Pairwise Ranking Aggregation by Non-interactive Crowdsourcing with Budget Constraints

Changjiang Cai*, Haipei Sun*, Boxiang Dong†, Bo Zhang*, Ting Wang‡, Hui (Wendy) Wang*

*Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ
{ccai1,hsun15,bzhang31,Hui.Wang}@stevens.edu

†Department of Computer Science
Montclair State University
Montclair, NJ
bxdong7@gmail.com

‡Department of Computer Science
Lehigh University
Bethlehem, PA
ting@cse.lehigh.edu

***Abstract*—Crowdsourced ranking algorithms ask the crowd to compare the objects and infer the full ranking based on the crowdsourced pairwise comparison results. In this paper, we consider the setting in which the task requester is equipped with a limited budget that can afford only a small number of pairwise comparisons by the crowd. We consider the *non-interactive* crowdsourcing setting in which the task requester assigns the tasks to the crowd *only once*. This is different from most of the existing crowdsourced ranking work that consider the *interactive* setting in which the task requester keeps on assigning the tasks to the crowd and collecting the workers' feedback in a continuous fashion. Based on the transitive property of pairwise comparisons, we design solutions that enable the task requester to obtain a good-quality full ranking from a small number of pairwise preferences. Our approach consists of two steps, namely *task assignment* and *result inference*. For the task assignment step, we design efficient algorithms that generate *fair* pairwise comparison tasks that enable to construct a full ranking from a small number of pairwise preferences with high probability. For the truth inference step, we leverage the truth discovery algorithm to aggregate the possibly conflicting pairwise preferences with the workers' quality taken into consideration. We also design both exact and heuristic algorithms to find the best full ranking from the pairwise preferences. By using simulated datasets and real-world crowdsourcing preference data collected from Amazon Mechanical Turk, we demonstrate that our approach can construct good-quality full rankings from non-expert crowd in the non-interactive fashion.**

I. Introduction

Crowdsourcing has emerged over the last few years as an important new tool for a variety of tasks, ranging from micro-tasks on Amazon's Mechanical Turk (AMT) to big innovations such as Netflix Challenge [1]. Among various crowdsourcing platforms, AMT establishes a marketplace that coordinates the supply and demand of tasks that require human intelligence to complete. It is an online labor market where employees (called *workers*) are recruited by employers (called *requesters*) for the execution of tasks (called *HITs*, acronym for Human Intelligence Tasks) in exchange for a wage (called a *reward*). Typical tasks on AMT include relevance feedback, document labeling, and some other small tasks that would be challenging for the computers to accomplish. It enables faster task completion and cheaper cost than in-house solutions.

Among various types of crowdsourcing tasks, ranking objects according to a specific criteria have witnessed many applications (e.g., image processing and data cleaning). The setting is as follows: given a set of n objects $\mathcal{O}$, the requester wishes to get a *full* ranking of $\mathcal{O}$, i.e., a *total order* of $\mathcal{O}$, from the ranking preferences collected from the crowd. Crowdsourced ranking has been widely used in many real-world applications where the objects are rather hard to be compared by machines but easy for human. An example of such applications is to rank images based on their subjective image quality [2].

There are two widely-used ways to generate the full ranking. The first approach is *rating-based* ranking, which asks the crowd to assign a rate for each object. Then the full ranking is generated by aggregating the rates and sorting the aggregated rates in a particular order. The second approach is *pairwise comparison based* ranking, which asks the crowd to compare pairs of objects and choose the preferred one. The full ranking is aggregated from the pairwise comparison results. The rating-based method has some weakness, e.g., it is difficult for the crowd to assign an accurate rate. Thus the rating-based method usually has a lower accuracy than the pairwise comparison method [3]. Therefore, in this paper, we focus on the pairwise comparison based approach.

To collect the pairwise preferences, a straightforward solution is that the task requester asks the crowd to compare all $\binom{n}{2}$ pairs of n objects. However, in practice, the requester may be equipped with a limited budget that is not able to afford all pairwise comparisons. Therefore, the requester has to select $\ell < \binom{n}{2}$ pairs for comparison, where ℓ is decided by the budget.

We assume that the requester has no ground truth of the final ranking of $\mathcal{O}$. Therefore, the requester desires that by harnessing fewer pairwise preferences, he still can obtain a ranking result that is reasonably good. However, the crowd may return noisy and conflicting pairwise preferences. Therefore, one of the challenges is to infer a trustworthy full ranking by identifying reliable workers. There has been growing interest from both the theory and machine learning communities in developing algorithms and techniques to compute accurate ranking results while minimize the set of unreliable crowd judgment. Simple solutions typically use heuristic methods such as majority voting or weighted majority voting [4]. However, these methods do not consider the reliability of different workers and they treat all judgments as equally reliable. More sophisticated methods such as heuristic-based solutions (e.g., [5], [6]) and learning-based methods (e.g., [7], [8]) are designed to integrate the workers' quality with the result inference. Most of these work focus on *continuous, interactive* crowdsourcing; the task requester keeps on assigning the tasks to the crowd and collecting the workers' feedback, until either the crowdsourcing budget is used up or the aggregated ranking result reaches the expected accuracy [9]. However, for time-sensitive tasks (e.g., traffic monitoring, location-aware surveys, and points-of-interest suggestions), the task requester has strict time constraints to finish the task, and thus cannot perform crowdsourcing in the continuous fashion. Instead, the requester releases the tasks to the crowd only once, and accepts the results generated from the collected answers. We call this setting as the *non-interactive* crowdsourcing.

Compared with the *interactive* setting, the *non-interactive* crowdsourced ranking is more challenging as the task requester has to maximize the probability of obtaining a good ranking result by one single round of crowdsourcing. This is possible when the ranking preferences have the transitive property (i.e., if the object O_i is preferred to O_j, and O_j is preferred to O_k, then it can be inferred that O_i is preferred to O_k). The transitivity property can save the unnecessary comparison between O_i and O_k, as long as the comparison results between O_i and O_j, as well as O_j and O_k, have high quality.

In this paper, we consider the non-interactive crowdsourced ranking setting that allows transitive pairwise comparisons. Most existing crowdsourcing ranking solutions employ a two-step strategy. The first step (called *task assignment*) selects ℓ object pairs and generates HITs to crowdsource the pairwise comparisons of these objects to the crowd. The second step (called *result inference*) infers the ranking based on the pairwise comparison results collected from the crowd. We follow this two-step strategy and make the following contributions.

First, for the *task assignment* step, we design a graph model to represent the pairwise comparison tasks as a *task graph*, and the workers' preferences as a *preference graph*. The preference graph can be considered as a directed, weighted instance of the task graph. The reachability relation of the preference graph constructs its transitive closure. A full ranking of objects is equivalent to a Hamiltonian path (HP) of the transitive closure of the preference graph. We design an efficient *task assignment* scheme that constructs the pairwise comparison tasks that guarantee: (1) *fairness*, meaning that each object has equal probability to be at any position in the HP; (2) *high HP-likelihood*, meaning that it is highly likely that there exists at least one HP in the transitive closure of the preference graph; and (3) *budget-conscious*, i.e., the number of edges in the task graph is decided by the given budget.

Second, we split the *result inference* step into the following four sub-steps: (1) Infer the *direct* pairwise preferences by aggregating the (possibly conflicting) pairwise preferences by the crowd; (2) *Smooth* the direct pairwise preferences to ensure there is at least one HP (i.e., a full ranking); (3) From the smoothed direct preference, compute the *indirect* pairwise preferences that are inferred by transitivity, and aggregate both direct and indirect pairwise preferences; and (4) Compute the preference of the full rankings aggregated from pairwse preferences, and pick the full ranking of the highest preference as the final output. For Step (1), we design a *truth discovery* method that combines the estimation of workers' quality and the inference of true pairwise preference together to resolve (possible) conflicts among the workers. For Step (2), we apply *preference smoothing* to adjust the direct pairwise preferences based on the workers' pairwise preferences as well as their estimated quality by Step (1). For Step (3), we design an efficient *preference propagation* method that computes the indirect pairwise preferences inferred by preference transitivity. For Step (4), we design both exact and heuristic methods to find the full ranking of the highest preference. The exact solution uses an early-termination condition so that it can find the full ranking of top-1 probabilistic preference without examining all paths. The heuristic method utilizes *simulated annealing* to approximate the global optimal probabilistic preference.

