So Who Won? Dynamic Max Discovery with the Crowd

Stephen Guo
Stanford University
Stanford, CA, USA
sdguo@cs.stanford.edu

Aditya Parameswaran
Stanford University
Stanford, CA, USA
adityagp@cs.stanford.edu

Hector Garcia-Molina
Stanford University
Stanford, CA, USA
hector@cs.stanford.edu

ABSTRACT

We consider a crowdsourcing database system that may cleanse, populate, or filter its data by using human workers. Just like a conventional DB system, such a crowdsourcing DB system requires data manipulation functions such as select, aggregate, maximum, average, and so on, except that now it must rely on human operators (that for example compare two objects) with very different latency, cost and accuracy characteristics. In this paper, we focus on one such function, *maximum*, that finds the highest ranked object or tuple in a set. In particularm we study two problems: given a set of votes (pairwise comparisons among objects), how do we select the maximum? And how do we improve our estimate by requesting additional votes? We show that in a crowdsourcing DB system, the optimal solution to both problems is NP-Hard. We then provide heuristic functions to select the maximum given evidence, and to select additional votes. We experimentally evaluate our functions to highlight their strengths and weaknesses.

1. INTRODUCTION

A crowdsourcing database system uses people to perform data cleansing, collection or filtering tasks that are difficult for computers to perform. For example, suppose that a large collection of maps is being loaded into the database, and we would like to add data fields to identify "features" of interest such as washed out roads, dangerous curves, intersections with dirt roads or tracks not on the maps, accidents, possible shelter, and so on. Such features are very hard for image analysis software to identify, but could be identified relatively easily by people who live in the area, who visited the area recently, or who are shown satellite images of the area. A crowdsourcing database system issues tasks to people (e.g., tag features on this map, compare the quality of one map as compared to another), collects the answers, and verifies the answers (e.g., by asking other people to check identified features). We focus on crowdsourcing systems where people are paid for their work, although in others they volunteer their work (e.g., in the Christmas Bird Count, http://birds.audubon.org/christmas-bird-count, volunteers identify birds across the US), while in other systems the tasks are presented as games that people do for fun (e.g., gwap.com). Amazon's Mechanical Turk (mturk.com) can be used by the database system to find human workers and perform the tasks, although there are many other companies now offering services in this space (CrowdFlower crowdflower.com, uTest utest.com, Microtask microtask.com, Tagasauris tagasauris.com).

Just like a conventional database system, a crowdsourcing database system will need to perform data processing functions like selects and aggregates, except that now these functions may involve interacting with humans. For example, to add a field "average movie rating" to movie tuples involves an aggregate over the user inputs. Selecting "horror movies" may involve asking humans what movies are "horror movies." The underlying operations (e.g., ask a human if a given movie is a "horror movie" or ask a human which of two cameras is best) have very different latency, cost and accuracy characteristics than in a traditional database system. Thus, we need to develop effective strategies for performing such fundamental data processing functions.

In particular, in this paper we focus on the *Max (Maximum) function*: The database has a set of objects (e.g., maps, photographs, Facebook profiles), where conceptually each object has an intrinsic "quality" measure (e.g., how useful is a map for a specific humanitarian mission, how well does a photo describe a given restaurant, how likely is it that a given Facebook profile is the actual profile of Lady Gaga). Of the set of objects, we want to find the one with the largest quality measure. While there are many possible underlying types of human operators, in this paper we focus on a pairwise operator: a human is asked to compare two objects and returns the one object he thinks is of higher quality. We call this type of pairwise comparison a *vote*.

If we ask two humans to compare the same pair of objects they may give us different answers, either because they makes mistakes or because their notion of quality is subjective. Either way, the crowdsourcing algorithm may need to submit the same vote to multiple humans to increase the likelihood that its final answer is correct (i.e. that the reported max is indeed the object with the highest quality measure). Of course, executing more votes increases the cost of the algorithm, either in running time and/or in monetary compensation given to the humans for their work.

There are two types of algorithms for the Max Problem: structured and unstructured. With a structured approach, a regular pattern of votes is set up in advance, as in a tournament. For example, if we have 8 objects to consider, we can first compare 1 to 2, 3 to 4, 5 to 6 and 7 to 8. After we get all results, we compare the 1-2 winner to the 3-4 winner and the 5-6 winner to the 7-8 winner. In the third stage, we compare the two winners to obtain the overall winner, which is declared the max. If we are concerned about voting errors, we can repeat each vote an odd number of times and use the consensus result. For instance, three humans can be asked to do

the 1-2 comparison, and the winner of this comparison is the object that wins in 2 or 3 of the individual votes.

While structured approaches are very effective in predictable environments (such as in a sports tournament), they are much harder to implement in a crowdsourcing database system, where humans may simply not respond to a vote, or may take an unacceptably long time to respond. In our 8-object example, for instance, after asking for the first 4 votes, and waiting for 10 minutes, we may have only the answers to the 1-2 and 5-6 comparisons. We could then re-issue the 3-4 and 7-8 comparisons and just wait, but perhaps we should also try comparing the winner of 1-2 with the winner of 5-6 (which was not in our original plan).

The point is that even if we start with a structured plan in mind, because of incomplete votes we will likely be faced with an unstructured scenario: some subset of the possible votes have completed (some with varying numbers of repetitions), and we have to answer one or both of the following questions:

- *Judgment Problem*: what is our current best estimate for the overall max winner?
- *Next Votes Problem*: if we want to invoke more votes, which are the most effective ones to invoke, given the current standing of results?

In this paper we focus precisely on these two problems, in an unstructured setting that is much more likely to occur in a crowdsourcing database system. Both of these problems are quite challenging because there may be many objects in the database, and because there are many possible votes to invoke. An additional challenge is contradictory evidence. For instance, say we have three objects, and one vote told us 1 was better than 2, another vote told us that 2 was better than 3, and a third one told us that 3 was better and 1. What is the most likely max in a scenario like this one where evidence is in conflict? Should we just ignore "conflicting" evidence, but how exactly do we do this? Yet another challenge is the lack of evidence for some objects. For example, say our evidence is that 1 is better than objects 2, 3 and 4. However, there are two additional objects, 5 and 6, for which there is no data. If we can invoke one more vote, should we compare the current favorite, object 1, against another object to verify that it is the max, or should we at least try comparing 5 and 6, for which we have no information?

The Judgment Problem draws its roots from the historical *paired comparisons* problem, wherein the goal is to find the best ranking of objects when noisy evidence is provided [20, 29, 12]. The problem is also related to the *Winner Determination* problem in the economic and social choice literature [6], wherein the goal is to find the best object via a *voting rule*: either by finding a "good" ranking of objects and then returning the best object(s) in that ranking, or by scoring each object and returning the best scoring object(s). As we will see in Section 2, our solution to the Judgment Problem differs from both of these approaches. As far as we know, no counterpart of the Next Votes problem exists in the literature. We survey related work in more detail in Section 4.

In summary, our contributions are as follows:

- We formalize the Max Problem for a crowdsourcing database system, with its two subproblems, the Judgment and the Next Votes problems.
- We propose a Maximum Likelihood (ML) formulation of the Judgment Problem, which finds the object that is probabilistically the most likely to be the maximum. We show that computing the Maximum Likelihood object is NP-Hard, while computation of the probabilities involved is #P-Hard. To the best of our knowledge, our ML formulation is the first formal definition and analysis of the Judgment Problem.
- We propose and evaluate four different heuristics for the Judgment Problem, some of which are adapted from solutions for sorting with noisy comparisons. For small problem settings, we compare the heuristic solutions to those provided by ML. When there is only a small number of votes available, we show that one of our methods, a novel algorithm based on PageRank, is the best heuristic.
- We provide the first formal definition of the Next Votes Problem, and again, propose a formulation based on ML. We show that selecting optimal additional votes is NP-Hard, while computation of the probabilities involved is #P-Hard.
- We propose four novel heuristics for the Next Votes Problem. We experimentally evaluate the heuristics, and when feasible, compare them to the ML formulation.

