(\mathcal{P}_1))$ is to compute all possible top-k results in S_k as well as their probabilities. However, the cardinality of such results is $C_{|I|}^k$, where $|I|$ is the total number of instances. As $|I|$ and k grow up, the computation cost would become prohibitively high. In this section, we propose a bound-based solution to estimate the value of $\Delta(A(\mathcal{P}_1))$. The basic idea is to get a tight lower bound and a tight upper bound for each $\Delta(A(\mathcal{P}_1))$, and to use an arbitrary value within the bounds to approximate the exact value of $\Delta(A(\mathcal{P}_1))$.

Let us assume $\mathcal{P}_1$=(o_1, o_2) is the crowdsourced object pair. According to the definition of $\Delta(A(\mathcal{P}_1))$, $\Delta(A(\mathcal{P}_1)) = H(S_k, (o_1, o_2)) - H(S_k) = \sum_{s \in S_k}(h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s))$, where s is an arbitrary combination of k objects in S_k, $P(s)$ is the probability of s being a top-k result, and $P(s, o_1 > o_2)$ is the joint probability of both s being a top-k result and $o_1 > o_2$. Obviously, for any s, $P(s) = P(s, o_1 > o_2) + P(s, o_1 < o_2)$. s can be divided into four subsets by o_1 and o_2: 1) $S_1 = \{s | o_1 \in s, o_2 \notin s\}$; 2) $S_2 = \{s | o_1 \notin s, o_2 \in s\}$; 3) $S_{1,2} = \{s | o_1 \in s, o_2 \in s\}$; and 4) $S_\phi = \{s | o_1 \notin s, o_2 \notin s\}$. For all $s \in S_1$, $o_1 > o_2$ always holds. Hence, $P(s) = P(s, o_1 > o_2)$ and $P(s, o_1 < o_2) = 0$, which means $h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s)) = 0$. Similarly, for all $s \in S_2$, we also have $h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s)) = 0$. Therefore, we only need to consider $s \in S_{1,2}$ or S_ϕ, *i.e.*, $\Delta(A(\mathcal{P}_1)) = \sum_{s \in S_{1,2} \bigcup S_\emptyset}(h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s))$.

Let $IP(o_1, o_2)$ denote the Cartesian product of instances of o_1 and o_2, *i.e.*, $IP(o_1, o_2) = \{(i_1, i_2) | i_1 \in o_1, i_2 \in o_2\}$. According to the comparison relationship of $i_1.v$ and $i_2.v$, $IP(o_1, o_2)$ can be divided into two subsets, *i.e.*, $IP_1(o_1, o_2) = \{(i_1, i_2) | i_1 \in o_1, i_2 \in o_2, i_1 < i_2\}$ and $IP_2(o_1, o_2) = \{(i_1, i_2) | i_1 \in o_1, i_2 \in o_2, i_1 > i_2\}$. Let $PT_k(i_1, i_2, O_t)$ denote the total probability that both i_1 and i_2 are contained by the top-k results of object set O_t, and $NPT_k(i_1, i_2, O_t)$ denote the total probability that neither i_1 nor i_2 is contained by the top-k results of O_t. For brevity, if O_t is identical to O, we omit it in the notation. To derive the bounds, we sort the instance pairs (i_1, i_2) in $IP(o_1, o_2)$ to get two sorted lists $IP_{pt}(o_1, o_2)$ and $IP_{npt}(o_1, o_2)$, where the former is in the descending order of $max\{i_1.v, i_2.v\}$, and the latter is in the ascending order of $min\{i_1.v, i_2.v\}$. Let $\Delta_{1,2}(o_1, o_2) = \sum_{s \in S_{1,2}}(h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s)))$, $\Delta_\phi(o_1, o_2) = \sum_{s \in S_\phi}(h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s)))$. Let $P(i_1, i_2)$ denote the probability that a possible world contains both i_1 and i_2, *i.e.*, $P(i_1, i_2) = i_1.p \times i_2.p$. Algorithm 5 shows the derivation of lower bound and upper bound of $\Delta_{1,2}(o_1, o_2)$ (denoted by "$\check{\Delta}_{1,2}(o_1, o_2)$" and "$\hat{\Delta}_{1,2}(o_1, o_2)$"). The derivation of lower bound and upper bound of $\Delta_\phi(o_1, o_2)$ (denoted by "$\check{\Delta}_\phi(o_1, o_2)$" and "$\hat{\Delta}_\phi(o_1, o_2)$") is similar, except that we use list $IP_{npt}(o_1, o_2)$ instead of $IP_{pt}(o_1, o_2)$, and use function NPT_k instead of PT_k.

For all possible instance pairs (i_1, i_2) belonging to (o_1, o_2), $PT_k(i_1, i_2)$ and $NPT_k(i_1, i_2)$ can be derived by slightly modifying the algorithm in [4], which can compute the probability of a given instance i being a top-k result in O (denoted by $PT_k(i, O)$). Without loss of