Last but not least, we perform an extensive set of experiments by using pairwise preferences collected from the workers on Amazon Mechanical Turk (AMT) as well as from the simulated setting. The results show that the ranking accuracy of our method is reasonably good; the ranking accuracy is at least 0.89 (out of 1) for 100 objects when only 10% pairwise comparisons are picked. It can achieve 0.95 when the number of objects grows to 1000 (the same 10% selection ratio). Furthermore, the ranking accuracy of our method shows higher accuracy and faster rank inference than the *interactive* crowdsourcing setting when it requires to rank a large number of objects by low-quality workers with small budgets.

The paper is organized as follows. Sec. II discusses the preliminaries. Sec. III introduces the graph model. Sec. IV discusses the task generation approach. Sec. V discusses the result inference algorithms. Sec. VI presents our experimental results. Sec. VII discusses the related work. Sec. VIII concludes the paper.

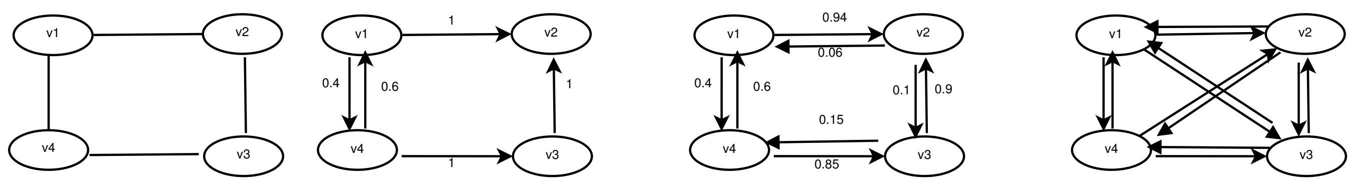
II. Preliminaries

Ranking Setting. In this paper, we consider the ranking problem as finding a *full* ranking (permutation) π of a set of n objects $\mathcal{O}$, by combining ℓ input pairwise preferences, where $\ell < \binom{n}{2}$. No tie is allowed in π. Each pairwise preference specifies that either O_i is preferred to O_j (denoted as $O_i \prec O_j$), or vice versa. O_i is ranked before O_j if $O_i \prec O_j$. We assume that the preference ranking is *transitive*, i.e., if $O_i \prec O_j$ and $O_j \prec O_k$, then $O_i \prec O_k$.

Crowdsourcing Setting. The requester creates a number of human intelligence tasks (HITs) $\mathcal{T}$ off-line, and assigns $\mathcal{T}$ to $m > 1$ workers $\mathcal{W}$ via a crowdsourcing platform (e.g., Amazon Mechanic Turk). Each HIT $T \in \mathcal{T}$ consists of $c \geq 1$ pairwise comparisons. There are ℓ unique pairwise comparisons in total in $\mathcal{T}$. For each pairwise comparison (O_i, O_j), we assume the workers vote for either $O_i \prec O_j$ or $O_j \prec O_i$. The workers' preferences may be inconsistent, e.g., the worker w_i votes that $O_i \prec O_j$, while the worker w_j votes that $O_j \prec O_i$. The same pairwise comparison may appear in multiple HITs. To increase the accuracy, each HIT is assigned to $w > 1$ workers, where $w \leq m$. We assume that the ground truth of ranking is not available to the requester.

The workers receive a *reward* for their ranking. In this paper, we assume that each pairwise comparison receives a reward r, which is the same for all workers. We assume that the requester has a budget B for crowdsourcing, where B cannot afford all pairwise comparisons of $\mathcal{O}$. Since each pairwise comparison is performed by w workers, the total number of unique pairwise comparison tasks equals $\ell = [\frac{B}{wr}]$.

In this paper, we assume that the HIT assignment is a *one-time* process, meaning that the task requester crowdsources



(a) Task graph G_T (b) Preference graph G_P (c) Preference graph with preference smoothing $\widetilde{G_P}$ (d) Transitive closure G_P^* of $\widetilde{G_P}$ (edge weight not specified)

Fig. 1: An example of task graph & preference graph

the tasks only once, and accepts the ranking result from the collected answers, even if the ranking result may not be of high accuracy. Such one-time setting applies to many time-sensitive crowdsourcing tasks, e.g., time-sensitive Web image ranking [10]. This setting is fundamentally different from most of the existing crowdsourced ranking work (e.g. [11], [12]), which consider an *interactive* setting in which the task requester is allowed to assign the tasks to the crowd iteratively, until either the budget runs out or the ranking result is sufficiently good. Compared with the interactive setting, our task assignment problem is more challenging as the task requester has to maximize the probability of obtaining a good ranking result by one single round of crowdsourcing.

III. Graph Model

Task graph. Given n objects $\mathcal{O}$ and ℓ pairwise comparison tasks $\mathcal{T}$, $\mathcal{T}$ can be modeled as a *task graph*, which is a *un-weighted*, *un-directed* graph G_T that consists of n vertices and ℓ edges: (1) each object $O \in \mathcal{O}$ corresponds to a vertex $v \in G_T$; (2) each pairwise comparison $(O_i, O_j) \in \mathcal{T}$ corresponds to an edge $(v_i, v_j) \in G_T$, where v_i and v_j correspond to O_i and O_j. The *degree* of the vertex $v_i \in G_T$ is the total number of edges incident to v_i. Figure 1 (a) shows an example of the task graph G_T; each vertex has degree 2.

Preference graph. We model the crowd's ranking preferences by the *preference graph*. Formally, given n objects $\mathcal{O}$ and ℓ pairwise preferences, its corresponding *preference graph* is a *weighted*, *directed* graph G_P that consists of n vertices and ℓ edges such that: (1) each object $O \in \mathcal{O}$ corresponds to a vertex $v \in G_P$; (2) each preference $O_i \prec O_j$ corresponds to an edge $v_i \to v_j$ in G_P, where v_i and v_j correspond to the objects O_i and O_j. Each edge $v_i \to v_j$ is associated with a weight $w_{ij} \in (0, 1]$, indicating the truth confidence of $O_i \prec O_j$. How to calculate w_{ij} will be explained in Section V-A. We must note that when $w_{ij} = 0$, there is no edge $(v_i, v_j) \in G_P$. We say a node $v \in G_P$ is an *in-node* (*out-node*, resp.) if v only has incoming edges (outgoing edges, resp.). Figure 1 (b) shows an example of the preference graph G_P. In this graph, the vertex v_2 is an in-node.

Due to the transitive property of pairwise comparisons, a new edge (v_i, v_j) can be added to G_P if there exists a path $P(v_i, \dots, v_j)$ in G_P. As we focus on the full ranking of the n objects, we only consider the paths in the preference graph whose length is no more than $n - 1$. We use $\widetilde{G_P}$ to represent G_P after preference smoothing (i.e., all edges of weights 1 are converted to the edges with smoothed weights), and G_P^* to denote the transitive closure (i.e., the reachability relation) of $\widetilde{G_P}$. How to construct $\widetilde{G_P}$ from the preference graph G_P will be explained in Section V. Figure 1 (c) shows an example of $\widetilde{G_P}$ by smoothing of G_P in Figure 1 (b), and Figure 1 (d) shows an example of G_P^* constructed from $\widetilde{G_P}$ in Figure 1 (c).

Hamiltonian path v.s. full ranking. Given a set of objects $\mathcal{O}$, a preference graph G_P of $\mathcal{O}$, and the transitive closure G_P^* of G_P, apparently any Hamiltonian path (HP) (i.e., a path that visits each vertex exactly once) of G_P^* corresponds to a full ranking of $\mathcal{O}$. Each HP P is associated with a *preference probability* $Pr[P]$ that measures the preference of the full ranking that is represented by P. How to compute the preference probability of HPs will be discussed in Section V. We pick the HP that is of the highest preference probability, and construct the object ranking accordingly.

From task graph to preference graph. For each edge $(v_i, v_j) \in G_T$, it has three possible permutations in G_P, $v_i \to v_j$, $v_i \leftarrow v_j$, and $v_i \rightleftharpoons v_j$ (i.e., with inconsistent preferences). Therefore, G_P can be considered as a possible weighted instance of G_T, in which each edge has a corresponding edge in G_T, whose direction takes one of the three aforementioned permutations. Therefore, given a G_T that contains ℓ edges, the number of possible instances of G_P that can be constructed from G_T is:

$$N = 3^\ell. \tag{1}$$

Consider the task graph shown in Figure 1 (a), it has $3^4 = 81$ possible instances of the preference graphs. Figure 1 (b) shows one such possible instance.