2. JUDGMENT PROBLEM

2.1 Problem Setup

Objects and Permutations: We are given a set O of n objects $\{o_1, ..., o_n\}$, where each object o_i is associated with a latent *quality* c_i, with no two c's being the same. If $c_i > c_j$, we say that o_i is *greater* than o_j. Let π denote a permutation function, e.g., a bijection from N to N, where $N = \{1, ..., n\}$. We use $\pi(i)$ to denote the *rank*, or index, of object o_i in permutation π, and $\pi^{-1}(i)$ to denote the object index of the ith position in permutation π. If $\pi(i) < \pi(j)$, we say that o_i is ranked *higher* than o_j in permutation π. Since no two objects have the same quality, there exists a *true* permutation π^* such that for any pair (i, j), if $\pi^*(i) < \pi^*(j)$, then $c_{\pi^*(i)} > c_{\pi^*(j)}$. Note that throughout this paper, we use the terms *permutation* and *ranking* interchangeably.

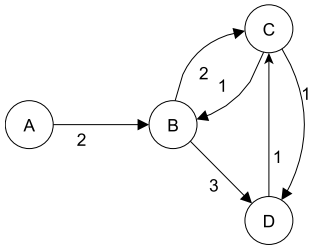
Voting: We wish to develop an *algorithm* to find the maximum (greatest) object in set O, i.e., to find $\pi^{*-1}(1)$. The only type of operation or information available to an algorithm is a pairwise *vote*: in a vote, a human worker is shown two objects o_i and o_j, and is asked to indicate the greater object. We assume that all workers vote correctly with probability p ($0.5 < p \leq 1$), where p is a quantity indicating average worker accuracy. We also assume p is unaffected by worker identities, object values, or worker behavior. In other words, each vote can be viewed as a independent Bernoulli trial with success probability p. Note that in general the value p is not available to the algorithm, but we may use it for evaluating the algorithm. However, for reference we do study two algorithms where p is known.

Goals: No matter how the algorithm decides to issue vote requests to workers, at the end it must select what it thinks is the maximum object based on the evidence, i.e., based on the votes completed so far. We start by focusing on this *Judgment Problem*, which we define as follows:

> PROBLEM 1 (JUDGMENT). *Given W, predict the maximum object in O, $\pi^{*-1}(1)$.*

In Section 3, we then address the other important problem, i.e., how to request additional votes (in case the algorithm decides it is not done yet). In general, a solution to the Judgment Problem is based upon a *scoring function* s. The scoring function first computes a *score* $s(i)$ for each object o_i, with the score $s(i)$ representing the "confidence" that object o_i is the true maximum. As we will see, for some strategies scores are actual probabilities, for others they are heuristic estimates. Then, the strategy selects the object with the largest score as its answer.

Representation: We represent the evidence obtained as an $n \times n$ vote matrix W, with w_{ij} being the number of votes for o_j being greater than o_i. Note that $w_{ii} = 0$ for all i. No other assumptions



$$W = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 1: How should these objects be ranked? Vote matrix (left) and equivalent graph representation (right). Arc weights indicate number of votes.

are made about the structure of matrix W. The evidence can also be viewed as a directed weighted graph $G_v = (V, A)$, with the vertices being the objects and the arcs representing the vote outcomes. For each pair (i, j), if $w_{ij} > 0$, arc (i, j) with weight w_{ij} is present in A. For example, Figure 1 displays a sample vote matrix and equivalent graph representation. In this example, object 1 is called A, object 2 is B, and so on. For instance, there is an arc from vertex (object) B to vertex C with weight 2 because $w_{2,3} = 2$, and there is a reverse arc from C to B with weight 1 because $w_{3,2} = 1$. If there are no votes ($w_{ij} = 0$, as from B to A), we can either say that there is no arc, or that the arc has weight 0.

2.2 Maximum Likelihood

Preliminaries: We first present a Maximum Likelihood (ML) formulation of the Judgment Problem. We directly compute the object that has the highest probability of being the maximum object in O, given vote matrix W. Assuming that average worker accuracy p is known, the ML formulation we present is the optimal feasible solution to the Judgment Problem.

Let $\boldsymbol{\pi}$ be a random variable over the set of all $n!$ possible permutations, where we assume a-priori that each permutation is equally likely to be observed. We denote the probability of a given permutation π_d given the vote matrix W as $P(\boldsymbol{\pi} = \pi_d|W)$. For the ease of exposition, we adopt the shorthand $P(\pi_d|W)$ instead of writing $P(\boldsymbol{\pi} = \pi_d|W)$. To derive the formula for $P(\pi_d|W)$, we first apply Bayes' theorem,

$$P(\pi_d|W) = \frac{P(W|\pi_d)P(\pi_d)}{P(W)} = \frac{P(W|\pi_d)P(\pi_d)}{\sum_j P(W|\pi_j)P(\pi_j)} \quad (1)$$

From our assumption that the prior probabilities of all permutations are equal, $P(\pi_d) = \frac{1}{n!}$.

Now consider $P(W|\pi_d)$. Given a permutation π_d, for each unordered pair $\{i, j\}$, the probability $f_{\pi_d}(i, j)$ of observing w_{ij} and w_{ji} is the binomial distribution probability mass function (p.m.f.):

$$f_{\pi_d}(i,j) = \begin{cases} \binom{w_{ij}+w_{ji}}{w_{ij}} p^{w_{ji}}(1-p)^{w_{ij}} & \text{if } \pi_d(i) < \pi_d(j) \\ \binom{w_{ij}+w_{ji}}{w_{ji}} p^{w_{ij}}(1-p)^{w_{ji}} & \text{if } \pi_d(j) < \pi_d(i) \end{cases} \quad (2)$$

Note that if both w_{ij} and w_{ji} are equal to 0, then $f_{\pi_d}(i, j) = 1$. Now, given a permutation π_d, observing the votes involving an unordered pair $\{i, j\}$ is conditionally independent of observing the votes involving any other unordered pair. Using this fact, $P(W|\pi_d)$, the probability of observing all votes given a permutation π_d is simply:

$$P(W|\pi_d) = \prod_{i,j:i<j} f_{\pi_d}(i,j) \quad (3)$$

Since we know the values of both p and W, we can derive a formula for $P(\pi_d|W)$ in Equation 1. In particular, the most likely permutation(s), is simply:

$$\arg\max_d P(\pi_d|W) \quad (4)$$

The permutations optimizing Equation 4 are also known as *Kemeny permutations* or *Kemeny rankings* [9].

For example, consider the matrix W of Figure 1. We do not show the computations here, but it turns out that the two most probable permutations of the objects are (D, C, B, A) and (C, D, B, A), with all other permutations having lower probability. This result roughly matches our intuition, since object A was never voted to be greater than any of the other objects, and C and D have more votes in favor over B.