Algorithm 5 Derivation of the Bounds of $\Delta_{1,2}(o_1, o_2)$

```
INPUT: $IP_1(o_1, o_2)$, $IP_2(o_1, o_2)$, $IP_{pt}(o_1, o_2)$
OUTPUT: $\check{\Delta}_{1,2}(o_1, o_2)$, $\hat{\Delta}_{1,2}(o_1, o_2)$
1: $pt_1 \leftarrow \sum_{(i_1,i_2)\in IP_1(o_1,o_2)} PT_k(i_1, i_2)$
2: $pt_2 \leftarrow \sum_{(i_1,i_2)\in IP_2(o_1,o_2)} PT_k(i_1, i_2)$
3: $\hat{\Delta}_{1,2}(o_1, o_2) \leftarrow h(pt_1)$ + $h(pt_2)$ - $h(pt_1 + pt_2)$
4: $\check{\Delta}_{1,2}(o_1, o_2) \leftarrow 0$
5: while $IP_{pt}(o_1, o_2)$ is not empty do
6:   $(i_1^x, i_2^x) \leftarrow$ Dequeue($IP_{pt}(o_1, o_2)$)
7:   $p_1 \leftarrow 0$
8:   $p_2 \leftarrow 0$
9:   if $i_1^x.v < i_2^x$ then
10:    $p_1 \leftarrow PT_k(i_1^x, i_2^x)$
11:  else
12:    $p_2 \leftarrow PT_k(i_1^x, i_2^x)$
13:  for each instance pair $(i_1^y, i_2^y)$ after $(i_1^x, i_2^x)$ in $IP_{pt}(o_1, o_2)$ do
14:    if $i_1^y.v < i_2^y$ then
15:      $p_1 \leftarrow p_1$ +$PT_k(i_1^x, i_2^x) \times \frac{P(i_1^y, i_2^y)}{P(i_1^x, i_2^x)}$
16:    else
17:      $p_2 \leftarrow p_2$ + $PT_k(i_1^x, i_2^x) \times \frac{P(i_1^y, i_2^y)}{P(i_1^x, i_2^x)}$
18:    $PT_k(i_1^y, i_2^y) \leftarrow PT_k(i_1^y, i_2^y)$ - $PT_k(i_1^x, i_2^x) \times \frac{P(i_1^y, i_2^y)}{P(i_1^x, i_2^x)}$
19:  $\check{\Delta}_{1,2}(o_1, o_2) \leftarrow \check{\Delta}_{1,2}(o_1, o_2)$ + $(h(p_1) + h(p_2)$-$h(p_1 + p_2))$
```

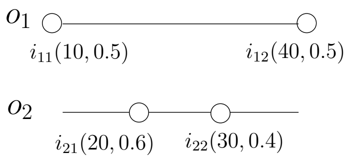


Fig. 5. An Example of Algorithm 5

generality, we assume $i_1.v < i_2.v$. To derive $PT_k(i_1, i_2)$, we first compute $PT_{k-1}(i_2, O-\{o_1\})$, the probability that instance i_2 is a top-$k-1$ result of object set $O - \{o_1\}$. In this case, if i_1 exists, both i_1 and i_2 must be in the top-k of O. Hence, $PT_k(i_1, i_2) = P(i_1) \cdot PT_{k-1}(i_2, O - \{o_1\})$. Similarly, $NPT_k(i_1, i_2) = P(i_2) \cdot (1 - PT_{k-1}(i_1, O - \{o_2\}))$. The computational complexity for deriving $PT_k(i, O)$ is $O(k|I|)$, where $|I|$ is the total number of instances. Let m_i denote the number of instances of object o_i. The complexity of computing the bound of $\Delta_{1,2}(o_1, o_2)$ by Algorithm 5 is $O(km_1m_2)$.

Fig. 5 shows an example of Algorithm 5. Both o_1 and o_2 contain two instances, whose values and probabilities are shown in the parentheses. There are four instance pairs (i_{11}, i_{21}), (i_{11}, i_{22}), (i_{12}, i_{21}), and (i_{12}, i_{22}). According to the comparison relationship of two instances, $IP_1(o_1, o_2) = \{(i_{11}, i_{21}), (i_{11}, i_{22})\}$, $IP_2(o_1, o_2) = \{(i_{12}, i_{21}), (i_{12}, i_{22})\}$. We assume $PT_k(i_{11}, i_{21}) = 0.015$, $PT_k(i_{11}, i_{22}) = 0.01$, $PT_k(i_{12}, i_{21}) = 0.012$, and $PT_k(i_{12}, i_{22}) = 0.008$. Hence, $pt_1 = 0.015 + 0.01 = 0.025$, $pt_2 = 0.012 + 0.008 = 0.02$. We have $\hat{\Delta}_{1,2}(o_1, o_2) = h(0.025) + h(0.02) - h(0.025 + 0.02) = 0.030$. The list $IP_{pt}(o_1, o_2) = \{(i_{12}, i_{22}), (i_{12}, i_{21}), (i_{11}, i_{22}), (i_{11}, i_{21})\}$. To derive $\check{\Delta}_{1,2}(o_1, o_2)$, we access the instance pairs in $IP_{pt}(o_1, o_2)$ sequentially. When accessing the first instance pair (i_{12}, i_{22}), $p_1 = PT_k(i_{12}, i_{22}) \cdot \frac{P(i_{11}, i_{21})}{P(i_{12}, i_{22})} + PT_k (i_{12}, i_{22}) \cdot \frac{P(i_{11}, i_{22})}{P(i_{12}, i_{22})} = 0.02$, $p_2 = PT_k(i_{12}, i_{22}) + PT_k(i_{12}, i_{22}) \cdot \frac{P(i_{12}, i_{21})}{P(i_{12}, i_{22})} = 0.02$, $\check{\Delta}_{1,2}(o_1, o_2)$ is updated to $h(p_1) + h(p_2) - h(p_1 + p_2) = 0.028$. According to Line 17, $PT_k(i_{12}, i_{21})$, $PT_k(i_{11}, i_{22})$, and $PT_k(i_{11}, i_{21})$ are respectively updated to 0, 0.002, and 0.003. As such, we can skip (i_{12}, i_{21}). When accessing (i_{11}, i_{22}) and (i_{11}, i_{21}), p_2 is 0. Thus, $h(p_1)+h(p_2)-h(p_1+p_2)$=0 and $\check{\Delta}_{1,2}(o_1, o_2)$ remains unchanged and finally equals 0.028.