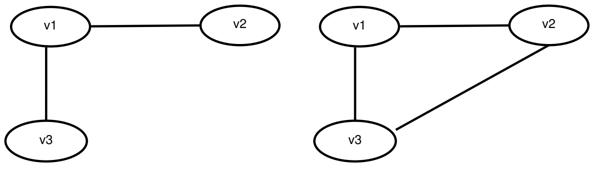
IV. Task Assignment

A. Task Assignment Requirements

Our goal is to generate the pairwise comparison tasks that guarantee: (1) *fairness*, meaning that every object has equal probability to be at any position in the HP; (2) *high HP-likelihood*, meaning that it is highly likely a full ranking can be constructed from the pairwise preferences, i.e., an HP exists in the transitive closure of the preference graph. First, we formally define task fairness and high HP-likelihood.

1) Fairness: Apparently, in-nodes and out-nodes in the preference graph G_P are special compared with other nodes, since the positions of their corresponding objects in the rankings are determined. In particular, any in-node (out-node, resp.) represents an object that should be ranked the last (the first, resp.) in the ranking. For example, consider the vertex v_2 in Figure 1 (b). It must be ranked as the least preferred object in the full ranking. Careless task assignment may lead to some objects that have higher probability to be in-/out- nodes in G_P. For instance, consider the task graph G_T in Figure 2 (a). The vertices v_2 and v_3 have higher probability to be in-/out-nodes than the vertex v_1 (the details of how to calculate the probability will be found in Example 4.1). Therefore, to ensure that each node has a fair ranking, we define *fair tasks*.

Definition 4.1: **[Task Fairness]** Given n objects $\mathcal{O}$ and ℓ pairwise comparison tasks $\mathcal{T}$ of $\mathcal{O}$, let G_T be the corresponding task graph. We say $\mathcal{T}$ is *fair* if for each vertex in G_T, it has equal probability to be an in-/out- node in the preference graph G_P of G_T.



(a) Unfair task (b) Fair task

Fig. 2: An example of (un)fair tasks

Next, we compute the probability that a given node $v_i \in G_P$ is an in-node/out-node. Let d_i be the degree of v_i in G_T. Given G_T that contains ℓ edges, there are 3^ℓ possible G_P instances of G_T (Equation 1). If v_i is an in-node, all possible instances that contain v_i only has in-edges connected with v_i. Therefore, there are $3^{\ell-d_i}$ possible instances of G_T in which v_i is an in-node. The same reasoning also holds for the out-nodes. Therefore, the probability that v_i is either an in-node or an out-node, denoted as $Prob(v_i^{IO})$, is:

$$Prob(v_i^{IO}) \quad = \quad 2 * \frac{3^{\ell-d_i}}{3^\ell} = \frac{2}{3^{d_i}}. \qquad (2)$$

Example 4.1: Consider the task graph in Figure 2 (a), following Formula (2), we can compute that $Prob(v_1^{IO}) = \frac{2}{9}$, while $Prob(v_2^{IO}) = Prob(v_3^{IO}) = \frac{2}{3}$. Consider the task graph in Figure 2 (b), all three vertices have the same probability $\frac{2}{9}$ of being in-/out-nodes.

From Equation 2, we have the following theorem.

Theorem 4.1: **(Fair Task) :** Given n objects $\mathcal{O}$ and ℓ pairwise comparison tasks $\mathcal{T}$ of $\mathcal{O}$, let G_T be the corresponding task graph. Then $\mathcal{T}$ is fair if all vertices in G_T have the same degree.

The correctness of Theorem 4.1 is straightforward as the probability of being in-/out- nodes is decided by the node degree.

2) High HP-likelihood: Since the Hamiltonian path (HP) corresponds to a full ranking, when we design the task graph G_T, we prefer that for any possible preference graph G_P constructed from G_T, its transitive closure is highly likely to have an HP. We use $Pr(G_P^*)$ to denote the probability that the transitive closure G_P^* of a given preference graph G_P has an HP. Our goal is to design the tasks $\mathcal{T}$ (and thus G_T) that is of *high HP-likelihood*, i.e., it maximizes $Pr(G_P^*)$ for any G_P constructed from G_T. First, we have the following theorem to show that the existence of HPs in the task graph G_T is a *necessary* condition of the HP existence in G_P^*.

Theorem 4.2: Given a task graph G_T, if G_T does not have an HP, then for any preference graph G_P constructed from G_T, its transitive closure G_P^* does not have any HP.

The correctness of Theorem 4.2 is straightforward. If G_T does not have an HP, there must exist at least a vertex that is not reachable by other vertices. Note that the HPs in G_T does not guarantee HPs in G_P^*.

Next, we analyze under which circumstances with a given task graph G_T that contains HPs, the transitive closure G_P^* of the preference graph constructed from G_T does not have any HP. We have the following theorem.

Theorem 4.3: If the graph G_P^* has at least two in-nodes or out-nodes, then it does not have any HP.

The correctness of Theorem 4.3 is based on the fact that for the preference graph that contains at least two in-nodes (out-nodes, resp.), the corresponding objects of these in-nodes (out-nodes, resp.) must be ties, as they are at the last (first, resp.) position in the ranking at the same time. Therefore, there does not exist a full ranking of all objects.

Following Theorem 4.3, if G_P^* has an HP, G_P^* must contain no more than one in-/out-node, given G_T that contains HPs. The following theorem shows the probability that G_P^* contains no more than one in-/out-node.

Theorem 4.4: Given a task graph G_T of n nodes, let d_{min} (d_{max}) be the minimal (maximal, resp.) node degree of G_T. Then the probability Pr that the transitive closure G_P^* of any preference graph G_P constructed from G_T does not contain more than one in-nodes/out-nodes satisfies that $Pr \geq Pr_l$, where

$$Pr_l = (1 - \frac{2}{3^{d_{min}}})^n [1 + \frac{2n}{3^{d_{max}} - 2} + \frac{n(n-1)}{2(3^{d_{max}} - 2)^2}].$$

We omit the proof of Theorem 4.4 due to the limited space. Based on Theorem 4.4, maximizing Pr_l can maximize the likelihood that G_P^* has at least one HP. Since Pr_l increases when d_{max} decreases and d_{min} increases, the maximum of Pr_l can be achieved when d_{max} reaches its minimum while d_{min} reaches its maximum. Note that any given task graph G_T, $\sum_{\forall v \in G_T} d_v = 2\ell$, where d_v is the degree of the vertex v, and ℓ is the number of edges of G_T. Then it must hold that

$$d_{min} \leq \frac{2\ell}{n} \leq d_{max} \qquad (3)$$

Thus, Pr_l reaches its maximum when $d_{min} = d_{max} = \frac{2\ell}{n}$.

We must note that although our preference smoothing step (Section V-B) guarantees that there is an HP, the construction of tasks of high HP-likelihood is still necessary, as it minimizes the number of edges whose pairwise preferences have to be smoothed.

B. HITs Generation

In this section, we discuss how to generate the pairwise comparison HITs that are budget-constrained, fair, and of high HP-likelihood. Following Theorem 4.1 and Equation 3, we generate the HITs by the following approach. Let PS be the set of vertices that have been selected, initialized as PS=$\{\}$. Let G_T be a graph that is initialized with vertices only. Then we randomly construct an acyclic path P that connects all vertices of G_T. Apparently P is an HP. Next, for each vertex $v \in G_T$, we randomly pick $(\frac{2\ell}{n} - d_v)$ vertices $V \subseteq G_T - PS$, where d_v is the current degree of v, which is updated dynamically during the task generation. In this process, we insert v into PS once d_v reaches $\frac{2\ell}{n}$, as the desired degree of node v is obtained. The pseudo code of the task generation algorithm is shown in Algorithm 1. The complexity of the graph construction procedure is $O(n)$. The output task graph G_T contains ℓ pairwise comparisons, which satisfies the budget-constrained requirement. Each node of G_T has the same degree, while the degree maximizes the HP-likelihood. Thus G_T is fair and high HP-likely.

Algorithm 1 HITs generation

```
Require: The objects $\mathcal{O}$ to be ranked
Ensure: Task assignment graph $G_T$
1: for each object $O_i \in \mathcal{O}$ do
2:     Construct a vertex $v_i \in G_T$
3: end for
4: Randomly draw a path $P$ that connects all vertices in $G_T$; Obviously $P$ is a HP.;
5: for each vertex $v \in G_T$ do
6:     Pick $\frac{2\ell}{n} - d_v$ vertices in $G_T$ ($d_v$: current degree of $v$);
7:     Add edge between $v$ with these picked vertices;
8: end for
```

V. Result Inference

Based on the collected pairwise preferences from the workers, we compute the full ranking by the *result inference* step. The result inference step consists of four steps: (1) Infer the *direct* pairwise preference based on the comparison results by the crowd; (2) *Smooth* the inferred direct pairwise preferences of those edges of weight 1, so that there exist at least an HP (i.e., a full ranking); (3) Compute the *indirect* pairwise preferences that are inferred by transitivity, and aggregate both direct and indirect pairwise preferences; and (4) Compute the preference of full rankings by aggregating the pairwise preferences, and pick the full ranking of the highest preference as the final output. Next, we discuss the details of each step.