We can derive the formula for the probability that a given object o_j has a given rank k. Let $\boldsymbol{\pi}^{-1}(i)$ denote the position of object i in the permutation associated with random variable $\boldsymbol{\pi}$. We are interested in the probability $P(\boldsymbol{\pi}^{-1}(k) = j|W)$. Since the event $(\boldsymbol{\pi} = \pi_d)$ is disjoint for different permutations π_d, we have:

$$P(\boldsymbol{\pi}^{-1}(k) = j|W) = \sum_{d:\pi_d^{-1}(k)=j} P(\pi_d|W)$$

Substituting for $P(\pi_d|W)$ using Equation 1 and simplifying, we have:

$$P(\boldsymbol{\pi}^{-1}(k) = j|W) = \frac{\sum_{d:\pi_d^{-1}(k)=j} P(W|\pi_d)}{\sum_l P(W|\pi_l)} \quad (5)$$

Since we are interested in the object o_j with the highest probability of being rank 1, e.g., $P(\boldsymbol{\pi}^{-1}(1) = j|W)$, we now have the Maximum Likelihood formulation to the Judgment Problem:

ML FORMULATION 1 (JUDGMENT). *Given W and p, determine:* $\arg\max_j P(\boldsymbol{\pi}^{-1}(1) = j|W)$.

In the example graph of Figure 1, while C and D both have Kemeny permutations where they are the greatest objects, D is the more likely max over a large range of p values. For instance, for $p = 0.75$, $P(\boldsymbol{\pi}^{-1}(1) = C|W) = 0.36$ while $P(\boldsymbol{\pi}^{-1}(1) = D|W) = 0.54$. This also matches our intuition, since C has one vote where it is less than B, while D is never voted to be less than either A or B.

Maximum Likelihood Strategy: Equation 5 implies that we only need to compute $P(W|\pi_d)$ for each possible permutation π_d, using Equation 3, in order to determine $P(\boldsymbol{\pi}^{-1}(k) = j|W)$ for all values j and k. In other words, by doing a single pass through all permutations, we can compute the probability that any object o_j has a rank k, given the vote matrix W.

We call this exhaustive computation of probabilities the *Maximum Likelihood Strategy* and use it as a baseline in our experiments. Note that the ML strategy is the optimal feasible solution to the Judgment Problem. The pseudocode for this strategy is displayed in Strategy 1. The strategy utilizes a ML scoring function, which computes the score of each object as $s(j) = P(\boldsymbol{\pi}^{-1}(1) = j|W)$. The predicted max is then the object with the highest score. Note that the strategy can be easily adapted to compute the maximum likelihood of arbitrary ranks (not just first) over the set of all possible permutations.

2.3 Computational Complexity

Maximum Likelihood Permutations: We begin by considering the complexity of computing Equation 4. The problem has been shown to be NP-Hard [31]. We briefly describe the result, before moving on to the Judgment Problem.

Consider Equation 3. Let $\psi(\pi_d, W)$ denote the number of votes in W that agree with permutation π_d, e.g. $\psi(\pi_d, W) = \sum_{ij:\pi_d(j)<\pi_d(i)} w_{ij}$. Let $T(W)$ denote the total number of votes in W, e.g. $T = ||W||_1$. Equation 3 can be rewritten as:

$$P(W|\pi_d) = Zp^{\psi(\pi_d,W)}(1-p)^{T(W)-\psi(\pi_d,W)} \quad (6)$$

In this expression, Z is a constant. Note that $p^x(1-p)^{a-x}$ is an increasing function with respect to x, for $0.5 < p < 1$ and constant a. Therefore, maximizing $P(W|\pi_d)$ is equivalent to maximizing $\psi(\pi_d, W)$. This implies that computing $\arg\max_i \psi(\pi_i, W)$ is equivalent to computing the most likely permutation(s), $\arg\max_i P(\pi_i|W)$. $\psi(\cdot)$ is known as the *Kemeny metric*, and has been well studied in the economic and social choice literature [19]. Referring to the example in Figure 1, the Kemeny metric is maximized by two permutations of the objects, (D, C, B, A) or (C, D, B, A). For both these permutations, the Kemeny metric is equal to $2+2+1+3 = 8$. We next show that maximizing the Kemeny metric is equivalent to solving a classical NP-Hard problem.

Consider the directed graph $G_v = (V, A)$ representing the votes of W (Figure 1 is an example). The *minimum feedback arc set* of G_v is the smallest weight set of arcs $A^{'} \subseteq A$ such that $(V, A \backslash A^{'})$ is acyclic. Equivalently, the problem can also be seen as maximizing the weight of acyclic graph $(V, A \backslash A^{'})$.

Now, supposing we have a method to solve for the minimum feedback arc set, consider a topological ordering π_t of the vertices in the resulting acyclic graph. Referring back to Equation 6, the permutation π_t maximizes the Kemeny metric $p^{\psi(\pi,W)}$. Therefore, solving the minimum feedback arc set problem for G_v is equivalent to our original problem of finding the most likely permutation given a set of votes. Referring back to our example in Figure 1, the minimum feedback arc set is 2, equivalent to cutting arc (C, B) and one of arcs (C, D) or (D, C).

Finding the minimum feedback arc set in a directed graph is a classical NP-Hard problem, implying that the problem of finding the ML permutation given a set of votes is NP-Hard as well.

THEOREM 1. *(Hardness of Maximum Likelihood Permutation) [31] Finding the Maximum Likelihood permutation given evidence is* NP*-Hard.*

Hardness of the Judgment Problem: In Section 2.2, we presented a formulation for the Judgment Problem based on ML for finding the object most likely to be the max (maximum) object in O. Unfortunately, the strategy based on that formulation was computationally infeasible, as it required computation across all $n!$ permutations of the objects in O. We now show that the optimal solution to the problem of finding the maximum object is in fact NP-Hard using a reduction from the problem of *determining Kemeny winners* [17]. (Hudry et al. [17] actually show that *determining Slater winners in tournaments* is NP-Hard, but their proof also holds for *Kemeny winners.* We will describe the *Kemeny winner* problem below.) Our results and proof are novel.

THEOREM 2. *(Hardness of the Judgment Problem) Finding the maximum object given evidence is* NP*-Hard.*

PROOF. We first describe the Kemeny winner problem. In this proof, we use an alternate (but equivalent) view of a directed weighted graph like Figure 1. In particular, we view weighted arcs as multiple arcs. For instance, if there is an arc from vertex A to B with weight 3, we can instead view it as 3 separate edges from A to B. We use this alternate representation in our proof.

An arc $i \rightarrow j$ respects a permutation if the permutation has o_j ranked higher than o_i (and does not if the permutation has o_i ranked higher than o_j). A *Kemeny permutation* is simply a permutation of the vertices (objects), such that the number of arcs that do not respect the permutation is minimum. There may be many such permutations, but there always is at least one such permutation. The starting vertex (rank 1 object) in any of these permutations is a *Kemeny winner*. It can be shown that finding a Kemeny winner is NP-Hard (using a reduction from the feedback arc set problem, similar to the proof in Hudry et al. [17]).

We now reduce the Kemeny winner determination problem to one of finding the maximum object. Consider a directed weighted graph G, where we wish to find a Kemeny winner. We show that with a suitable probability p, which we set, the maximum object (i.e., the solution to the Judgment Problem) in G is a Kemeny winner. As before, the probability that a certain object o_j is the maximum object is the right hand side of Equation 5 with k set to 1. The denominator can be ignored since it is a constant for all j. We set worker accuracy p to be very close to 1. In particular, we choose a value p such that $\frac{1-p}{p} < \frac{1}{n!}$.