In the following, we will prove that the bounds described above are correct. The tightness of such bounds will be demonstrated by the experiments in Section 6.3.4.

Theorem 3: The bounds computed by Algorithm 5 are correct.

Proof. Omitted due to space limitations. See Appendix C for detailed proof.

4.3 Complexity Analysis

As discussed in Section 4.2, the complexity to approximate $\Delta(A(\mathcal{P}_1))$ is $O(km_1m_2)$, where m_1 and m_2 are the numbers of instances in the object pair. In the worst case, the pruning effect of Algorithm 1 is zero and all object pairs should be explored. As such, Algorithm 1 has the worst time complexity $O(n^2)$, where n is the number of the objects. Thus, the totally complexity of the object pair selection is $O(km_1m_2n^2) = O(kN^2)$, where N is the total number of the instances. Fortunately, in practice, most of the object pairs can be pruned by Algorithm 1 so that its real performance is much better than the worst case.

4.4 Optimizations

Although many unnecessary object pairs can be pruned by PB-tree, the pruning rule in Algorithm 3 is still loose and therefore many index nodes could not be pruned at early time. If we consider Algorithms 1 and 3 together, $\hat{H}(n_1, n_2)$ is regarded as the upper bound of $EI(\mathcal{S}_k|\mathcal{P}_1)$, where $P_1 = (o_1, o_2)$, and o_1 and o_2 are two arbitrary objects under n_1 and n_2, respectively. Since $EI(\mathcal{S}_k|\mathcal{P}_1) = H(A(\mathcal{P}_1)) - \Delta(A(\mathcal{P}_1))$, such an upper bound can be further tightened if it also considers $\Delta(A(\mathcal{P}_1))$ as below.

Let us revisit Algorithms 1 and 3. The searching process will be stopped if EI_{max} is larger than $H(E(\mathcal{P}_1^h))$, where $\mathcal{P}_1^h$ is the first object pair of $\mathcal{OP}$ in Algorithm 3. On the other hand, the upper bound of $EI(\mathcal{S}_k|\mathcal{P}_1)$ can also be computed by the bound objects of n_1 and n_2, where (n_1, n_2) is the first element of $\mathcal{NP}$ in Algorithm 3. As such, we replace $\hat{H}(n_1, n_2)$ in Algorithms 2 and 3 by such upper bounds (denoted by $\hat{EI}(n_1, n_2)$). Also, the node pairs in $\mathcal{NP}$ are sorted by $\hat{EI}(n_1, n_2)$, which will make some index node pairs pruned early. In the following theorem, we propose an efficient method to derive $\hat{EI}(n_1, n_2)$.

Theorem 4: Given two PB-tree nodes n_1 and n_2, let i_{u1}^b and i_{u2}^b denote the largest instances of $n_1.ubo$ and $n_2.ubo$, i_{l1}^b and i_{l2}^b denote the smallest instances of $n_1.lbo$ and $n_2.lbo$. Further let i_{u1}, i_{u2}, i_{l1}, and i_{l2} denote the instances under n_1 and n_2 that contribute to i_{u1}^b, i_{u2}^b, i_{l1}^b, and i_{l2}^b, respectively. $\hat{EI}(n_1, n_2)$ can be assigned as follows:

$$\hat{EI}(n_1, n_2) = \hat{H}(n_1, n_2) \times (1 - \frac{PT_k(i_{u1}, i_{u2})}{P(i_{u1}) \times P(i_{u2})} - \frac{NPT_k(i_{l1}, i_{l2})}{P(i_{l1}) \times P(i_{l2})}). \quad (18)$$

Proof. Omitted due to space limitations. See Appendix G for detailed proof.

4.5 Extension to Order-Sensitive Crowdsourcing Problem

Similar to the order-insensitive setting, Eq. (11), *i.e.*, $EI(\mathcal{S}_k|\mathcal{P}_n) = H(A(\mathcal{P}_n)) - \Delta(A(\mathcal{P}_n))$, still works in the order-sensitive setting. As such, the derivation of $H(A(\mathcal{P}_1))$ is the same as that of the order-insensitive setting, except to modify the derivation of $\Delta(A(\mathcal{P}_1))$. Let s be an arbitrary k-object combination in $\mathcal{S}_k$. As discussed in Section 4.2, s can be divided into four cases: S_1, S_2, $S_{1,2}$, and $S_\emptyset$. While S_1 and S_2 can be omitted due to the same reason as in the order-insensitive setting, $S_{1,2}$ can also be ignored because either "$o_1 > o_2$" or "$o_1 > o_2$" holds. If "$o_1 > o_2$" holds, $P(s, o_1 < o_2) = 0$ and $h(P(s, o_1 > o_2)) = h(P(s))$, which means $h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s)) = 0$. A similar rule works if "$o_1 < o_2$" holds. Thus, we only need to consider $S_\emptyset$, *i.e.*, $\Delta(A(\mathcal{P}_1)) = \sum_{s \in S_\emptyset}(h(P(s, o_1 > o_2)) + h(P(s, o_1 < o_2)) - h(P(s)) = \Delta_\emptyset(o_1, o_2)$. Finally, we can still use Algorithm 5 to derive $\Delta_\emptyset(o_1, o_2)$ in the order-sensitive setting.