A. Step 1: Truth Discovery of Direct Pairwise Comparisons

It is challenging to infer the *truth* of the preference for each assigned pairwise ranking task from the pairwise preferences that are possibly noisy and conflicting. As the workers may produce the comparison results of different quality, the accuracy of the aggregated preferences can be improved by capturing the quality (i.e., reliability) of the workers. Intuitively, the pairwise preferences by the workers who are more reliable should be considered as the truth. The challenge is that workers' reliability is usually unknown a priori in practice and has to be inferred from the pairwise comparison results.

We observe that the workers' quality and the inferred truth are correlated: the workers who provide true pairwise preferences more often will be assigned higher quality (i.e., reliability degrees), and the pairwise preference that is supported by reliable workers will be regarded as the truth. Based on this observation, we estimate the workers' quality from their preference, and perform weighted aggregation of the workers' preferences based on their estimated quality. In particular: (1) for any given pairwise comparison task T, if the workers quality is high, then her preference is likely to be the truth for T; and (2) for any given worker W, if W often answers tasks correctly, then W's quality is high. Based on these intuitions, we develop an iterative approach, which updates the pairwise preference and the workers' quality until it reaches convergence. Next, we discuss the details of the iterative approach.

Formally, given a set of workers $\mathcal{W}$, and a set of pairwise comparison tasks $\mathcal{T}$, for any worker $W_k \in \mathcal{W}$, Let q_k be the quality of W_k, and $\mathcal{T}_k \in \mathcal{T}$ be the set of tasks assigned to the worker W_k. For any task $T(O_i, O_j) \in \mathcal{T}$, let $\overline{W_{ij}}$ be the set of workers who perform it, and $\hat{x_{ij}}$ be the true preference of T (i.e., the probability that $O_i \prec O_j$). Our goal is to infer: (1) q_k, for each worker $W_k \in \mathcal{W}$; and (2) $\hat{x_{ij}}$, for each task $T(O_i, O_j) \in \mathcal{T}$. We require that both $\hat{x_{ij}}$ and q_k are in the range [0, 1]. We call the inferred $\hat{x_{ij}}$ the *direct* pairwise preference, which will be used as the weight preference w_{ij} on G_P. We follow the same assumption in [12], [13] that the workers' error follows the normal distribution $\epsilon_k \sim \mathcal{N}(0, \sigma_k^2)$, where σ_k is the standard error deviation of worker W_k. Intuitively, the lower σ_k is, the higher the worker's quality q_k will be. We assume that the workers' quality stays stable across all the tasks.

First, we discuss how to model each worker's pairwise preference. Given a task $T(O_i, O_j)$, we denote a worker W_k's preference input x_{ij}^k as:

$$x_{ij}^k = \begin{cases} 1, & O_i \prec O_j; \\ 0, & O_i \succ O_j. \end{cases}$$

Intuitively, each single pairwise preference is valued 0 or 1. Next, we discuss how to aggregate these pairwise preferences by the truth discovery approach.

Based on the worker's preference result, we estimate the true preference $\hat{x_{ij}}$ on the task $T(O_i, O_j)$. A commonly used strategy is to compute the weighted average of the workers' preferences, namely,

$$\hat{x_{ij}} = \frac{\sum_{W_k \in \overline{W_{ij}}} x_{ij}^k \times q_k}{\sum_{W_k \in \overline{W_{ij}}} q_k}. \quad (4)$$

Based on $\hat{x_{ij}}$, we update the workers' quality. Intuitively, if a worker provides reliable information more often, he has a larger quality. In this paper, we adopt the weight estimation introduced in [12]. Specifically, a source weight is inversely proportional to its total difference from the estimated truth, namely,

$$q_k \propto \frac{\mathcal{X}^2(\alpha/2, |\mathcal{T}_k|)}{\sum_{t_j \in \mathcal{T}_k} (x_{ij}^k - \hat{x_{ij}})^2}, \quad (5)$$

where α is a pre-defined value of the confidence interval, and $\mathcal{X}^2(\alpha/2, |\mathcal{T}_k|)$ is the $\frac{\alpha}{2}$-th percentile of a $\mathcal{X}^2$-distribution with degree of $|\mathcal{T}_k|$.

Using Equation (4) and (5), we propose an iterative algorithm to update $\{q_i\}$ and $\{\hat{x_{ij}}\}$. Initially, we assign equal quality to all the workers. Based on the collected workers' pairwise preferences, we estimate the true pairwise preferences. Then we update the workers' quality based on the inferred truth. We keep updating $\{q_k\}$ and $\{\hat{x_{ij}}\}$ until convergence. Since the truth inference takes an iterative approach, it could be expensive to run until convergence. To alleviate this issue, we allow the convergence condition to be either reaching a certain number of iterations which is user-specified, or the small change of $\{\hat{x_{ij}}\}$ and $\{q_k\}$ (i.e., the difference between $\{\hat{x_{ij}}\}$ and $\{q_k\}$ in two consecutive iterations is smaller than a user-specified threshold).

The iterative algorithm outputs: (1) the estimated direct pair preferences $\{\hat{x_{ij}}\}$, which will be used for the following Step 2 - 4; and (2) the estimated workers' quality, which will be utilized for the next Step 2. The complexity is $O(ym\ell + n^2)$, where y is the number of iterations to reach convergence, m is the number of workers, ℓ is the number of unique pairwise comparison tasks, and n is the number of objects. Our experiment results show that the algorithm achieves convergence within 10 iterations for most of the testing cases. More details can be found in Section VI.

B. Step 2: Preference Smoothing

Our task assignment scheme cannot guarantee the existence of HP in G_P (i.e., full rankings). In this step, we discuss how to adjust the pairwise preferences inferred by Step 1 to ensure a full ranking. Our key idea is to do *smoothing* over the pairwise preferences of those edges that disable the HP traversal.

In principle, the smoothing is applied on the weights of edges that involve at least one in-/out-nodes. Following Theorem 4.3, the existence of in-/out- nodes in G_P is the main reason of HP failure. An important property of the in-/out- nodes is that the edges connected with these nodes are always of weight preference 1. We call those edges of weight 1 as *1-edges*. Note that there is no edge of weight 0 in G_P. Each 1-edge (v_i, v_j) indicates the fact that all the workers agree completely with the pairwise preference $O_i \prec O_j$ *in this round*. There might be votings of $O_j \prec O_i$ if there is more budget for additional rounds of crowdsourcing. Indeed, the *unknown* preference $O_j \prec O_i$ is the main reason of HP failure.

To ensure the existence of HP in G_P (and its transitive closure), our solution is to *smooth* the preference of each 1-edge (v_i, v_j) by estimating the *unknown* preference on the edge (v_j, v_i), such that $w_{ij} < 1$, $w_{ji} > 0$, while $w_{ij} + w_{ji} = 1$. Let $\widetilde{G_P}$ be the preference graph after smoothing. Figure 1 (c) shows an example of $\widetilde{G_P}$, the preference graph G_P in Figure 1 (a) after smoothing. An important property of $\widetilde{G_P}$ is that it is always a strongly connected graph. This is the key to guarantee the existence of HP in the final inference result (more details see Theorem 5.1). We must note that we apply smoothing *only* on the 1-edges, aiming to minimize the amounts of errors introduced by estimation.

Our idea of smoothing is to predict the unseen preferences by incorporating the workers' estimated quality with their pairwise preferences. Intuitively, as we assumed, given a worker W_k with quality q_k, the error of his pairwise preferences follow the distribution of $\mathcal{N}(0, \sigma_k^2)$. Given an 1-edge $(O_i, O_j) \in G_P$, let its corresponding pairwise comparison task be T. Let $\overline{W}$ be the set of workers that undertake T. We apply *smoothing* on w_{ij} and w_{ji} by the following:

$$w_{ij} = w_{ij} - \frac{\sum_{\forall W_k \in \overline{W}} err_k}{|\overline{W}|},$$

and

$$w_{ji} = w_{ji} + \frac{\sum_{\forall W_k \in \overline{W}} err_k}{|\overline{W}|},$$

where err_k is the error of the worker W_k that follows the distribution of $\mathcal{N}(0, \sigma_k^2)$. As the higher quality q_k is, the smaller error probability err_k will be. We value $\sigma_k = -log(q_k)$.