Now, consider all permutations π_d that are not Kemeny permutations. In this case, it can be shown that $\sum_{d:\pi_d \text{ is not Kemeny}} P(W|\pi_d) < P(W|\pi_s)$ for any Kemeny permutation π_s. Thus, the object o_j that maximizes Equation 5 (for $k = 1$) has to be one that is a Kemeny winner.

To see why $\sum_{d:\pi_d \text{ is not Kemeny}} P(W|\pi_d) < P(W|\pi_s)$ for a Kemeny permutation π_s, notice that the left hand side is at most $n! \times P(W|\pi_d^{'})$ where $\pi_d^{'}$ is the permutation (not Kemeny) that has the least number of arcs that do not respect the permutation. Note that $P(W|\pi_d^{'})$ is at most $P(W|\pi_s) \times \frac{1-p}{p}$, since this permutation has at least one more mistake as compared to any Kemeny permutation.

Therefore, we have shown that, for a suitable p, the maximum object in G is a Kemeny winner. Thus, we have a reduction from the Kemeny winner problem to the Judgement problem. Since finding a Kemeny winner is NP-Hard, this implies that finding the maximum object in G is NP-Hard. □

#P-Hardness of Probability Computations: In addition to being NP-Hard to find the maximum object, we can show that evaluating the numerator of the right hand side of Equation 5 (with $k = 1$) is #P-Hard, in other words: computing $P(\boldsymbol{\pi}^{-1}(1) = j, W)$ is #P-Hard.

We use a reduction from the problem of counting the number of linear extensions in a directed acyclic graph (DAG), which is known to be #P-Hard.

THEOREM 3. *(#P-Hardness of Probability Computation) Computing $P(\boldsymbol{\pi}^{-1}(1) = j, W)$ is #P-Hard.*

PROOF. A linear extension is a permutation of the vertices, such that all arcs in the graph respect the permutation (i.e., a linear extension is the same as a Kemeny permutation for a DAG).

Consider a DAG $G = (V, A)$. We add an additional vertex x such that there is an arc from each of the vertices in G to x, giving a new graph $G' = (V', A')$. We now show that computing $P(\boldsymbol{\pi}^{-1}(1) = x, W)$ in G' can be used to compute the number of

linear extensions in G. Notice that:

$$P(\boldsymbol{\pi}^{-1}(1) = x, W) = \sum_{i=0}^{|A'|} a_i p^i (1-p)^{|A'|-i}$$

$$= p^{|A'|} \times \sum_{i=0}^{|A'|} a_i (\frac{1-p}{p})^{|A'|-i} \quad (7)$$

where a_i is the number of permutations where there are i arcs that respect the permutation. Clearly, the number that we wish to determine is $a_{|A'|}$, since that is the number of permutations that correspond to linear extensions. Equation 7 is a polynomial of degree $|A'|$ in $\frac{1-p}{p}$, thus, we may simply choose $|A'|+1$ different values of $\frac{1-p}{p}$, generate $|A'|+1$ different graphs G', and use the probability computation in Equation 7 to create a set of $|A'|+1$ equations involving the a_i coefficients. We may then derive the value of $a_{|A|}$ using Lagrange's interpolation formula.

Since vertex x is the only maximum vertex in G', by computing $P(\boldsymbol{\pi}^{-1}(1) = x, W)$ in G', we count the number of linear extensions in DAG G. Since counting the number of linear extensions in a DAG is #P-Hard, this implies that the computation of $P(\boldsymbol{\pi}^{-1}(1) = x, W)$ in G' is #P-Hard, which implies that the computation of $P(\boldsymbol{\pi}^{-1}(k) = j, W)$ for directed graph G_v (associated with vote matrix W) is #P-Hard. □

Strategy 1 Maximum Likelihood

Require: n objects, probability p, vote matrix W
Ensure: *ans* = maximum likelihood maximum object
  $s[\cdot] \leftarrow 0$ {$s[i]$ is used to accumulate the probability that i is the maximum object}
  for each permutation π of n objects **do**
    $prob \leftarrow 1$ {$prob$ is the probability of permutation π given vote matrix W}
    for each tuple $(i, j) : i < j$ **do**
      if $\pi(i) < \pi(j)$ **then**
        $prob \leftarrow prob \times \binom{w_{ij}+w_{ji}}{w_{ij}} p^{w_{ji}} (1-p)^{w_{ij}}$
      else
        $prob \leftarrow prob \times \binom{w_{ij}+w_{ji}}{w_{ji}} p^{w_{ij}} (1-p)^{w_{ji}}$
      end if
    end for
    $s[\pi^{-1}(1)] \leftarrow s[\pi^{-1}(1)] + prob$
  end for
  $ans \leftarrow \text{argmax}_i\, s[i]$

2.4 Heuristic Strategies

The ML scoring function is computationally inefficient and also requires prior knowledge of p, the average worker accuracy, which is not available to us in real-world scenarios. We next investigate the performance and efficiency of four heuristic strategies, each of which runs in polynomial time. The heuristics we present, excluding the Indegree heuristic, do not require explicit knowledge of the worker accuracy p.

Indegree Strategy: The first heuristic we consider is an Indegree scoring function proposed by Coppersmith et al. [11] to approximate the optimal feedback arc set in a directed weighted graph where arc weights l_{ij}, l_{ji} satisfy $l_{ij} + l_{ji} = 1$ for each pair of vertices i and j. In this section, we describe how to transform our vote matrix W to a graph where this Indegree scoring function can be applied. Let $\boldsymbol{\pi}(i)$ denote the rank, or index, of object o_i in the permutation associated with random variable $\boldsymbol{\pi}$.

Given vote matrix W, we construct a complete graph between all objects where arc weights l_{ji} are equal to $P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji})$. l_{ji} reflects the probability that o_i is greater than o_j given the *local* evidence w_{ij} and w_{ji}. It is important to note that this method is a heuristic, e.g., we compute $P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji})$, rather than $P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|W)$, which requires full enumeration over all $n!$ permutations.

How do we compute arc weight $P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji})$?

$$P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji}) = \frac{P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j))P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j))}{P(w_{ij}, w_{ji})} \quad (8)$$

Assuming that a priori all permutations π are equally likely, $P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)) = P(\boldsymbol{\pi}(j) < \boldsymbol{\pi}(i))$ by symmetry. Using Equation 8 to find expressions for $P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji})$ and $P(\boldsymbol{\pi}(i) > \boldsymbol{\pi}(j)|w_{ij}, w_{ji})$, we can derive the following:

$$\frac{P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji})}{P(\boldsymbol{\pi}(i) > \boldsymbol{\pi}(j)|w_{ij}, w_{ji})} = \frac{P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j))}{P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) > \boldsymbol{\pi}(j))} \quad (9)$$

Since $P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji}) + P(\boldsymbol{\pi}(j) < \boldsymbol{\pi}(i)|w_{ij}, w_{ji}) = 1$, we can simplify Equation 9 to get the following expression:

$$P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji}) = \frac{P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j))}{P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)) + P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) > \boldsymbol{\pi}(j))} \quad (10)$$

Using Equation 2, $P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j))$ and $P(w_{ij}, w_{ji}|\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j))$ can be computed directly from the binomial distribution p.m.f. Therefore, we can compute the arc weight $l_{ji} = P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji})$ needed for the Indegree scoring function. It should be clear that $l_{ij} + l_{ji} = 1$. Also, if w_{ij} and w_{ji} are both equal to zero, then both l_{ij} and l_{ji} are computed to be 0.5.

