Holistic Crowd-Powered Sorting via AID

Optimizing for *Accuracies*, *Inconsistencies*, and *Difficulties*

Shreya Rajpal
University of Illinois, Urbana Champaign
srajpal2@illinois.edu

Aditya Parameswaran
University of Illinois, Urbana Champaign
adityagp@illinois.edu

ABSTRACT

We revisit the fundamental problem of sorting objects using crowdsourced pairwise comparisons. Prior work either treats these comparisons as independent tasks—in which case the resulting judgments may end up being *inconsistent*, or fails to capture the accuracies of workers, or difficulties of the pairwise comparisons—in which case the resulting judgments may end up being consistent with each other, but ultimately more *inaccurate*. We adopt a *holistic* approach that constructs a graph across the set of objects respecting consistency constraints. Our key contribution is a novel method of encoding difficulty of comparisons in the form of constraints on edges. We couple that with an iterative E-M-style procedure to uncover information about latent variables and constraints, along with the graph structure. We show that our approach predicts edge directions as well as difficulty values more accurately than baselines on both real and simulated data, across graphs of various sizes.

CCS CONCEPTS

• **Information systems** → *Retrieval models and ranking*; *Information extraction*;

KEYWORDS

graphs; crowdsourcing; constrained optimization; pairwise comparisons

ACM Reference Format:
Shreya Rajpal and Aditya Parameswaran. 2018. Holistic Crowd-Powered Sorting via AID: Optimizing for *Accuracies*, *Inconsistencies*, and *Difficulties*. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3269206.3269279

1 INTRODUCTION

Crowdsourcing is an effective means to collect fine-grained training data to build accurate machine learning models. One useful type of training data often collected is a *sort order* or a *partial sort* of a set of objects, assembled by having crowd workers compare pairs of objects at a time. Sorting is useful in many settings, including search engine result evaluation and sentiment analysis in text processing [2], and evaluating material properties (such as reflectance and shading) [17] and constructing depth maps in computer vision [3].

Unfortunately, there are three aspects that make crowd-powered sorting difficult:

Inconsistencies. Crowd workers may provide answers that are inconsistent with each other, and that are internally inconsistent with their own answers, making it hard to reconcile these answers. For example, a given worker may say that $a > b$, $b > c$, and $c > a$, where $>$ indicates a preference for the former object over the latter.

Abilities. Crowd workers have different degrees of ability or accuracies —and these may not be known up-front.

Difficulties. Some pairwise comparisons may be easier than others: for example comparing an object that is close to the "top" with an object that is close to the "bottom" is an easy comparison, with a smaller chance of mistakes, while objects that are "close" to each other in the sort order may be harder to compare with each other.

Prior work in crowdsourcing has failed to address all three aspects.

Expectation-Maximization (EM) style approaches without Resolution of Inconsistencies. One approach to construct a sort order is to treat each pairwise comparison as an independent binary question, and then applying an iterative procedure [10, 13, 14, 16] to learn the difficulties of questions, and the accuracies of crowd workers. Unfortunately, while this approach identifies a consensus answer for each pairwise comparison, it does not deal with the aspect of ensuring consistencies across answers, giving rise to a sort order.

Sorting without Inference of Accuracies and Difficulties. There has been a lot of work on crowd-powered sorting, e.g., [1, 4, 5, 7, 9, 11, 12, 15]. Most papers in this area assume that accuracies are fixed across crowd workers, and that the difficulties are fixed across comparisons. Even with restrictive assumptions, sorting ends up being NP-Hard, via *feedback arc-set* [7].

Our Approach. Instead, we adopt a holistic approach for crowd-powered sorting that identifies partial sort orders using pairwise comparisons, by modeling it as a graph problem with votes on edge directions—then, our problem involves identifying the "most likely" directed acyclic graph (DAG) using these votes.

We explicitly model the notion of difficulty of edges, as unknown edge properties, and use this notion to obtain the most likely partial sort order that is internally consistent. In particular, if the underlying "true" sort order is $a \rightarrow b \rightarrow c$, our constraints will indicate that the difficulty of comparing a and c can be only as hard as comparing a and b, or b and c. The eventual DAG that we identify will prioritize agreement with crowd workers for internal consistency on easier edges, while possibly disagreeing with crowd workers

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3269279

on more difficult edges. These constraints allow us to encode additional information about the problem, leading to a more accurate solution.