C. Step 3: Computation of Indirect Pairwise Preference

Due to the transitivity property, a *hidden* edge (v_i, v_j) is added to the smoothed preference graph $\widetilde{G_P}$ if there exists a path $P(v_i, \ldots, v_j)$ exists in $\widetilde{G_P}$. We use G_P^* to denote the transitive closure (i.e., the reachability relation) of the preference graph G_P. In this section, we discuss how to compute the *indirect* preference probability of these hidden edges in G_P^* that are constructed by the transitivity.

Given a path $P(v_i, \ldots, v_j)$ whose length is larger than 1, the weight of the edge (v_i, v_j) inferred from the path P is calculated as $w_{ij}^P = \prod_{\forall (v_x, v_y) \in P} w_{xy}$. If there are $k > 1$ paths $P_1, \ldots, P_x$ between the vertices v_i and v_j (excluding the direct edge $(v_i, v_j) \in G_P$), we assume that each path has equal importance. Thus the indirect weight w_{ij}, denoted as w_{ij}^*, is aggregated as $w_{ij}^* = \sum_{x=1}^{k} w_{ij}^{P_x}$.

For any edge $(v_i, v_j) \in G_P^*$, let w_{ij} be its direct preference and w_{ij}^* be its (possibly smoothed) indirect preference in $\widetilde{G_P}$, the final preference is computed as

$$\check{w_{ij}} = \alpha w_{ij} + (1 - \alpha) w_{ij}^*,$$

where α is a user-specified value that defines the weights on the (in)direct preferences. $\check{w_{ij}}$ will be used as the edge weight for Step 4 to find the best ranking. For simplicity, for later discussions, $\check{w_{ij}}$ and w_{ij} are exchangeable.

Finally, to satisfy the probability constraint [14] (i.e., $w_{ij} + w_{ji} = 1$), for any edge (v_i, v_j), the weight w_{ij} and w_{ji} are normalized as follows:

$$w_{ij} = \frac{w_{ij}}{w_{ij} + w_{ji}}, \; w_{ji} = \frac{w_{ji}}{w_{ij} + w_{ji}}.$$

We have the following theorem that shows there always exists an HP in G_P^*.

Theorem 5.1: Given a preference graph G_P, let G_P^* be its transitive closure constructed by Step 1 - 3. Then there always exists an HP in G_P^*.

The correctness of Theorem 5.1 is straightforward. Recall that by Step 2 $\widetilde{G_P}$ is a strongly connected graph. After Step 3, G_P^* is guaranteed to be a complete graph. It is well known that a complete graph with more than two vertices is always Hamiltonian. Therefore, our algorithm always guarantees a full ranking to be aggregated from the pairwise preferences.

D. Step 4: Find Best Ranking

Given the pairwise preferences computed by Step 1 - 3, next, we compute the preference probability of all HPs. Formally, given a Hamiltonian path $P(v_1, \ldots, v_n)$, the *preference probability* $Pr[P]$ of P is measured as $Pr[P] = \prod_{\forall (v_i, v_j) \in P} w_{ij}$. We pick the HP(s) of the highest preference probability as the output full ranking. We design two algorithms, namely the threshold-based path search (TAPS) algorithm that returns the *exact* solution, and a simulated annealing based path search (SAPS) algorithm that returns the *heuristic* solution.

1) Exact Solution: The brute-force approach is to compute the preference probability of every HP in G_P^*, and pick the HP of the highest probability. However, the brute-force approach can be prohibitively costly. Thus we design an efficient algorithm, named threshold-based path search (TAPS) algorithm, that can find the HP of top-1 preference probability with early termination. TAPS is adapted from the state-of-the-art *threshold algorithm* (TA) [15]. Similar to TA, TAPS uses a threshold to control the stop rule; the algorithm halts as soon as the top k answers (in our case $k = 1$) have been accessed. Next, we explain the TAPS algorithm.

TAPS algorithm first constructs $n-1$ lists, with the i-th list corresponding to the i-th edge in the HP. For each list, its entries consisting of a tuple < *pathID*, *edgeWeight*>, where *pathID* consists of ID of the path. Each list is sorted by the value of *edgeWeight* in descending order. TAPS executes the following two main steps on these lists. The output Y is initialized as an empty set, and the highest preference probability max is initialized as 0.

Step 1: Do sorted access in parallel to each of the $n-1$ sorted lists L_i, starting from the first entry. When a path P is seen under sorted access in some list, do random access to the other lists to find the weights of each edge of P in every list L_i. Compute the preference probability $Pr[P]$ of P. If $Pr[P] > max$, then update the output $Y = \{P\}$, and set $max = Pr[P]$. Otherwise if $Pr[P] = max$, then add P to Y. This ensures to include all tie paths (i.e., the paths of the same preference probability) in the output.

Step 2: For each list L_i, let w_i be the edge weight of the last path seen under the sorted access. Define the threshold value $\theta = \Pi_{i=1}^{n-1} w_i$. If $max \geq \theta$, then halt and output Y.

Given n objects, there are $n!$ rows and $2(n-1)$ columns in total in all lists. Thus the total space that TAPS algorithm needs is $n!(2n-1)$. The time complexity of TAPS is $O(n! \sqrt[n-1]{\frac{1}{n!}})$.

2) Heuristic solution: The complexity of TAPS is still high. Therefore, we design a heuristic solution named simulated annealing based path search (SAPS) algorithm. Recall that our objective is to find the HP of the maximum probabilistic preference. By making negative logarithm of the product of weights w_{ij}, the equivalent problem is to look for an HP P of minimum $\sum_{\forall(v_i,v_j)\in\mathbf{P}} \log w_{ij}^{-1}$.

We adapt the Simulated Annealing (SA) algorithm to heuristically searching a solution P^*. Simulated annealing (SA) [16] aims to approximate the global minimum solution by simulating the physical process whereby metal are slowly cooled down. The key idea is to find a global optimal solution with high probability by retaining worse solutions with a probability based on Boltzmann probability distribution $P(E=k) \propto exp\left(-\frac{k}{T}\right)$, where T is temperature, and E is energy.

Now we are ready to explain our simulated annealing based path search (SAPS) algorithm. The pseudo code is shown in Algorithm 2. Basically SAPS generates the next path in an iterative way (Line 4 - 14). The next path can be generated by three operations: (1) Rotate(*P, first, middle, last*) (Line 6): rotates the nodes in the index range [*first, last*], in such a way that the node labeled by *middle* becomes the new first element of the new range and *middle - 1* becomes the last element; (2) Reverse(*P, first, last*) (Line 8): selects two random nodes and reverse the nodes between them; and (3) RandomSwap(*P, first, last*) (Line 10): selects a random pair of nodes and swap them. Then the algorithm calls updateHP() function (Algorithm 3) to decide whether the next path should be selected. In particular, if the probability of the next path is better than the current path, the next path is picked with probability 1 (Line 3 - 4). Otherwise, the next path is selected with a probability that follows Boltzmann probability distribution (Line 5 - 7). The complexity of SAPS is $O(Nn^2 + n^3 + n^2 log(n))$, where n is the number of vertices in G_P, and N is the number of iterations (a user-specified parameter). When n is large enough, the complexity of SAPS is $O(n^3)$.

Algorithm 2 SAPS Algorithm

```
Require: Transitive closure $G_P^*$ of preference graph $G_P$, iteration # N, temperature T, cooling rate c;
Ensure: An HP $P \in G_P^*$ that is of the maximum probabilistic preference.
1:  Initialize iteration index $i = 0$. Randomly pick $P_i \in G_P^*$. Compute $d_i = \sum_{\forall(u,v)\in\mathbf{P_i}} \log w_{uv}^{-1}$. Initialize HP = $P_i$.
2:  for all $v \in V$ in $G_P^*$ do
3:      Generate an initial path starting from vertex $v$, by selecting the nearest neighbors, or by ranking the nodes based on the difference of their out-/in- edge weights in $G_P^*$.
4:      while $i < N$ do
5:          // three heuristic path permutation methods;
6:          $\mathbf{P_{i+1}} \longleftarrow$ Rotate($\mathbf{P_i}, \mathbf{first}, \mathbf{middle}, \mathbf{last}$);
7:          HP $\longleftarrow$ updateHP($P_i$, $P_{i+1}$) (Algorithm 3);
8:          $\mathbf{P_{i+1}} \longleftarrow$ Reverse($\mathbf{P_i}, \mathbf{first}, \mathbf{last}$);
9:          HP $\longleftarrow$ updateHP($P_i$, $P_{i+1}$) (Algorithm 3);
10:         $\mathbf{P_{i+1}} \longleftarrow$ RandomSwap($\mathbf{P_i}, \mathbf{first}, \mathbf{last}$);
11:         HP $\longleftarrow$ updateHP($P_i$, $P_{i+1}$) (Algorithm 3);
12:         $T \longleftarrow T \cdot c$;
13:         $i \longleftarrow i + 1$;
14:     end while
15: end for
16: return HP
```