Strategy 2 Indegree

Require: n objects, probability p, vote matrix W
Ensure: *ans* = predicted maximum object
  $s[\cdot] \leftarrow 0$
  for $i : 1 \ldots n$ **do**
    for $j : 1 \ldots n, j \neq i$ **do**
      $s[i] \leftarrow s[i] + l_{ji}$ {$l_{ji} = P(\boldsymbol{\pi}(i) < \boldsymbol{\pi}(j)|w_{ij}, w_{ji})$}
    end for
  end for
  $ans \leftarrow \text{argmax}_i\, s[i]$

The Indegree scoring function, displayed in Strategy 2, computes the score of object o_j as: $s(j) = \sum_i l_{ij}$. Intuitively, vertices with higher scores correspond to objects which have compared favorably to other objects, and hence should be ranked higher. The predicted ranking has been shown to be a constant factor approximation to the feedback arc set for directed graphs where all arcs (i, j) are present and $l_{ij} + l_{ji} = 1$ [11]. The running time of this heuristic is dominated by the time to do the final sort of the scores.

Let us walk through the example graph in Figure 1. First, for those pairs of vertices that do not have any votes between them, we have $l_{AC} = 0.5, l_{CA} = 0.5, l_{AD} = 0.5$, and $l_{DA} = 0.5$. By symmetry, $l_{CD} = 0.5$ and $l_{DC} = 0.5$. Given a value of p, we use Equation 10 to compute the rest of the arc weights. For $p = 0.55$, we have $l_{AB} = 0.599, l_{BA} = 0.401, l_{BC} = 0.55, l_{CB} = 0.45, l_{BD} = 0.646$, and $l_{DB} = 0.354$. With these computed arc weights, we obtain the scores: $s(A) = 1.401, s(B) = 1.403, s(C) = 1.55$, and $s(D) = 1.65$, generating a predicted ranking of (D, C, B, A),

with object D being the predicted maximum object. Note that if p is larger, e.g. $p = 0.95$, the Indegree heuristic predicts the same ranking.

Local Strategy: The Indegree heuristic is simple to compute, but only takes into account local evidence. That is, the score of object o_i only depends on the votes that include o_i directly. We now consider a Local scoring function, adapted from a heuristic proposed by David [13], which considers evidence two steps away from o_i. This method was originally proposed to rank objects in incomplete tournaments with ties. We adapted the scoring function to our setting, where there can be multiple comparisons between objects, and there are no ties in comparisons.

This heuristic is based on the notion of wins and losses, defined as follows: $wins(i) = \sum_j w_{ji}$ and $losses(i) = \sum_i w_{ij}$. For instance, in Figure 1, vertex B has 3 wins and 5 losses.

The score $s(i)$ has three components. The first is simply $wins(i) - losses(i)$, reflecting the net number of votes in favor of o_i. For vertex B, this first component would be $3 - 5 = -2$. Since this first component does not reflect the "strength" of the objects o_i was compared against, we next add a "reward": for each o_j such that $w_{ji} > w_{ij}$ (i has net wins over j), we add $wins(j)$ to the score of o_i. In our example, B only has net wins over A, so we reward B with $wins(A)$ (which in this case is zero). On the other hand, since C beat out B, then C gets a reward of $wins(B) = 3$ added to its score. Finally, we "penalize" $s(i)$ by subtracting $losses(j)$ for each o_j that overall beat o_i. In our example, we subtract from $s(B)$ both $losses(C) = 2$ and $losses(D) = 1$. Thus, the final score $s(B)$ is -2 plus the reward minus the penalty, i.e., $s(B) = -2 + 0 - 3 = -5$.

More formally, score $s(i)$ is defined as follows:

$$s(i) = wins(i) - losses(i) + \sum_j [\mathbf{1}(w_{ji} > w_{ij}) wins(j)] - \sum_j [\mathbf{1}(w_{ij} > w_{ji}) losses(j)] \quad (11)$$

Strategy 3 Local

Require: n objects, vote matrix W
Ensure: *ans* = predicted maximum object
  $wins[\cdot], losses[\cdot], s[\cdot] \leftarrow 0$ {objects are ranked by $s[\cdot]$}
  for each tuple (i, j) : **do**
    $wins[j] \leftarrow wins[j] + w_{ij}$
    $losses[i] \leftarrow losses[i] + w_{ij}$
  end for
  for $i : 1 \ldots n$ **do**
    $s[i] \leftarrow wins[i] - losses[i]$ {add $wins - losses$ to s}
    for $j : 1 \ldots n, j \neq i$ **do**
      if $w_{ij} < w_{ji}$ **then**
        $s[i] \leftarrow s[i] + wins[j]$ {add reward}
      else if $w_{ij} > w_{ji}$ **then**
        $s[i] \leftarrow s[i] - losses[j]$ {subtract penalty}
      end if
    end for
  end for
  $ans \leftarrow \text{argmax}_i\, s[i]$

Having computed $s(\cdot)$, we sort all objects by decreasing order of s. The resulting permutation is our predicted ranking, with the vertex having largest s being our predicted maximum object. An implementation of the method is displayed in Strategy 3.

To complete the example of Figure 1, Strategy 3 computes the following scores: $s(A) = 0-2-5 = -7$, $s(B) = 3-5-3 = -5$, $s(C) = 3-2+3 = 4$, and $s(D) = 4-1+3 = 6$. The predicted ranking is then (D, C, B, A), with object D being the predicted maximum object.

PageRank Strategy: Both the Indegree and Local heuristics use only information one or two steps away to make inferences about the objects of O. We next consider a global heuristic scoring function inspired by the PageRank [26] algorithm. The general idea behind using a PageRank-like procedure is to utilize the votes in W as a way for objects to transfer "strength" between each other. The use of PageRank to predict the maximum has been previously considered [5] in the literature. Our contribution is a modified PageRank to predict the maximum object in O, which in particular, can handle directed cycles in the directed graph representing W.

Consider again the directed graph G_v representing the votes of W (Figure 1 is an example). Let $d^+(i)$ to denote the outdegree of vertex i in G_v, e.g. $d^+(i) = \sum_j w_{ij}$. If $d^+(i) = 0$, we say that i is a *sink* vertex.

Let $pr_t(i)$ represent the PageRank of vertex i in iteration t. We initialize each vertex to have the same initial PageRank, e.g., $pr_0(\cdot) = \frac{1}{n}$. In each iteration $t+1$, we apply the following update equation to each vertex i:

$$pr_{t+1}(i) = \sum_j \frac{w_{ji}}{d^+(j)} pr_t(j) \quad (12)$$

For each iteration, each vertex j transfers all its PageRank (from the previous iteration) proportionally to the other vertices i whom workers have indicated may be greater than j, where the proportion of j's PageRank transferred to i is equal to $\frac{w_{ji}}{d^+(j)}$. Intuitively, $pr_t(i)$ can be thought as a proxy for the probability that object o_i is the maximum object in O (during iteration t).

What happens to the PageRank vector after performing many update iterations using Equation 12? Considering the strongly connected components (SCCs) of G_v, let us define a *terminal* SCC to be a SCC whose vertices do not have arcs transitioning out of the SCC. After a sufficient number of iterations, the PageRank probability mass in G_v becomes concentrated in the terminal SCCs of G_v, with all other vertices outside of these SCCs having zero PageRank [4]. In the context of our problem, these terminal SCCs can be thought of as sets of objects which are ambiguous to order.