5 Multi-Quota Selection Problem

The main drawback of single-quota crowdsourcing is that it can only post one pair of objects at a time, whereas usually multiple object pairs need to be outsourced to gain significant quality improvement. Sequentially crowdsourcing them one by one is inefficient because of the high latency incurred in the crowdsourcing process. An alternative, therefore, is to select a batch of object pairs and crowdsource them altogether. As such, the multi-quota selection problem, as a reiteration of Definition 3, is to select n object pairs that can achieve the highest excepted quality improvement $EI(\mathcal{S}_k|\mathcal{P}_n)$.

From Eq. (11), to find the optimal $EI(\mathcal{S}_k|\mathcal{P}_n)$ we need to enumerate all combinations of n object pairs. Obviously, the cost is prohibitively high for large k and n, as shown in the following theorem.

Theorem 5: Let $O(X_k)$ be the asymptotic complexity to compute $EI(\mathcal{S}_k|\mathcal{P}_n)$ for an object pair set $\mathcal{P}_n$ with n pairs, then the asymptotic complexity for finding the optimal $EI(\mathcal{S}_k\ |\mathcal{P}_n)$ is $O(n!m^{2n}X_k)$. Here $n \ll m$, the number of uncertain objects.

Proof. We prove this by mathematical induction. Let the theorem be true for $t \leq n$, that is, the complexity is $O(n!m^{2n}X_k)$ and $\mathcal{P}_n^*$ denotes the n object pairs that lead to this optimality. The key observation is that the optimal $\mathcal{P}_{n+1}^*$ has the property that one of its subset must be the optimal object pairs of its size. As such, the optimal $\mathcal{P}_{n+1}^*$ can only be among these object pair combinations: $\mathcal{P}_n^* \cup \Pi_m^1$, $\mathcal{P}_{n-1}^* \cup \Pi_m^2$, $\cdots$, $\mathcal{P}_1^* \cup \Pi_m^n$, where Π_m^i denotes all possible combinations of i pairs from m objects. By replacing $\mathcal{P}_i^*$ with $O(i!m^{2i}X_k)$ and Π_m^i with m^{2i} in the above combinations and summing them up, the asymptotic complexity of finding $\mathcal{P}_{n+1}^*$ is $O((n+1)!m^{2n+2}X_k)$. $\square$

In what follows, we propose two heuristic algorithms to find an approximate solution. First, we can use the single-quota selection algorithm to select top-t object pairs with the highest expected quality improvement. This can be achieved by slightly modifying the stop condition in Algorithm 1 as to find t object pairs with larger quality improvement than the next object pair. Although this is efficient, the disadvantage is that some pairs may contain the same object, which reduces their joint quality improvement. For example, given t=2, the top-2 object pairs with the highest expected quality improvement are (o_1, o_2) and (o_2, o_3) independently. However, by joining these two pairs, we may not achieve the highest quality improvement as the information about the object o_2 from both pairs overlaps.

The second heuristic greedily finds the next pair that achieves the largest joint quality improvement with currently selected object pairs, and it stops until all t quota is used up. In this heuristic, the key problem is to find that next object pair. Formally, assuming j object pairs (denoted by $\mathcal{P}_j$) have been selected so far, the problem is to find the next object pair $\mathcal{P}_1$ that maximizes $EI(\mathcal{S}_k|(\mathcal{P}_j + \mathcal{P}_1))$. According to Eq. (11), $EI(\mathcal{S}_k|(\mathcal{P}_j + \mathcal{P}_1)) = H(E(\mathcal{P}_j + \mathcal{P}_1)) - \Delta(E(\mathcal{P}_j + \mathcal{P}_1))$. For the object pairs at the top, $\Delta(E(\mathcal{P}_j + \mathcal{P}_1))$ is much smaller than $H(E(\mathcal{P}_j + \mathcal{P}_1))$, so we can use $\Delta(E(\mathcal{P}_j))$ + $\Delta(A(\mathcal{P}_1))$ to approximate it. For $H(E(\mathcal{P}_j + \mathcal{P}_1))$, we can compute it by aggregating the probabilities of 2^{j+1} combinations of these object pairs. However, the computational complexity is $O(2^j)$, which is large if j grows up. To speed up, we exploit a property of entropy — if events are independent, their joint entropy is equal to the sum of individual entropies of each event. Since two object pairs are independent when they do not share an object, we can divide $j+1$ pairs into disjoint subsets that do not share objects each other. This effectively reduces the time complexity from $O(2^j)$ to $O(2^x)$, where x is the maximum cardinality of these subsets.