Algorithm 3 HP updating function: updateHP(P_i, P_{i+1})

```
Require: Two paths $P_i$ and $P_{i+1}$;
Ensure: An HP of the maximum probabilistic preference so far.
1: Given $P_i$, compute $d_i = \sum_{\forall(u,v)\in\mathbf{P_i}} \log w_{uv}^{-1}$.
2: Given $P_{i+1}$, compute $d_{i+1} = \sum_{\forall(u,v)\in\mathbf{P_{i+1}}} \log w_{uv}^{-1}$.
3: if $d_{i+1} < d_i$ then
4:     $P_i \longleftarrow P_{i+1}$ // accept a better permutation with prob. 1;
5: else
6:     // accept a worst permutation with a prob. $p_i$;
7:     $P_i \longleftarrow P_{i+1}$, with probability $p_i = exp[\frac{-(d_{i+1}-d_i)}{T}]$;
8: end if
9: return $HP \longleftarrow P_i$;
```

VI. Experiments

To evaluate the performance of our approach, we perform an extensive set of experiments by using both simulated and real crowdsourcing datasets. In this section, we explain the details of our empirical study.

A. Setup

We execute the experiments on a testbed with 2.0 GHz CPU and 64 GB RAM, running Windows OS. We implement all the algorithms in C++.

1) Parameters: We change the settings of the following parameters: (1) *selection ratio* r: the portion of pairwise comparisons that can be selected for crowdsourcing with regard to the given budget. When $r = 1$ it is equivalent to the all-pair ranking setting; (2) w: number of workers per HIT; and (3) n: number of objects for ranking.

2) Baselines: We generate the baselines for both task assignment and result inference steps. The baselines serve different purposes for performance measurement.

Task Assignment. We generate the *all-pair* comparison tasks that include all $\binom{n}{2}$ pairwise comparisons of n objects.

In our experiments, the all-pair ranking baseline corresponds to the setting where the selection ratio $r = 1$.

Result Inference. From the existing work, we pick three categories that are most relevant to our problem, including: (1) rank aggregation, (2) crowdsourced ranking, and (3) truth discovery (more details of related work are in Sec. VII). For each category, we pick one state-of-the-art algorithm as the baseline:

Rank aggregation category: we pick the *RepeatChoice* (RC) algorithm [17] that tries to minimize the sum of distances between the output and the individual rankings. It uses Kendall-tau distance as the distance measurement.

Crowdsourced ranking category: we pick the QuickSort (QS) algorithm [18] that models the ranking preferences by the Condorcet graph. Unlike our graph model, its preference is scored by majority voting.

Truth discovery category: we pick the *CrowdBT* algorithm [7] that takes machine learning approach to combine the workers' quality with the Bradley-Terry Model [19] to estimate the latent score. We must note that unlike our work, *CrowdBT* is an algorithm for the *interactive* setting.

We will compare our ranking aggregation with these three baseline approaches with regard to the ranking accuracy and time performance.

3) Crowdsourcing Setting: We use Amazon Mechanical Turk[1] (AMT) for the crowdsourcing setting. We prepared a set of image ranking HITs that include a number of celebrity photos picked from the Public Figures Face Database [2]. The AMT workers are asked to rank the photos based on how much the celebrity smiled in the photos. To test the robustness of our approach, we only pick those photos that have conflicting opinions. To do this, first, we use an image ranking algorithm [20] to rank all $1,800$ images in the dataset. Then we pick a number of images whose rankings are close - the ranking difference of any two picked images that are next to each other in the ranking never exceed 46. We prepared the *10-image* setting and the *20-image* setting that includes 10 images and 20 images respectively. We must note that the ranking produced by the algorithm cannot be considered as the ground truth; it is well-known that the humans are better than the machines for image processing.

For the HIT configuration, we vary the number of workers per HIT (w=100, 125, 150, 200) and the crowdsourcing budget to study their impact on the ranking result. We pay \$0.025 for each pairwise comparison. Different budgets lead to different selection ratios. In our setting, we have the selection ratio r= 0.25, 0.5, 0.75, and 1.

4) Simulation Setting: We construct a number of preference datasets at large scale to simulate the workers of different quality.

Ground Truth. We design the following algorithm to simulate the preferences. Given n nodes, we randomly generate a permutation π, and consider π as the ground truth of their ranking. For any two nodes v_i and v_j, we use r_i and r_j to denote their positions in π.

[1]https://www.mturk.com/.

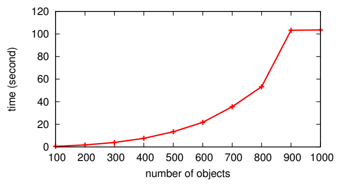
[2]http://www.cs.columbia.edu/CAVE/databases/pubfig/

Workers' Quality. As in the previous work [21], we assume that the worker W_k's error follows the normal distribution $\epsilon_k \sim \mathcal{N}(0, \sigma_k^2)$. Intuitively, the smaller σ_k is, the higher quality does the worker have. In our experiments, we use two different distribution settings of σ_k.

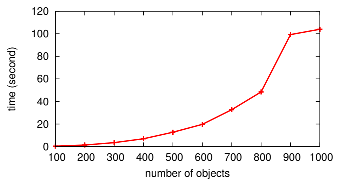
- **Gaussian distribution**: $\sigma_k \sim \mathcal{N}(0, \sigma_s^2)$, i.e., the standard deviation of each worker's error follows a normal distribution. We set $\sigma_s = 0.01, 0.1, 1$ to simulate the scenario where the workers have high, medium and low quality.
- **Uniform distribution**: σ_k follows a uniform distribution. We set the range of the uniform distribution to be $[0, 0.2]$, $[0.1, 0.3]$, and $[0.2, 0.4]$ to simulate the workers who have high, medium and low quality respectively.

Workers' Preference We generate the directed edges in the preference graph to simulate a worker's preference. For any task (O_i, O_j) assigned to worker W_k, the worker has ϵ_k probability to make the wrong voting, where $\epsilon_k \sim \mathcal{N}(0, \sigma_k^2)$. Thus, we add an edge $v_i \rightarrow v_j$ with probability $1 - \epsilon_k$, and the edge $v_j \rightarrow v_i$ with probability ϵ_k.

5) Accuracy Evaluation Metrics: We use the well-known Kendall Tau distance metrics [22] to evaluate the accuracy of the aggregated ranking. For the simulation setting, we assume that the ground truth is available, and we measure the Kendall Tau distance between the ground truth and the aggregated ranking result by our algorithm. For the crowdsourcing setting (image ranking), we do not have the ground truth. Thus we measure the Kendall Tau distance between the ranking result of $TAPS$ and *SAPS*. For both settings, we report $1 - d$ as the accuracy, where d is the Kendall Tau distance. Intuitively, the smaller d is, the higher is the accuracy.



(a) Workers' quality (Gaussian distribution)



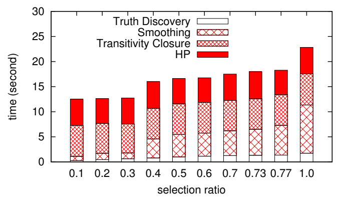
(b) Workers' quality (Uniform distribution)

Fig. 3: Result inference time w.r.t. number of objects (simulated setting: $r = 0.5$, medium worker quality)

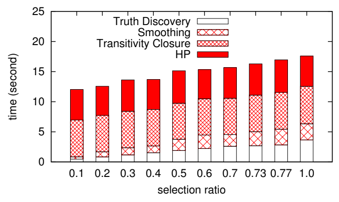
B. Time Performance

Number of Objects. We measure the time performance of the result inference algorithm *SAPS* with regards to various number of objects as well as different selection ratios (i.e., different budgets). In Figure 3, we evaluate the scalability of *SAPS* by varying the number of objects from 100 to 1000. We witness that *SAPS* is fast. Even when the number of objects reaches $1,000$, *SAPS* can finish in 2 minutes. This demonstrates that *SAPS* can support efficient rank aggregation on large number of objects. We also observe that different worker quality distributions make little impact on the time performance of *SAPS*. This is due to the fact that the time complexity of *SAPS* is independent from the weights on the edges.