Our proposed PageRank algorithm is described in Strategy 4. How is our strategy different from the standard PageRank algorithm? The original PageRank update equation is:

$$pr_{t+1}(i) = \frac{1-d}{n} + d \sum_j \frac{w_{ji}}{d^+(j)} pr_t(j))$$

Comparing the original equation and Equation 12, the primary difference is that we use a damping factor $d = 1$, e.g. we remove jump probabilities. PageRank was designed to model the behavior of a random surfer traversing the web, while for the problem of ranking objects, we do not need to model a random jump vector.

A second difference between our modified PageRank and the original PageRank is that prior to performing any update iterations, for each sink vertex i, we set w_{ii} equal to 1 in W. In our setting, sinks correspond to objects which may be the maximum object (e.g., no worker voted that o_i is less than another object). By setting w_{ii} to 1 initially, from one iteration to the next, the PageRank in sink i remains in sink i. This allows PageRank to accumulate in sinks. Contrast this with the standard PageRank methodology, where when a random surfer reaches a sink, it is assumed that (s)he transitions to all other vertices with equal probability.

Finally, a caveat to our PageRank strategy is that the PageRank vector ($pr(\cdot)$ in Strategy 4) may not converge for some vertices

Strategy 4 PageRank

Require: n objects, vote matrix W, K iterations
Ensure: *ans* = predicted maximum object
  construct $G_v = (V, A)$ from W
  compute $d^+[\cdot]$ for each vertex {compute all outdegrees}
  for $i : 1 \ldots n$ **do**
    if $d^+[i] == 0$ **then**
      $w_{ii} \leftarrow 1$
    end if
  end for
  $pr_0[\cdot] \leftarrow \frac{1}{n}$ {pr_0 is the PageRank vector in iteration 0}
  for $k : 1 \ldots K$ **do**
    for $i : 1 \ldots n$ **do**
      for $j : 1 \ldots n, j \neq i$ **do**
        $pr_k[i] \leftarrow pr_k[i] + \frac{w_{ji}}{d^+[j]} pr_{k-1}[j]$
      end for
    end for
  end for
  compute $period[\cdot]$ of each vertex using final iterations of $pr[\cdot]$
  for $i : 1 \ldots n$ **do**
    $s[i] \leftarrow 0$ {$s[\cdot]$ is a vector storing average PageRank}
    for $j : 0 \ldots period[i] - 1$ **do**
      $s[i] \leftarrow s[i] + pr_{K-j}[i]$
    end for
    $s[i] \leftarrow \frac{s[i]}{period[i]}$
  end for
  $ans \leftarrow \text{argmax}_i\, s[i]$

in terminal SCCs. To handle the oscillating PageRank in terminal SCCs, we execute our PageRank update equation (Equation 12) for a large number of iterations, denoted as K in Strategy 4. Then, we examine the final iterations, say final 10%, of the PageRank vector to empirically determine the *period* of each vertex, where we define the period as the number of iterations for the PageRank value of a vertex to return to its current value. In practice, we find that running PageRank for K iterations, where $K = O(n)$, is sufficient to detect the period of nearly all vertices in terminal SCCs. For example, consider a graph among 3 objects A, B, C with 3 arcs: $(A, B), (B, C)$, and (C, B). All vertices initially have $\frac{1}{3}$ PageRank probability. After 1 iteration, the PageRank vector is $(0, \frac{2}{3}, \frac{1}{3})$. After 2 iterations, the PageRank vector is $(0, \frac{1}{3}, \frac{2}{3})$. And so on. In this example, object B and C each have periods of 2.

With the periods computed for each vertex, we compute an *average* PageRank value for each vertex over its period. This average PageRank is used as the scoring function $s(\cdot)$ for this strategy. After the termination of PageRank, we sort the vertices by decreasing order of $s(\cdot)$, and predict that the vertex with maximum average PageRank corresponds to the maximum object in O. Note that our PageRank heuristic is primarily intended to predict a maximum object, not to predict a ranking of all objects (as many objects will end up with no PageRank). However, for completeness, when evaluating PageRank in later experiments, we still do consider the predicted ranking induced by PageRank. The details of our implementation are displayed in Strategy 4.

To illustrate our PageRank heuristic, consider again the example in Figure 1. There are 2 SCCs in the graph: (A) and (B, C, D), with (B, C, D) being a terminal SCC. Each of the 4 vertices is initialized with 0.25 PageRank. After the first iteration, the PageRank vector is $(0, 0.375, 0.35, 0.275)$. After the second iteration, the PageRank vector is $(0, 0.375, 0.35, 0.275)$. After ~20 iterations, the PageRank vector oscillates around $(0, 0.217, 0.435, 0.348)$. With a sufficiently large number of iterations and an appropriately chosen convergence threshold, the heuristic determines a period of 1 for both SCCs and computes an average PageRank vector of $(0, 0.217, 0.435, 0.348)$. The PageRank heuristic then predicts object C to be the maximum object in O.

Heuristic	Prediction
ML	(D, C, B, A) and (C, D, B, A)
Indegree	(D, C, B, A)
Local	(D, C, B, A)
PageRank	Maximum object = C
Iterative	$(C, D, B, A), (C, D, A, B), (D, C, B, A)$, or (D, C, A, B)

Table 1: Predictions using each heuristic for Figure 1.

Iterative Strategy: We next propose an Iterative heuristic strategy to determine the maximum object in O. The general framework is the following:

1. Place all objects in a set.
2. Rank the objects in the set by a scoring metric.
3. Remove the lower ranked objects from the set.
4. Repeat steps 3 and 4 until only one object remains.

There are two parameters we can vary in this framework: the scoring metric and the number of objects eliminated each iteration. Let us define the $dif(i)$ metric of object o_i to be equal to $wins(i) - losses(i)$. An implementation of the Iterative strategy using the dif metric is displayed in Strategy 5. In our particular implementation, we emphasize computational efficiency and remove half of the remaining objects each iteration. The Iterative strategy relies upon the elimination of lower ranked objects before re-ranking higher ranked objects. With each iteration, as more objects are removed, the $difs$ of the higher ranked objects separate from the $difs$ of the lower ranked objects. Basically, by removing lower ranked objects, the strategy is able to more accurately rank the remaining set of objects. The strategy can be thought of as iteratively narrowing in on the maximum object.

It is important to note that other scoring metrics can be used with this Iterative strategy as well. For example, by iteratively ranking with the Local heuristic, we were able to achieve (slightly) better performance than the simple dif metric. Our method is similar to the Greedy Order algorithm proposed by Cohen et al. [7], who considered a problem related to feedback arc set. Our strategy differs in that it is more general (e.g., it can utilize multiple metrics), and our strategy can be optimized (e.g., if we eliminate half of the objects each iteration, we require only a logarithmic number of sorts, as opposed to a linear number).

The Iterative strategy can also be viewed as a scoring function $s(\cdot)$, like the prior heuristics we have examined. Denoted as $s[\cdot]$ in Strategy 5, we can assign each object a score equal to the iteration number in which it was removed from set T. Using this scoring function $s(\cdot)$, the predicted maximum object is then simply $\text{argmax}_i\, s(i)$.