6 Performance Evaluation

6.1 Experiment Setup

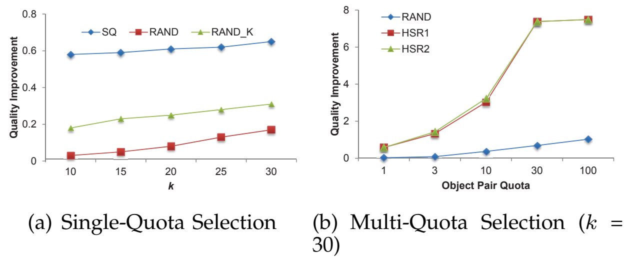
In this section, we conduct experimental studies to show the effectiveness and efficiency of our proposed algorithms. The experiments aim to: (1) evaluate the effectiveness of our pairwise crowdsourcing model to result quality improvement for top-k queries on a real crowdsourcing platform; (2) evaluate the efficiency and scalability of our proposed object-pair selection algorithms. We perform experiments on three datasets:

- **AGE**: We use the data from the AgeGuessing website [1], which provides photos of people for users to guess their ages. We crawled 600 photos and the associated guessed answers as the uncertain dataset. Each photo can be regarded as an uncertain object and each answer as an instance (with a guessed age being the value and its percentage being the probability). On average, each object contains 8 instances.
- **IMDB**: This dataset contains the user ratings (*i.e.*, instances) of 4,999 movies (*i.e.*, uncertain objects) from imdb.com [2]. Each movie contains 2 ratings on average, and each rating is associated with a confidence, which is treated as the probability of this rating instance.
- **SYN**: Besides the real AGE and IMDB datasets, we generate a synthetic dataset SYN mainly for the efficiency and scalability tests. The dataset contains 100,000 synthesized uncertain objects, each with 3

instances on average. More specifically, for each object we create a random cluster of values in an interval of 50 from [0, 10000]. The probabilities of these values follow a skewed distribution.

The experiments were executed on a laptop (Intel Core i5 2.5GHz CPU and 8GB RAM) running Mac OS X 10.8.5 operating system. The codes were written in Java (JDK 1.6).

6.2 Effectiveness (Real Crowdsourcing)



(a) Single-Quota Selection (b) Multi-Quota Selection (k = 30)

Fig. 6. Experiments on Real Crowdsourcing

In this experiment, for the single-quota setting, we investigate the quality improvement by cleaning a single object pair (labeled by "SQ"). The quality improvement is the difference between the query result qualities before and after cleaning, where the query result quality is computed by Eq. 4. For some large k, to reduce the computational cost, we approximate the quality by omitting some possible worlds with extremely low probabilities. For comparison, we also show the quality improvement of a random selection method (labeled by "RAND"), which is the average out of 100 randomly selected object pairs. An improved version of this method, labeled by "RAND_K", selects the object pairs only from the top 20% highest probable objects. For the multi-quota setting, we adopt two heuristics introduced in Section 5. "HRS1" represents the method which selects top-m object pairs with the highest expected quality improvement, while "HRS2" is based on the greedy algorithm.

We conducted the experiments on the real crowdsourcing platform, Amazon Mechanical Turk (AMT for short)[3] for dataset AGE. With the object pair selection algorithm developed in Sections 4 and 5, we selected pairs of photos and posted them on AMT. The workers (photo experts) in AMT compared the ages in pair of photos and selected the photo with an elder age. Each pair was assigned to 10 workers and each worker can get \$0.01 as the reward. We then measured the quality improvement brought by these comparison results. We used the probability distribution of the crowdsourcing results as $P(o_x > o_y)$ and $P(o_x < o_y)$ to compute the actual quality in Eq. (6). As shown in Fig. 6, both the single-quota selection and multi-quota selection algorithms significantly outperform the random methods. We also observed that the results from the crowd follow almost the same probability distribution as the original dataset from the AgeGuessing website, but with a shift of its mean by a probability bias. In our results, the average value of this bias is 0.19. This observation will be used in the following simulation-based experiments.

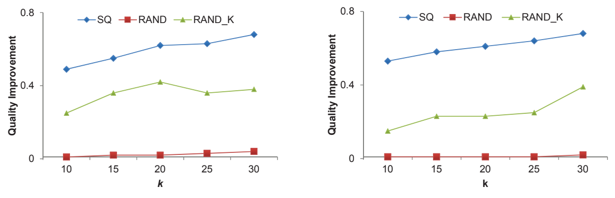
3. https://www.mturk.com

6.3 Effectiveness (Simulation)

In this subsection, we evaluate the effectiveness of our pairwise-comparison solution by simulating the crowd's behaviors on datasets IMDB and SYN, which are significantly larger than the AGE dataset used in the real experiment. According to our observation in Section 6.2, we assume that the results from the crowd follow the same probability distribution as the original data, added by a probability distribution bias θ (set to 0.19 by default). That is, for objects o_x and o_y, the crowd will return $o_x > o_y$ with the probability P_{real} defined in Eq. (19), and vice versa.

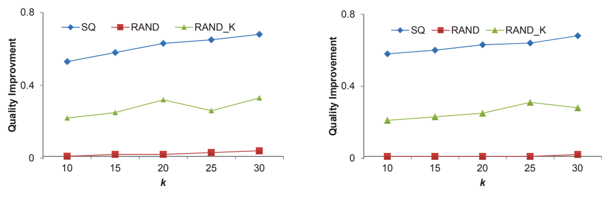
$$P_{real} = \begin{cases} \min\{1, P(o_x > o_y) + \theta\} & \text{if } P(o_x > o_y) > 0.5, \\ \max\{0, P(o_x > o_y) - \theta\} & \text{if } P(o_x > o_y) < 0.5. \end{cases} \quad (19)$$

6.3.1 Quality Improvement by Single Quota



(a) IMDB (b) SYN

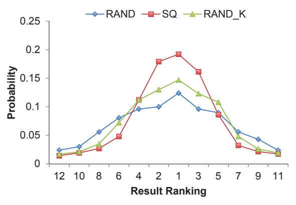
Fig. 7. Quality Improvement of Single Quota (Order-Insensitive)



(a) IMDB (b) SYN

Fig. 8. Quality Improvement of Single Quota (Order-Sensitive)

We conduct the experiments under both the order-sensitive and order-insensitive top-k definitions; the results are shown in Figs. 7 and 8, respectively. In Fig. 7, we can see that the quality improvement of SQ is about two times larger than that of RAND_K, while RAND hardly gains any improvement. As k grows up, the performance of RAND becomes slightly better as the probability of the selected pair hitting the top-k result increases. Fig. 8 exhibits similar trends, except that the absolute values of each experiment are larger than those under the order-insensitive definition. This is because the order-sensitivity brings a larger degree of diversity. In the interest of space, we only show the results under the order-insensitive setting in the sequel.