Budgets. To simulate the different budgets for crowdsourcing, we change the selection ratio r from 0.1 to 1. Intuitively, a bigger budget leads to a larger r value. When $r = 1$, it is equivalent to the *all-pair comparison* baseline. Our results are reported in Figure 4. Overall, *SAPS* is very efficient. It only



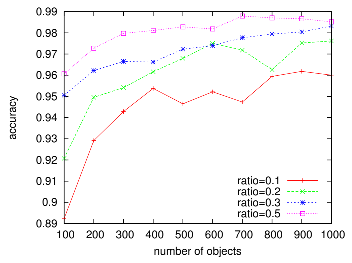
(a) Workers' quality (Gaussian distribution)



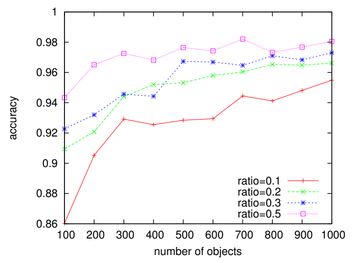
(b) Workers' quality (Uniform distribution)

Fig. 4: Result inference time w.r.t. budgets (simulated setting: $n = 1000$, medium worker quality)

takes around 15 seconds to rank 1000 objects, even for high selection ratio. Furthermore, the time performance of *SAPS* rises slightly with the increase of the selection ratio for both worker quality distributions, since there are more pairwise preferences to be aggregated. We further measure the time consumed by each step of the result inference algorithm. The first observation is that Step 4 (i.e. find the best ranking) takes more time than the other three steps. Second, for the worker quality of Gaussian distribution, Step 1 is faster than Step 2, while for the quality of Uniform distribution, Step 2 is faster than Step 1. We measure the number of 1-edges for both distributions. It turned out that for Gaussian distribution, the number of 1-edges is much more than that for the uniform distribution. This is not surprising as it is more likely to have 1-edges for the worker quality of Gaussian distribution, as the workers of high- (low-, resp.) quality intend to vote for the same correct (wrong. resp.) preferences.



(a) Workers' quality (Gaussian distribution)



(b) Workers' quality (Uniform distribution)

Fig. 5: Ranking accuracy w.r.t. number of objects and budgets (simulated setting, medium worker quality)

C. Ranking Accuracy

We measure the accuracy of our result inference method with regards to various parameters, including the number of objects, the budget (i.e., selection ratio), and workers' quality.

Number of Objects. We report the impact of the number of objects n on ranking accuracy in Figure 5. Overall the ranking accuracy is high (between [0.86, 0.99]), even for small selection ratio as 0.1. Furthermore, the ranking accuracy improves with the growth of the number of objects n. This is because when n increases, more pairwise preferences can be inferred by the transitive property, and thus lead to more accurate ranking result. An interesting observation is that the accuracy for the worker quality of Gaussian distribution is higher than that of uniform distribution, as there are more high-quality workers for the worker quality of Gaussian distribution.

Crowdsourcing Budgets. We vary the selection ratio r to simulate various budgets. Figure 5 reports the results. Intuitively, with the growth of r, the ranking accuracy gets better. This is not surprising as more pairwise comparisons leads to more accurate ranking. However, we must note that even when r is as small as 0.1 (i.e., only a tenth of pairwise comparisons are picked), the ranking accuracy can achieve at least 0.86 (for 100 objects and uniform quality distribution). When the selection ratio $r = 0.3$, the accuracy can reach 0.98! This demonstrates that our task assignment approach works well for budget-conscious crowdsourcing setting.

D. AMT Crowdsourcing Setting

Ranking Accuracy. We compare the accuracy of *SAPS* compared with the exact solution *TAPS*. We observe that for most cases, *SAPS* generates the same ranking result as *TAPS*. We omit the results due to the limited space.

TABLE I: Comparison with Baselines

# of objects	100		200		300	
	Accu.	Time(s)	Accu.	Time(s)	Accu.	Time(s)
SAPS	89.2%	0.363	92.9%	1.464	94.2%	3.882
RC	2.00%	0.038	1.70%	0.090	1.50%	0.179
QS	19.6%	0.275	23.1%	0.654	19.6%	1.159
CrowdBT	93.8%	3529	92.4%	14938	89.1%	26012

(a) Workers' quality (Gaussian distribution)

# of objects	100		200		300	
	Accu.	Time(s)	Accu.	Time(s)	Accu.	Time(s)
SAPS	86.0%	0.330	90.5%	1.235	92.9%	3.185
RC	1.83%	0.035	1.59%	0.089	1.46%	0.178
QS	18.8%	0.274	22.4%	0.661	19.5%	1.114
CrowdBT	92.9%	3490	93.0%	14100	83.0%	28425

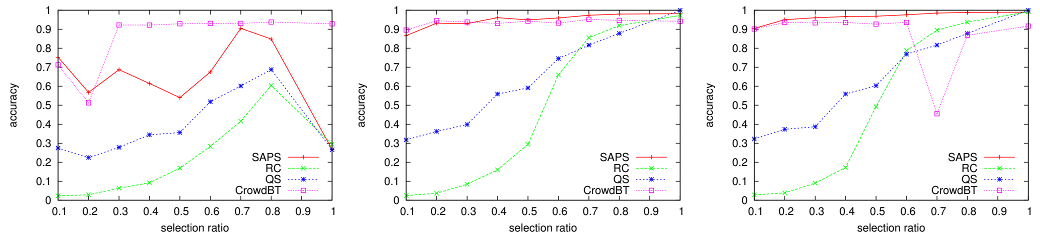
(b) Workers' quality (Uniform distribution)

E. Comparison with Baselines

Time Performance. We compare *SAPS* with the three baseline approaches. It turned out that *RC* is the fastest, while *QS* the second, and *SAPS* the third (3.882 seconds for 300 objects). *CrowdBT* is significantly slower than *SAPS* (26,012 seconds for 300 objects), as it performs in an *interactive* fashion. More details of the time performance can be found in Table I.

Accuracy. First, we compare the accuracy of the four approaches for various number of objects. We fix the selection ratio $r = 0.5$ and vary the number of objects from 100 to 300. Table I shows the comparison results. We observe that *SAPS* wins *RC* and *QS* for both worker quality distributions. The ranking accuracy of *RC* and *QS* never exceeds 20%, while the accuracy of *SAPS* is always higher than 86%. Furthermore, the accuracy of *CrowdBT* is comparable to *SAPS*. An interesting observation is that the accuracy of *SAPS* improves with the increase of the number of objects, while the accuracy of *CrowdBT* degrades. This is because *SAPS* benefits from the transitive property of ranking preferences; more objects leads to more preferences that can be inferred from the transitive closure. However, *CrowdBT* suffers from the impact of the number of objects on active learning. As stated in [7], for large number of objects, the active learning strategy may perform too much exploration but with small accuracy gain.

Second, we compare the accuracy of the four approaches with regards to various budgets (i.e. selection ratios) and workers' quality. We only report the results when the workers' quality follows the Gaussian distribution. The results from the uniform distribution is similar to the Gaussian distribution and thus we omit them due to the limited space. We have the following observations. First, in most cases, the accuracy



(a) Low worker quality (b) Medium worker quality (c) High worker quality

Worker quality: Gaussian distribution

Fig. 6: *SAPS* versus baselines w.r.t. workers' quality (simulated setting)

improves when the selection ratio grows. This makes sense as more pairwise preferences always lead to more accurate ranking result. Second, under various selection ratios, *SAPS* always produces the top-2 ranking accuracy. Compared with *RC* and *QS*, *SAPS* wins more for smaller r. For these cases, the accuracy of *RC* and *QS* (below 0.5) is no better than a random guess. In contrast, the ranking accuracy of *SAPS* is at least 0.88. Third, *CrowdBT* demonstrates its advantage when only a small fraction of comparison tasks can be crowdsourced. But it loses to *SAPS* when the budgets increases. Fourth, all the four approaches produce higher ranking accuracy when the worker quality improves. Furthermore, *SAPS* wins the other three baseline approaches in almost all testing cases when the workers have medium or high quality.

VII. Related Work

We classify the related work into three categories: (1) crowdsourcing ranking; (2) rank aggregation; and (3) truth discovery.

Crowdsourced Ranking. The problem of finding top-k ranking is highly related to ours. While this problem aims at finding the top-k answers, we aim to find the full ranking. Most of the existing crowdsourced top-k ranking approaches follow the same idea: estimate the score for each object from the crowd's pairwise preferences, and find the k objects of the highest estimated scores. There are two type of algorithms to estimate the object scores: heuristic-based algorithms and machine-learning methods [23]. The heuristic algorithms estimate the object scores by either the number of neighbors in a graph model (e.g., [5]), or the probability theory [24], or the number of winning counts (e.g., [25]). The machine-learning methods (e.g. [7], [8]) assume that each object has a latent score which follows a certain distribution. These algorithms use machine learning techniques to estimate the scores. We refer the readers to [23] for a wonderful survey on crowdsouced top-k ranking. Unlike these work, we do not associate a score with the objects. Instead, we compute the probabilistic preference on the pairwise comparisons, and aggregate the preference to generate the full ranking.