We find the most likely DAG using a modified version of the EM algorithm. In the expectation step, we find the maximum likelihood edge direction for each edge in the graph, using edge difficulty information. We then use a greedy algorithm to find the maximum likelihood DAG from the set of determined edges, ensuring that the DAG is internally consistent at each iteration. In the maximization step, we maximize the likelihood of the edge difficulties using constrained convex optimization, using the constraints on edge difficulties imposed by the graph structure. We also demonstrate how our solution can be extended to model crowd abilities.

Our Contributions.

(1) We model the problem as one of finding a maximum-likelihood partial order, corresponding to a maximum-likelihood DAG.
(2) We propose an EM-based solution that can iteratively identify this maximum-likelihood DAG over undirected graphs.
(3) We propose a greedy algorithm that ensures that the predicted DAGs are internally consistent.
(4) We develop a novel method to encode information about graph structure using constraints on edge difficulties, leading to increased accuracy.
(5) We demonstrate that our approach results in up to 7.91% improvement in the accuracy of sorting, as well as up to a 45.37% improvement in the entropy and 4.45% improvement in the accuracy of predicted difficulty values of comparisons.

The rest of this paper is organized as follows: we describe our problem statement in Section 2, our modified EM solution in Section 3, and our experiments comparing against two baselines on simulated and real data in Section 4.

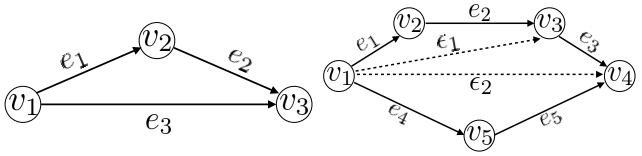
2 PROBLEM SETUP

Given crowdsourced pairwise comparisons between a set of items, our goal is to find a partial order that is consistent with most crowd responses. Typically, the more challenging pairwise comparisons result in more disagreement amongst crowd workers, as well as lower confidence in the determined order. Ideally, our solution should agree with the crowd on easy comparisons, while possibly disagreeing on difficult comparisons so as to determine an internally consistent partial order.

Graph Model. We model this problem as a graph, wherein we map items to vertices and pairwise comparisons between items to directed edges in the graph. Edges are directed from the 'less than' vertex to the 'greater than' vertex, and there can be multiple edges between any given pair of vertices. Inconsistencies in the crowd often result in cycles in the graph, therefore our new goal is to find a Directed Acyclic Graph (DAG) over the graph, such that the DAG is maximally consistent with the crowd.

In general, DAGs are desirable because cycles suggest inconsistencies in the ordering. A cycle between vertices v_1, v_2, v_3 means that $v_1 < v_2$, $v_2 < v_3$ and $v_3 < v_1$, simultaneously, which does not reflect properties of real-world applications.

Difficulty Constraints. Consider any three vertices v_1, v_2, v_3 belonging to the graph, with the true edges e_1, e_2, e_3 between $v_1 \to v_2$, $v_2 \to v_3$ and $v_1 \to v_3$ (as shown in Figure 1a). Then, the difficulty of edge e_3 will be less than the difficulty of both e_1, e_2. This is because intuitively, the difficulty of comparing a pair of items that are far apart is less than the difficulty of comparing nearby items. The difference in the ranks of v_1 and v_3 is obviously greater than that between v_1, v_2 and v_2, v_3, making the comparison between v_1, v_3 easier than the comparison between v_1, v_2 and v_2, v_3. This relationship allows us to impose constraints on the edge difficulties, and provides us with extra information that we utilize to reach a more accurate solution.



(a) Simple loop (b) Longer loop

Figure 1: Examples of difficulty constraints on loops.

We can easily extend this model to add difficulty constraints for loops with more than 3 vertices. For example, in Figure 1b, we demonstrate how difficulty constraints are enumerated for a closed loop with five vertices. Here, e_i are the actual edges present in the graph, and ϵ_i are introduced as psuedo-edges for the purpose of creating triangle inequality constraints as shown in Figure 1a. Thus, the loop below can be decomposed into 3 loops of the form shown in Figure 1a, namely $v_1 - v_2 - v_3$, $v_1 - v_3 - v_4$ and $v_1 - v_4 - v_5$. We can now impose constraints on these loops as before.

2.1 Formal Problem Statement

We define our crowd-sorting problem formally as follows: given (1) a set of items $X = \{x_1, \ldots, x_n\}$, and (2) a set of crowd responses on pairwise comparison tasks between $x_i, x_j \in X$, we wish to find the most probable partial ordering on X. This is equivalent to the problem of finding a DAG within an undirected graph G, where the n vertices V of G represent X, and the m edges E of G represent the pairwise comparisons in X.