Returning to the example graph in Figure 1, the Iterative heuristic first computes the dif metric for each object: $dif(A) = -2$, $dif(B) = -2$, $dif(C) = 1$ and $dif(D) = 3$. The objects are then placed in a set and sorted by dif. In the first iteration, objects A and B are assigned ranks 3 and 4 and removed from the set. Then, dif is recomputed among all remaining objects in the set, $dif(C) = 0$ and $dif(D) = 0$. In the second iteration, either object C or D is removed and assigned rank 2. In the third iteration, the remaining object is removed and assigned rank 1. Therefore, the predicted ranking of the Iterative heuristic is equally likely to be $(C, D, B, A), (C, D, A, B), (D, C, B, A)$, or (D, C, A, B), with the predicted maximum object of the heuristic being object C or D with equal probability.

To summarize, Table 1 displays the predictions for ML and our four heuristics for the example displayed in Figure 1. How can we determine which heuristic strategy is superior to the others?

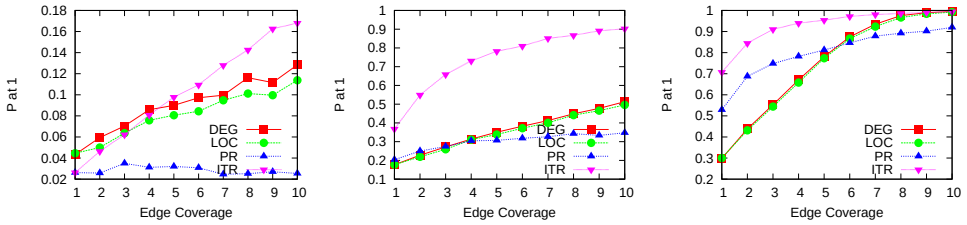


Figure 3: Precision at 1 (P@1) versus Edge Coverage. 100 objects. p=0.55 (left), p=0.75 (middle), p=0.95 (right).

Strategy 5 Iterative

Require: n objects, vote matrix W
Ensure: *ans* = predicted maximum object
  $dif[\cdot] \leftarrow 0$ {$dif[\cdot]$ is the scoring metric}
  for $i : 1 \ldots n$ **do**
    for $j : 1 \ldots n, j \neq i$ **do**
      $dif[j] \leftarrow dif[j] + w_{ij}; dif[i] \leftarrow dif[i] - w_{ij}$
    end for
  end for
  initialize set Q {which stores objects}
  for $i : 1 \ldots n$ **do**
    $Q \leftarrow Q \cup i$
  end for
  while $|Q| > 1$ **do**
    sort objects in Q by $dif[\cdot]$
    for $r : (\frac{|Q|}{2} + 1) \ldots |Q|$ **do**
      remove object i (with rank r) from Q
      for $j : j \in Q$ **do**
        if $w_{ij} > 0$ **then**
          $dif[j] \leftarrow dif[j] - w_{ij}; dif[i] \leftarrow dif[i] + w_{ij}$
        end if
        if $w_{ji} > 0$ **then**
          $dif[i] \leftarrow dif[i] - w_{ji}; dif[j] \leftarrow dif[j] + w_{ji}$
        end if
      end for
    end for
  end while
  $ans \leftarrow S[1]$ {$S[1]$ is the final object in S}

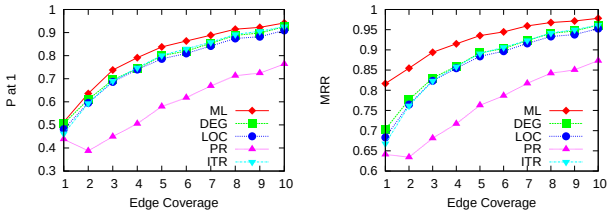


Figure 2: Comparison of ML and heuristics. Prediction performance versus Edge Coverage. 5 objects, p=0.75. P@1 (left), MRR (right).

2.5 Experiments

In this section, we experimentally compare our heuristic strategies: Indegree (DEG), Local (LOC), PageRank, (PR), and Iterative (ITR). We also compare them with the Maximum Likelihood (ML) Strategy, which we consider the best possible way to select the maximum. However, since ML is computationally very expensive, we only do this comparison on a small scenario. For our experiments, we synthetically generate problem instances, varying : n (the number of objects in O), v (the number of votes we sample for W), and p (average worker accuracy). We prefer to use synthetic data, since it lets us study a wide spectrum of scenarios, with highly reliable or unreliable workers, and with many or few votes.

In our base experiments, we vary the number of sampled votes v, from 0 to $5n(n-1)$ and vary worker accuracy p from 0.55 to 0.95. As a point of reference, we refer to $\frac{n(n-1)}{2}$ votes as v = 1x *Edge Coverage*, e.g. each pair of objects is sampled approximately once. So $5n(n-1)$ votes is equivalent to v = 10x Edge Coverage in our experiments.

Each data point (given n, p, v values) in our results graphs is obtained from 5,000 *runs*. Each run proceeds as follows: We initialize W as an $n \times n$ null matrix and begin with an arbitrary *true* permutation π^* of the objects in O. Let U denote the set of all tuples (i, j) where $i \neq j$. We randomly sample v tuples from U with replacement. After sampling a tuple (i, j), we simulate the human worker's comparison of objects o_i and o_j. If $\pi^*(i) < \pi^*(j)$, with probability p, we increment w_{ji}, and with probability $1 - p$, we increment w_{ij}. If $\pi^*(j) < \pi^*(i)$, with probability p, we increment w_{ij}, and with probability $1 - p$, we increment w_{ji}.

For each generated matrix W in a run, we apply each of our heuristic strategies to obtain predicted rankings of the objects in O. Comparing the predicted ranking with π^* we record both (a) a "yes" if the predicted maximum agrees with the true maximum, and (b) reciprocal rank, the inverse rank of the true maximum object in the predicted ranking. Finally, after all runs complete, we compute (a) Precision at 1 (P@1), the fraction of "yes" cases over the number of runs, and (b) the Mean Reciprocal Rank (MRR), the average reciprocal rank over all runs.

As a first experiment, we consider the prediction performance of Maximum Likelihood (ML) and the four heuristics for a set of 5 objects with $p = 0.75$, displayed in Figure 2. We choose a small set of objects, so that ML can be computed. Looking at Figure 2(left), we find that as the number of votes sampled increases, the P@1 of all heuristics (excluding PageRank) increase in a concave manner, approaching a value of 0.9 for 10x Edge Coverage. In other words, if $5n(n-1)$ votes are uniformly sampled, the heuristics can predict the maximum object 90% of the time, even though average worker accuracy is 0.75. Similar prediction curves measuring MRR are displayed in Figure 2(right).

> ML has better performance than all the four heuristics.

As expected, ML performs the best in Figure 2, but recall that ML requires explicit knowledge of p, and it is computationally very expensive. Still, the ML curve is useful, since it tells us how far the heuristics are from the optimal feasible solution (ML). Also, note that PageRank (PR) performs poorly in Figure 2, indicating that PageRank is poor when the number of objects is small.

> Iterative is the best of the four heuristics when the number of votes sampled is $\frac{n(n-1)}{2}$, e.g. 1x Edge Coverage.

For a larger experiment, we consider the problem of prediction for $n = 100$ objects in Figure 3. ML is necessarily omitted from this experiment. Looking at the graphs, we first note that the Iterative (ITR) heuristic performs significantly better than the other heuristics, particularly when $p = 0.55$ or $p = 0.75$. This is best demonstrated by Figure 3(middle), which shows that for $p = 0.75$ and 10x Edge Coverage, the Iterative heuristic has a P@1 of over 0.9, whereas the second best heuristic, Indegree (DEG), only has a P@1 of approximately 0.5. Looking at the middle graph again, note how the performance gap between the Iterative heuristic and

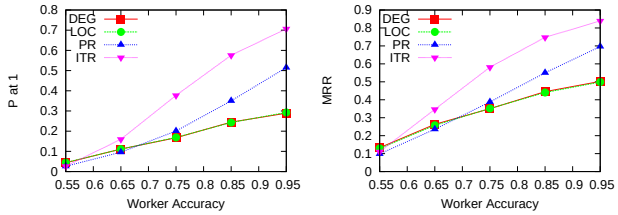


Figure 4: Prediction performance versus worker accuracy. 100 objects, 1x Edge Coverage. P@1 (left), MRR (right).