(a) IMDB

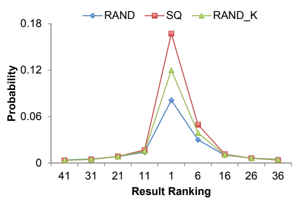
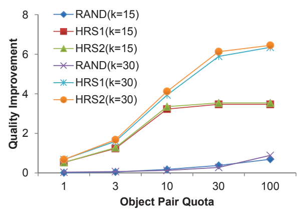

(b) SYN

Fig. 9. Probability Distribution of Top-k Results

6.3.2 *Meaning of Quality Improvement*

To show the direct consequence of quality improvement, Fig. 9 plots the probability distribution of top-ranked results after crowdsourcing. The horizontal axis represents the ranking of the result and the vertical axis is its corresponding probability. We can see that the probabilities of the SQ results are more centralized than those of RAND_K and RAND, which means that users can have higher confidence to identify a high ranking result from others.

6.3.3 *Quality Improvement by Multiple Quota*


(a) IMDB

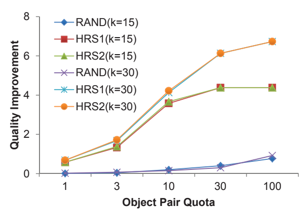
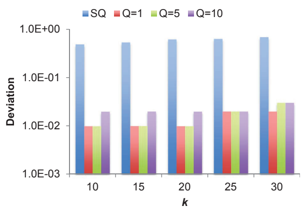

(b) SYN

Fig. 10. Effect of Object Pair Quota

In this subsection, we investigate the quality improvement under the multi-quota setting by varying k and the quota number. In Fig. 10, we can see that both HRS1 and HRS2 outperform RAND, while HRS2 performs slightly better than HRS1. As the object-pair quota grows up, the quality improvement increases as well. However, it starts to saturate after the quota reaches a certain value, which is called the *convergence value*. As shown in Fig. 10, for larger k, the convergence value is also larger because it brings more diversity. For the same reason, the convergence values in SYN are also larger than those in IMDB.

6.3.4 *Deviation of $\Delta(A(\mathcal{P}_1))$*


(a) AGE

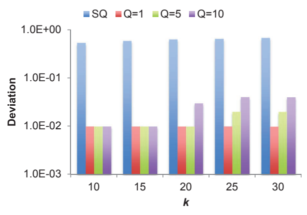

(b) IMDB

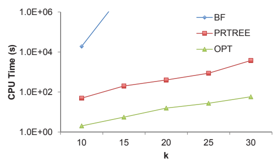
Fig. 11. Deviation of $\Delta(A(\mathcal{P}_1))$

In this subsection, we evaluate the deviation of approximated $\Delta(A(\mathcal{P}_1))$ from the final result. In Fig. 11, we list top-Q object pairs with the highest expected quality improvement and calculate the average deviation of $\Delta(A(\mathcal{P}_1))$, *i.e.*, the difference between the upper bound and lower bound of $\Delta(A(\mathcal{P}_1))$. For comparison, we also list the quality improvement of single quota (labeled by "SQ"). Fig. 11 shows that as the value of Q enlarges, the deviation of $\Delta(A(\mathcal{P}_1))$ slightly increases. It is because the object pairs with higher expected quality improvement tend to have lower $\Delta(A(\mathcal{P}_1))$, which subsequently leads to less deviation. We also observe that even when Q grows up to 10, the deviation is still much less than SQ and therefore can be ignored.

6.4 Efficiency

One of the main contribution of this paper to optimize the efficiency of the proposed algorithms. We developed several techniques including PB-tree, bound-based method and optimization in Section 4.4. In this subsection, we evaluate the performance of these techniques. We assume all data are preloaded into the main memory and therefore the main metric of the efficiency is the elapsed time of algorithm execution. The two algorithms under evaluation are "PBTREE" and "OPT." The former uses PB-tree to sort the object pairs in the descending order of $H(A(\mathcal{P}_1))$, whereas the latter adopts the optimization in Section 4.4. As a baseline algorithm, we also show the results of the brute-force algorithm (labeled by "BF"), which does not adopt any optimization.

6.4.1 *Overall Elapsed Time*


(a) AGE

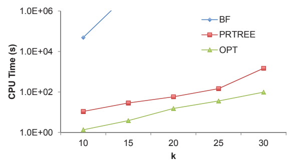
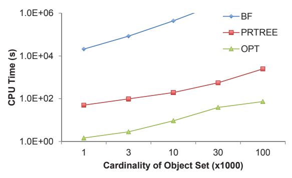

(b) IMDB

Fig. 12. Overall Elapsed Time

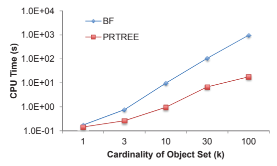
First, we conduct the experiments on datasets AGE and IMDB. Fig. 12 shows as k increase, the performance of BF degrades significantly since the computational cost of $\Delta(A(\mathcal{P}_1))$ becomes very huge (see Section 6.4.2). If k is set to 15, it will take over 10^6 seconds, which makes BF impractical. The elapsed time of PBTREE and OPT are both acceptable, while the latter performs the best since it prunes more node pairs which do not contain top-k results.