2.1.1 Difficulty Model. The difficulty d_j of an edge in the graph represents how likely the workers are to be correct about the direction of the edge. The difficulty values $d_j \in [0, 1]$, where $d_j = 0$ implies the task is misleading and $d_j = 1$ implies that the task is extremely easy. At $d_j = 0.5$, all workers are randomly guessing.

2.1.2 Difficulty Constraints. Formally, we impose the following constraints on all of the d_i's in the graph: For all closed loops in G of the form in Figure 1a, $d_3 \leq \min(d_1, d_2)$. We encode this constraint because e_3 compares two items farther apart, while e_1 and e_2 is a comparison between two nearby items. These constraints are introduced for all closed loops in the graph.

2.1.3 Error Model. We begin with a simple error model; we generalize to take into account worker accuracies in Section 3.2.2. Let z_j represent the true edge direction of an edge e_j, and $w_{k,j}$ represent the response of the k^{th} worker on e_j. Then, $P(w_{j,k} = z_j) = 1 - d_j$. Here, $z_j \in \{\text{left-direction } (l), \text{right-direction } (r)\}$.

3 EXPECTATION MAXIMIZATION SOLUTION

We first discuss two simpler methods for solving this problem: (1) **EM.** This method adapts the EM methodology commonly used in crowdsourcing tasks [6], without trying to infer a DAG and therefore allows the existence of cycles. (2) **Graph EM.** This method extends prior sorting work, and uses Expectation-Maximization (EM) to infer the DAG without using difficulty constraints. More specifically, the E-step of the solution predicts the most probable DAG by ensuring no cycles exist, and the M-step finds the difficulty estimates of each edge.

Comparing against the EM and Graph-EM baselines allows us to study the importance of the cycle breaking method and the difficulty constraints respectively.

Our Approach. We now present our EM solution as follows. We call our approach Constrained Graph EM (C-GEM).

(1) Initialize $d_j \in [0, 1]\ \forall j$ such that it is a valid solution.
(2) E-Step: Find the directed acyclic graph over G by finding the most probable edge directions for all edges in G, and then obtaining a DAG using cycle-breaking (described below).
(3) M-Step: Maximize (over d_j) the likelihood of the worker responses given the DAG.

3.1 E-Step

Let G' represent a DAG over the vertices of the undirected graph G. The naive estimation of the most probable DAG $\hat{G'}$ over G requires enumerating all possible DAGs, which is exponential. Therefore, we determine $\hat{G'}$ by decomposing the underlying graph G into edges, and then finding the most probable direction of each edge ($\hat{z_j}$). Let the possible directions of the edges be represented by l, r, then $\hat{z_j} = \arg\max_{z_j=l,r} p(z_j|d_j, w_{k,j})\ \ \forall\ k$, where $d_j \in [0, 1]$.

Given $\hat{z_j}$, we can obtain a directed graph over G. However, in most cases the directed graph thus obtained will not be acyclic. To obtain a DAG version of the $\hat{G'}$, we need to reverse a set of edges so as to break the cycles present in $\hat{G'}$, while still maintaining a high likelihood of $\hat{G'}$. Therefore, to break a cycle, we reverse edges in the cycle on the basis of $p(\hat{z_j}|d_j, w_j)$, so that after cycle breaking, the sum of $p(\hat{z_j}|d_j, w_j)$ for all of the edges in the cycle is maximum.

This problem can be shown to be NP-complete, using the minimum feedback arc problem, once the probability of each edge is set as 1. However, many tractable approximation algorithms exist for this problem. Below, we discuss our approximation for cycle-breaking to obtain a DAG over G.

3.1.1 Cycle-Breaking. We use a greedy strategy to perform cycle breaking. We sort the edges of the graph based on their confidence values (from the E-step), and then incrementally construct an acyclic graph $\overline{G}$ from $\hat{G'}$ by selecting the most confident edge in $\hat{G'}$, and adding it to $\overline{G}$ if it doesn't create a cycle in $\overline{G}$, and adding its reverse in case it forms a cycle. It can be shown that this approach converges to a DAG.

3.2 M-Step

In this step, we maximize the log-likelihood of the observed dataset $\mathcal{D}$ (i.e. worker responses on edges) given $\overline{G}$ over E.

$$LL(\mathcal{D}|\overline{G}, d_1, \ldots, d_m) = \sum_{j=1}^{m} \alpha_j \log(1 - d_j) + \beta_j \log(d_j)$$

where α_j and β_j are the numbers of workers that give labels agreeing and disagreeing with z_j respectively.