Rank Aggregation. The rank aggregation problem have gained much attention. The rank aggregation functions can be grouped into two categories: (1) ranking list aggregation; and (2) pairwise preference aggregation. The problem of rank list aggregation is the following: given m permutations of n elements (i.e., m full rankings), to find a permutation that minimizes the sum of *distances* between itself and each given permutation. Here, each permutation represents a ranking over n elements. A key of the rank aggregation problem is to define a measure of disagreement between the input rankings and the aggregated result. Correlation statistics such as Spearman's rank correlation coefficient [26] and Kendall's tau distance [22] are popularly used. It is known that the rank aggregation problem based on Kendall's tau distance is NP-hard [27]. A number of heuristic algorithms can compute the aggregation in O($nlogn$) [28] or $O(nlogn/loglogn)$ [29]. Algorithms for aggregation of partial rankings (i.e., rankings with ties) are also designed [17].

Pairwise preference aggregation methods organize their ranking inputs in a pairwise way. Possible approaches include modeling the ranking preferences as a Condorcet graph [18], a tournament [30], and a pairwise preference matrix [31]. The probabilistic approaches define the generative probability of pairwise preferences and optimize the likelihood function by a gradient based approach [32], [7].

Truth Discovery. Truth discovery [33], [12], [34] has caught much attention recently and many truth discovery methods have been proposed. The goal of truth discovery is to identify true information (i.e., truths) from the conflicting multi-source data. The advantage of truth discovery over the naive aggregation methods is that it can capture the variance in sources reliability degrees. Most truth discovery methods estimate source reliability automatically from the data, and integrate the source reliability into truth computation as source weight. Consequently, the more reliable sources contribute more in the final aggregation step. A large variety of truth discovery methods have been designed to jointly estimate truths and source reliability. We refer to [11] for a wonderful survey on this topic. In our work, we adapt the truth discovery method to estimate the true pairwise inference and the workers' quality.

VIII. Conclusion

In this paper, we study budget-conscious pairwise ranking aggregation problem in the non-interactive crowdsourcing setting. We design efficient algorithms that select a small number of pairwise comparisons by the crowd. We ensure that the selected pairs guarantee fair and robust ranking aggregation. We also design efficient result inference algorithms that construct the full ranking from the workers' pairwise preferences by taking the workers' quality into consideration. Experimental results demonstrate that our method can achieve high accuracy in full ranking generation under the non-interactive setting.

There are many interesting future directions. In this paper, we aim to bound the number of comparisons performed by the crowd under a given budget constraint. It is interesting to consider alternative objectives such as minimizing the number of comparisons to find the full ranking with acceptable accuracy. Another interesting research direction is to consider the same setting for top-k ranking.

References

[1] N. Prize, http://www.netflixprize.com/.

[2] D. Ghadiyaram and A. C. Bovik, "Crowdsourced study of subjective image quality," in *2014 48th Asilomar Conference on Signals, Systems and Computers*, 2014, pp. 84–88.

[3] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *Proceedings of the VLDB Endowment*, vol. 5, no. 1, pp. 13–24, 2011.

[4] L. Tran-Thanh, M. Venanzi, A. Rogers, and N. R. Jennings, "Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks," in *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, 2013, pp. 901–908.

[5] S. Guo, A. Parameswaran, and H. Garcia-Molina, "So who won?: dynamic max discovery with the crowd," in *Proceedings of the ACM International Conference on Management of Data*, 2012, pp. 385–396.

[6] S. B. Davidson, S. Khanna, T. Milo, and S. Roy, "Using the crowd for top-k and group-by queries," in *Proceedings of the International Conference on Database Theory*, 2013, pp. 225–236.

[7] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz, "Pairwise ranking aggregation in a crowdsourced setting," in *Proceedings of the International Conference on Web Search and Data Mining*, 2013, pp. 193–202.

[8] T. Pfeiffer, X. A. Gao, Y. Chen, A. Mao, and D. G. Rand, "Adaptive polling for information aggregation." in *Proceedings of AAAI Conference on Artificial Intelligence*, 2012.

[9] Q. Li, F. Ma, J. Gao, L. Su, and C. J. Quinn, "Crowdsourcing high quality labels with a tight budget," in *Proceedings of the ACM International Conference on Web Search and Data Mining*, 2016, pp. 237–246.

[10] G. Kim and E. P. Xing, "Time-sensitive web image ranking and retrieval via dynamic multi-task regression," in *Proceedings of the ACM International Conference on Web Search and Data Mining*, 2013, pp. 163–172.

[11] Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han, "A survey on truth discovery," *arXiv Preprint arXiv:1505.02463*, 2015.

[12] Q. Li, Y. Li, J. Gao, L. Su, B. Zhao, M. Demirbas, W. Fan, and J. Han, "A confidence-aware approach for truth discovery on long-tail data," *Proceedings of the VLDB Endowment*, vol. 8, no. 4, pp. 425–436, 2014.

[13] B. Zhao and J. Han, "A probabilistic model for estimating real-valued truth from conflicting sources," *Proceedings of International Workshop on Quality in Databases*, 2012.

[14] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: ranking and clustering," *Journal of the ACM (JACM)*, vol. 55, no. 5, p. 23, 2008.

[15] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 614–656, 2003.

[16] V. Černỳ, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, 1985.

[17] N. Ailon, "Aggregation of partial rankings, p-ratings and top-m lists," *Algorithmica*, vol. 57, no. 2, pp. 284–300, 2010.

[18] M. Montague and J. A. Aslam, "Condorcet fusion for improved retrieval," in *Proceedings of the International Conference on Information and Knowledge Management*, 2002, pp. 538–548.

[19] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: the method of paired comparisons," *Biometrika*, vol. 39, no. 3-4, pp. 324–345, 1952.

[20] D. Parikh and K. Grauman, "Relative attributes," in *International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 503–510.

[21] H. Xiao, J. Gao, Q. Li, F. Ma, L. Su, Y. Feng, and A. Zhang, "Towards confidence in the truth: A bootstrapping based truth discovery approach," in *International Conference on Knowledge Discovery and Data Mining*, 2016.

[22] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, pp. 81–93, 1938.

[23] X. Zhang, G. Li, and J. Feng, "Crowdsourced top-k algorithms: An experimental evaluation," *Proceedings of the VLDB Endowment*, vol. 9, no. 8, pp. 612–623, 2016.

[24] F. Wauthier, M. Jordan, and N. Jojic, "Efficient ranking from pairwise comparisons," in *Proceedings of the International Conference on Machine Learning*, 2013, pp. 109–117.

[25] B. Eriksson, "Learning to top-k search using pairwise comparisons," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2013, pp. 265–273.

[26] D. E. Critchlow, *Metric methods for analyzing partially ranked data*. Springer Science & Business Media, 2012, vol. 34.

[27] J. Bartholdi III, C. A. Tovey, and M. A. Trick, "Voting schemes for which it can be difficult to tell who won the election," *Social Choice and Welfare*, vol. 6, no. 2, pp. 157–165, 1989.

[28] W. R. Knight, "A computer method for calculating kendall's tau with ungrouped data," *Journal of the American Statistical Association*, vol. 61, no. 314, pp. 436–439, 1966.

[29] A. Andersson and O. Petersson, "Approximate indexed lists," *Journal of Algorithms*, vol. 29, no. 2, pp. 256–276, 1998.

[30] W. W. C. R. E. Schapire and Y. Singer, "Learning to order things," *Advances in Neural Information Processing Systems*, vol. 10, p. 451, 1998.

[31] D. F. Gleich and L.-h. Lim, "Rank aggregation via nuclear norm minimization," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2011, pp. 60–68.

[32] L. L. Thurstone, "The method of paired comparisons for social values." *The Journal of Abnormal and Social Psychology*, vol. 21, no. 4, p. 384, 1927.

[33] X. L. Dong, L. Berti-Equille, and D. Srivastava, "Integrating conflicting data: the role of source dependence," *Proceedings of the VLDB Endowment*, 2009.

[34] Y. Li, Q. Li, J. Gao, L. Su, B. Zhao, W. Fan, and J. Han, "On the discovery of evolving truth," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2015.