6.4.2 *Scalability*

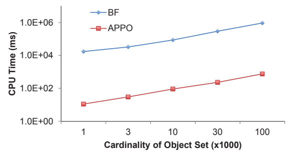
In this subsection, we examine the scalability of our proposed algorithms. We evaluate these algorithms on the synthesized data where we vary the cardinality of object set. Fig. 13(a) shows the results. As the cardinality increases, BF becomes very inefficient, while PBTREE and OPT are still acceptable. To explain the reason, we break down the time cost of each step. First, we measure the time to sort the object pairs by their $H(A(\mathcal{P}_1))$. The brute-force algorithm has to consider every object pair, which takes O(n^2) time. Fig. 13(b) shows that compared with BF, PBTREE only needs 1% of the total



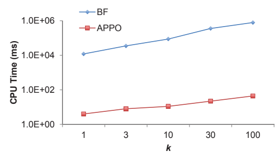
(a) Overall Performance



(b) Effect of PB-tree



(c) Effect of $\Delta(A(\mathcal{P}_1))$ Derivation (varying data cardinality)



(d) Effect of $\Delta(A(\mathcal{P}_1))$ Derivation (varying k)

Fig. 13. Scalability

CPU time when the cardinality is 100K. Second, we also test the effectiveness of $\Delta(A(\mathcal{P}_1))$ derivation introduced in Section 4.2. Since the number of possible top-k results is very large, our bound-based algorithm omits those results with low probabilities. By contrast, the brute-force algorithm calculates all top-k results by using the method developed in [29]. Fig. 13(c) and 13(d) show the average time cost in calculating $\Delta(A(\mathcal{P}_1))$ of one object pair under various cardinality and k settings. By default, we set k to 15 and the number of objects to 10K. The results show that BF uses much more time than our bound-based method, which is the main reason that BF is inferior in terms of the overall elapsed time.

7 Conclusions and Future Work

In this paper, we have proposed a new pairwise crowdsourcing model to reduce the uncertainty of top-k query results on probabilistic databases. We defined the quality of object-level top-k ranking in terms of entropy and formulated the optimal object-pair selection problem for maximum quality improvement. We devised a PB-tree index which facilitates the selection of object pairs under both single-quota and multi-quota budgets, followed by a bound-based method for quality improvement estimation. We have also extended the proposed solutions to order-sensitive settings. Extensive experimental results demonstrate the effectiveness and efficiency of our proposed models and algorithms under various system settings. In particular, our proposed pairwise selection solution outperforms the random selection method by up to 30 times in terms of quality improvement. In terms of efficiency, our proposed algorithms and optimizations reduce the elapsed time of the brute-force algorithm from several days to only one minute.

As for future work, we will extend this work to other probabilistic queries. First, we plan to extend our model to explore the correlations between the outcomes of objects. For example, two "*similar*" objects are likely to have similar values. Thus, an interesting idea is to cluster the objects and select representatives from each cluster for pairwise cleaning. Second, we plan to adapt our pairwise crowdsourcing model to reduce the uncertainty of probabilistic ranking-aggregate queries, which rank object groups by their aggregated values. Since such queries should consider the aggregated value of the whole group, it is more sophisticated than an ordinary probabilistic top-k query. Third, we are going to extend our model to improve the quality of probabilistic graph queries. Since the edges in a graph reflect the relationships among the nodes, it is promising to exploit crowdsourcing to determine whether such relationships exist.

Acknowledgement

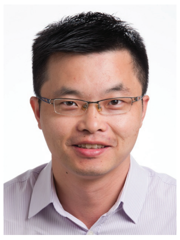
This work was supported by National Natural Science Foundation of China (Grant No.: 61572413, 61572193 and U1636205), NSF of Shanghai (No.: 17ZR1444900), and Research Grants Council, Hong Kong SAR, China, under projects 12244916, 12201615, 12202414, 12200914, 15238116, and C1008-16G.

References

[1] http://www.ageguess.org.
[2] http://infolab.stanford.edu/trio/code/movie_data.triql.
[3] Y. Baba and H. Kashima. Statistical quality estimation for general crowdsourcing tasks. In *KDD*, 2013.
[4] T. Bernecker, H. Kriegel, N. Mamoulis, M. Renz, and A. Zuefle. Scalable probabilistic similarity ranking in uncertain databases. *TKDE*, 22(9): 1234-1246, 2010.
[5] R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees. *PVLDB*, 1(1): 722-735, 2008.
[6] E. Ciceri, P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Crowdsourcing for Top-K Query Processing over Uncertain Data. *TKDE*, 28(1): 41-52, 2016.
[7] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, 2009.
[8] X. Chu, J. Morcos, I. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, 2015.
[9] G. Demartini, D. Difallah, and P. Cudré-Mauroux. Zen-Crowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW*, 2012.
[10] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, 2013.
[11] D. Deng, G. Li, J. Feng, and W. Li. Top-k string similarity search with edit-distance constraints. In *ICDE*, 2013.
[12] N. Dalvi, A. Parameswaran, and V. Rastogi. Minimizing uncertainty in pipelines. In *NIPS*, 2005.
[13] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.
[14] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won? Dynamic max discovery with the crowd. In *SIGMOD*, 2012.
[15] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD*, 2008.
[16] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Comprehensive and reliable crowd assessment algorithms. In *ICDE*, 2015.
[17] A. Krause, and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. In *UAI*, 2005.
[18] M. van Keulen and A. de Keijzer. Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *VLDB Journal*, 18(5): 1191-1217, 2009.
[19] B. Kanagal, J. Li, and A. Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *SIGMOD*, 2011.
[20] H. Kaplan, I. Lotosh, T. Milo, and S. Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9): 697-708, 2013.