While maximizing the log-likelihood of the data, we ensure that the constraints on the difficulties of edges are met. Since $LL(D; \overline{G}, d)$ is convex, and each constraint on d_j's is linear, we can use a convex solver to find the ML estimate of d_j's. The constraints are imposed from loops in G, as described below.

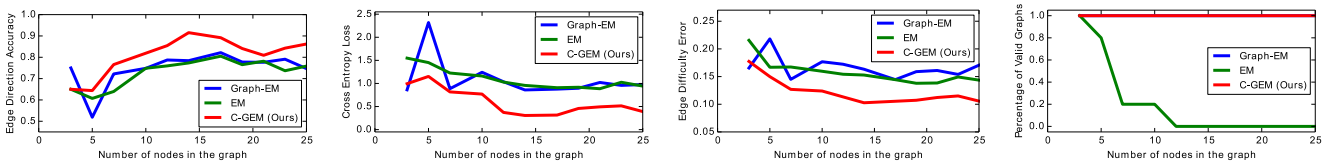
3.2.1 Enumerating Difficulty Constraints. We only enumerate difficulty constraints over those loops that are in the *cycle basis* of G. The cycle basis of a graph is defined as the minimal set of cycles (or loops) that allows every other cycle in the graph to be expressed as the disjunctive union of cycles in the basis [8]. It is easy to see that if a set of difficulty constraints is valid on two cycles in the cycle basis, then the constraints from the cycle found using the symmetric difference of the two cycles can be derived from the previous set of constraints. Therefore, we only need to find constraints on the loops in the cycle basis of G. We use the following algorithm to enumerate difficulty constraints.

(1) Find the cycle basis of G using the following procedure:
  (a) Find a Minimum Spanning Tree M of G.
  (b) For each edge e not in M, find the cycle c that e forms in M. Such a cycle must exist, otherwise e would have been a part of M.
  (c) Add c to the cycle basis of G.
(2) For each cycle c in the cycle basis, perform the following:
  (a) Find the source and sink nodes of the cycle, using the edge directions predicted in the E step of the EM algorithm.
  (b) Since in G', the edge directions over c do not form a cycle, there must be 2 paths going from the source node to the sink node in c. These paths will provide us with the longer edges and the shorter edges. Use these paths to enumerate the difficulty constraints as in Figure 1a.

3.2.2 Extension for Worker Accuracies. Let a_k represent the accuracy of worker k. We can extend our solution to handle varying worker accuracies by substituting any error model (that is a function of a_k and d_j) in $LL(D; \overline{G}, d)$, as long as $LL(D; \overline{G}, d)$ remains convex. As an example, $P(w_{j,k} = z_j) = 1 - (d_j)^{a_k}$ is one such error model. For this model, $a_k \in [0, 1]$, where $a_k = 1$ and $a_k = 0$ refer to a highly skilled and unskilled worker respectively.

4 EXPERIMENTS

We compare our algorithm with the baselines on 4 metrics: (1) **Edge Direction Accuracy** (Acc.): the fraction of correctly predicted to ground truth edge directions (higher is better); (2) **Edge Direction Cross Entropy Loss** (X-Entropy Loss) between the confidence values of each predicted and ground truth edge (lower is better); (3) **Edge Difficulty Error:** MSE between the predicted and ground truth edge difficulty values (can't be used for the real dataset, since no ground truth values exist) (lower is better); and (4) **Percentage**



(a) Edge Direction Accuracy. **(b) Cross Entropy Loss.** **(c) Edge Difficulty Error.** **(d) Fraction of Valid DAGs.**

Figure 2: Comparison of Our Method with Baselines on Simulated Data.

of Valid Graphs (% Valid): how many predicted graphs are acyclic, and therefore are acceptable solutions (higher is better).

Simulated Experiments. We test our method on simulated random graphs of varying sizes. We randomly sample the difficulty of each edge, such that the ground truth difficulties sampled are consistent with the difficulty constraints. We ensure that 10% of the edges are misleading and have $d_j < 0.5$. We randomly sample the responses of crowd workers on each edge using the error model described in Section 2.1.3. We collect 11 simulated crowd responses for each edge. For each graph size, we repeat the experiment 10 times, and report the average. Our results are shown in Figure 2.

We can see that C-GEM outperforms both baselines. The performance gain using our method is more pronounced for larger graphs. Moreover, the EM baseline frequently generates graphs that are in fact not DAGs, which is an unacceptable solution for the types of real world problems we wish to solve. In contrast, our method always predicts valid DAGs.