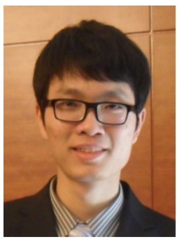
[21] X. Lin, Y. Peng, B. Choi, and J. Xu. Human-powered data cleaning for probabilistic reachability queries on uncertain graphs. em TKDE, 2017.
[22] L. Mo, R. Cheng, X. Li, D. W. Cheung, and X. Yang. Cleaning uncertain data for top-k queries. In *ICDE*, 2013.
[23] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2): 109-120, 2012.
[24] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, and Samuel Madden. Scaling up crowd-sourcing to very large datasets: a case for active learning. *PVLDB*, 8(2): 125-136, 2014.
[25] A. Marcus, E. Wu, R. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *PVLDB*, 5(1): 13-24, 2011.
[26] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 7(9): 685-696, 2014.
[27] T. Saaty. Decision making with the analytic hierarchy process. *Int. J. Services Sciences*, 1(1):83-98, 2008.
[28] C. Shannon. The mathematical theory of communication. *University of Illinois Press*, 1949.
[29] M. Soliman, I. Ilyas, and K. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.
[30] Y. Tong, C. Cao, C. Zhang, Y. Li, and L. Chen. CrowdCleaner: Data cleaning for multi-version data on the web via crowdsourcing. In *ICDE*, 2014.
[31] B. Trushkowsky, T. Kraska, M. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, 2013.
[32] Vassilis, L. Alfaro, and J. Davis. Human-powered top-k lists. In *WebDB*, 2013.
[33] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071-1082, 2014.
[34] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *ICDE*, 2015.
[35] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, 2012.
[36] T. Wu, L. Chen, P. Hui, C. Zhang, and W. Li. Hear the whole story: towards the diversity of opinion in crowdsourcing markets. *PVLDB*, 8(5): 485-496, 2015.
[37] J. Wang, T. Kraska, M. Franklin, and J. Feng. CrowdER: crowdsourcing entity resolution. *PVLDB*, 5(11):1483-1494, 2012.
[38] J. Wang, G. Li, T. Kraska, M. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.
[39] S. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6): 349-360, 2013.
[40] M. Yiu and N. Mamoulis. Multi-dimensional top-k dominating queries. In *VLDB Journal*, 18(3): 695-718, 2009.
[41] X.Yang, H. Steck,Y. Guo, and Y. Liu. On top-k recommendation using social networks. In *RecSys*, 2012.
[42] X. Zhang and J. Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *ICDE Workshop*, 2008.
[43] C. Zhang, L. Chen, H. Jagadish, and C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9): 757-768, 2013.
[44] C. Zhang, L. Chen, Y. Tong, and Z. Liu. Cleaning uncertain data with a noisy crowd. In *ICDE*, 2015.
[45] A. Zheng, I. Rish, and A. Beygelzimer. Efficient test selection in active diagnosis via entropy approximation. In *UAI*, 2005.
[46] C. Zhang, Y. Tong, and L. Chen. Where to: Crowd-aided path selection. *PVLDB*, 7(14): 2005-2016, 2014.
[47] C. Aggarwal, and P. Yu. A survey of uncertain data algorithms and applications. In *TKDE*, 21(5): 1-15, 2015.

Xin Lin received the BEng degree and PhD degree, both in computer science and engineering, from Zhejiang University, China. He is currently an associate professor in the Department of Computer Science, East China Normal University. He is also a visiting scholar in the Database Group at Hong Kong Baptist University (http://www.comp.hkbu.edu.hk/∼db/). His research interests include location-based services, spatial databases, and privacy-aware computing.



Jianliang Xu is a Professor in the Department of Computer Science, Hong Kong Baptist University. He received the BEng degree from Zhejiang University and the PhD degree from Hong Kong University of Science and Technology. He held visiting positions at Pennsylvania State University and Fudan University. His current research interests include data management, mobile and spatial databases, data security and privacy. He has published more than 150 technical papers in these areas. He has served as a program co-chair/vice chair for a number of major international conferences including IEEE ICDCS 2012, IEEE CPSNA 2015 and WAIM 2016. He is an Associate Editor of IEEE Transactions on Knowledge and Data Engineering (TKDE) and PVLDB 2018.

Haibo Hu is an assistant professor in the Department of Electronic and Information Engineering, Hong Kong Polytechnic University. His research interests include information security, privacy-aware computing, wireless data management, and location-based services. He has published over 60 research papers in refereed journals, international conferences, and book chapters. As principal investigator, he has received over 5 million HK dollars of external research grants from Hong Kong and mainland China. He is the recipient of ACM-HK Best PhD Paper Award, Microsoft Imagine Cup, and GS1 Internet of Things Award.



Fan Zhe is a PhD student in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in Computer Science from South China University of Technology in 2011. His research interests include database system optimization. He is a member of the Database Group at Hong Kong Baptist University.