Our method is expected to perform well on larger graphs since larger graphs result in more difficulty constraints, allowing our method to leverage additional information about the graph structure. Additionally, there are not enough edges in smaller graphs (≤ 7 vertices) for there to be too much opportunity for improvement.

Experiments on Real Data. In this experiment, our objective is to sort neighboring pixels in an image on the basis of their depth. This task is a significant intermediate step in inferring dense depth maps for images using pairwise comparisons [3, 17]. In general, depth estimation for RGB images is an important problem where human annotations can be valuable.

For each image, we ask a crowd worker to compare 2 pixels on the image, and sort them on the basis of their relative depth. We sample 16 such points, and create 52 such comparisons between them. We repeat this for 18 images selected from the NYU Depth Dataset. Our results are presented in Table 1. We can see that C-GEM outperforms both the baselines on edge direction accuracy as well as cross entropy loss. This implies that not only are our answers more accurate, we also have higher confidence in our responses. Moreover, our method always predicts valid solutions by generating DAGs, unlike the EM baseline.

Table 1: Experimental results on real data.

Method	Acc.	X-Ent. Loss	% Valid
EM	0.728	1.888	0%
Graph-EM	0.722	1.866	100%
Our Method	0.771	1.571	100%

5 CONCLUSION

We presented a method for determining partial sort orders for a set of items. Our main contributions are a novel method of performing EM that uses information from the graph structure, as well as a cycle breaking algorithm. We showed that our method outperformed baselines on prior work on all metrics.

Acknowledgments. We acknowledge support from NSF grant IIS-1652750 and grant W911NF-18-1-0335 awarded by the ARO. The content is the responsibility of the authors and does not necessarily represent the official views of the funding agencies.

REFERENCES

[1] Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. 2009. Sorting and Selection with Imprecise Comparisons. In *ICALP (1)*. 37–48.
[2] Omar Alonso, Daniel E. Rose, and Benjamin Stewart. 2008. Crowdsourcing for relevance evaluation. *SIGIR Forum* 42, 2 (2008), 9–15.
[3] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. 2016. Single-Image Depth Perception in the Wild. In *NIPS*.
[4] Don Coppersmith, Lisa Fleischer, and Atri Rudra. 2006. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *SODA*.
[5] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. 2013. Using the crowd for top-k and group-by queries. In *ICDT*. 225–236.
[6] A. P. Dawid and A. M. Skene. 1979. Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm. *Applied Statistics* 28, 1 (1979), 20–28.
[7] Stephen Guo, Aditya G. Parameswaran, and Hector Garcia-Molina. 2012. So who won?: dynamic max discovery with the crowd. In *SIGMOD Conference*.
[8] Telikepalli Kavitha, Christian Liebchen, Kurt Mehlhorn, Dimitrios Michail, Romeo Rizzi, Torsten Ueckerdt, and Katharina Anna Zweig. 2009. Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review* 3 (2009), 199–243.
[9] K. B. Lakshmanan, B. Ravikumar, and K. Ganesan. 1991. Coping with erroneous information while sorting. In *IEEE Transactions on Computers*.
[10] Qiang Liu, Jian Peng, and Alexander Ihler. 2012. Variational Inference for Crowdsourcing. In *NIPS*. 701–709.
[11] Adam Marcus, Eugene B Wu, David R. Karger, Samuel Madden, and Rob Miller. 2011. Human-powered Sorts and Joins. *PVLDB* 5 (2011), 13–24.
[12] Vassilis Polychronopoulos, Luca de Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. 2013. Human-Powered Top-k Lists. In *Proceedings of the 16th International Workshop on the Web and Databases 2013, WebDB 2013, New York, NY, USA, June 23, 2013*. 25–30. http://webdb2013.lille.inria.fr/Paper%2035.pdf
[13] Vikas C. Raykar and Shipeng Yu. 2012. Eliminating Spammers and Ranking Annotators for Crowdsourced Labeling Tasks. *JMLR* 13 (2012), 491–518.
[14] Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Anna K. Jerebko, Charles Florin, Gerardo Hermosillo Valadez, Luca Bogoni, and Linda Moy. 2009. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *ICML*.
[15] Petros Venetis and Hector Garcia-Molina. 2012. *Dynamic Max Algorithms in Crowdsourcing Environments*. Technical Report. Stanford University.
[16] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. 2009. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*.
[17] Daniel Zoran, Phillip Isola, Dilip Krishnan, and William T. Freeman. 2015. Learning Ordinal Relationships for Mid-Level Vision. *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), 388–396.