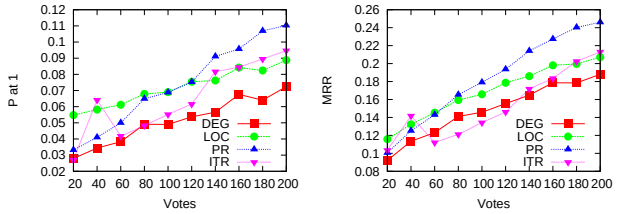


Figure 5: Sparse case. Prediction performance versus number of votes. 100 objects, p=0.95. P@1 (left), MRR (right).

the other heuristics widens as the Edge Coverage increases from 1x to 5x. The strength of the Iterative strategy comes from its ability to leverage the large number of redundant votes, in order to iteratively prune out lower-ranked objects until there is a predicted maximum. The strategy is robust even when worker accuracy is low. When average worker accuracy is high, Figure 3(right), the Iterative heuristic still is the heuristic of choice, although the performance gap between the Iterative and Indegree or Local (LOC) heuristics decreases to a minimal amount, as the number of votes sampled becomes very large.

PageRank is a poor heuristic when worker accuracy is low. However, when worker accuracy is reasonable, PageRank is quite effective, even when the number of votes is low.

We next focus upon the performance of the PageRank (PR) heuristic. For $p = 0.75$ and $p = 0.95$, the PageRank heuristic's prediction curve crosses the prediction curves for the Indegree (DEG) and Local (LOC) heuristics. This is an indication that the PageRank heuristic is quite effective when the number of votes is low, but is unable to utilize the information from additional votes when the number of votes is large. We also observe the poor performance of PageRank when $p = 0.55$, in Figure 3(left), indicating that PageRank is not a suitable heuristic when worker accuracy is low. Finally, note that the Indegree and Local heuristics perform similarly across all worker accuracies. This indicates that prior knowledge of worker accuracy p, which the Indegree heuristic requires, is not necessary to perform good prediction if a more sophisticated scoring function, such as the Local heuristic, is used.

Over various worker accuracies, Iterative is the best heuristic, followed by PageRank, Local and Indegree.

From the prior experiments, we see that prediction performance for each strategy varies greatly with respect to the average worker accuracy p. We next directly investigate prediction performance versus worker accuracy for a fixed 1x Edge Coverage. As shown in Figure 4, we find that for this fixed Edge Coverage, the Iterative (ITR) strategy performs the best, followed by PageRank (PR), then the Local (LOC) and Indegree (DEG) heuristics. As expected, prediction performance increases with worker accuracy across all strategies. In particular, note the large slope of the Iterative and PageRank prediction curves, as compared to the Local and Indegree prediction curves, which are near identical.

PageRank is the best of the four heuristics when there are few votes and worker accuracy is high.

All experiments considered thus far examine prediction when the number of votes is an order of magnitude larger than the number of objects. For a more difficult scenario, we examine prediction performance when the number of votes is approximately the same as the number of objects. Figure 5 displays prediction performance for 100 objects when the number of votes is varied from 20 to 200 and $p = 0.95$. We observe that PageRank (PR) has the highest prediction performance among the four heuristics. Conducting several other experiments, we find that, so long as worker accuracy is high, PageRank facilitates good prediction, even when the number of votes is low relative to the number of objects. This fact will prove useful when we consider the problem of selecting which additional votes to request, given an initial sparse vote graph.

From our experiments regarding prediction performance, we conclude that Iterative (ITR) is the strategy of choice when evaluating a large number of votes (relative to the number of objects), whereas PageRank is the preferred heuristic when evaluating a small number of votes.

3. NEXT VOTES PROBLEM

We now consider the second half of the Max Problem, the Next Votes Problem. Beginning with an initial vote matrix W, if we wish to submit additional vote requests to a crowdsourcing marketplace, which additional votes (i.e., comparisons between pairs of objects) should be requested to augment our existing vote matrix W, and improve our prediction of the maximum object? In particular, we assume that we are given a vote budget of b additional votes that may be requested. There are two ways in which we can use this vote budget: (a) an adaptive strategy, where we submit some initial votes, get some responses, then submit some more, get more responses, and so on, or (b) a one-shot strategy, where we submit all votes at once. In this paper, we consider a one-shot strategy with a vote budget of b. This strategy is more relevant in a crowdsourcing setting since the latency of crowdsourcing is high. Once the responses for these vote requests are received, we assume that the entire evidence thus far is our new vote matrix W'. Note that we can iteratively submit batches of votes to improve our prediction of the maximum object. As before, we assume that the response to each vote is i.i.d. correct with probability p. We define the Next Votes Problem as follows:

PROBLEM 2 (NEXT VOTES). *Given b, W, select b additional votes and predict the maximum object in O, $\pi^{*-1}(1)$.*

3.1 Maximum Likelihood

We first present a Maximum Likelihood (ML) formulation of the selection of votes for the Next Votes Problem; we directly compute the multiset of votes which most improves the prediction of the maximum object in O. Assuming that average worker accuracy p is known, the ML vote selection formulation we present is the optimal feasible solution to the Next Votes Problem. Beforing presenting the ML formulation, we first provide some definitions needed for the Next Votes Problem.

Vote and Answer Multisets: We represent a potential vote (comparison) between objects o_i and o_j as a unordered pair $\{o_i, o_j\}$. Given a vote budget b, all possible multisets Q of b votes are allowed (note that repetition of votes is allowed). For a potential vote $\{o_i, o_j\}$, we define an answer to be a tuple $(\{o_i, o_j\}, o_x)$, where the first element of the tuple is an unordered pair, and the second element is one of the objects in the pair indicating the human worker's

answer (e.g., $x = i$ if the worker states that o_i is greater than o_j, or $x = j$ otherwise).

For each vote multiset Q, we define an answer multiset a of Q to be a multiset of answer tuples, where there is a one-to-one mapping from each unordered pair in Q to an answer tuple in a. Note that each vote is answered (independently) with probability p. As an example, if $Q = \{\{o_i, o_j\}, \{o_k, o_l\}\}$, a possible answer multiset a that could be received from the workers is $\{(\{o_i, o_j\}, o_i), (\{o_k, o_l\}, o_k)\}$. Note that for a multiset of b votes, there are 2^b possible answer multisets. Let $A(Q)$ denote the multiset of all possible answer multisets of Q.

Having defined vote and answer multisets, we next consider the probability of receiving an answer multiset given W, then explain how to compute the confidence of the maximum object having received an answer multiset, before finally presenting the ML vote selection strategy.

Probabilities of Multisets and Confidences: Suppose that we submitted vote multiset Q and received answer multiset a from the crowdsourcing marketplace. Let $P(a|W)$ denote the probability of observing an answer multiset a for Q, given initial vote matrix W. We have the following:

$$P(a|W) = \frac{P(a \wedge W)}{P(W)} \tag{13}$$

where $a \wedge W$ is the new vote matrix formed by combining the votes of a and W.

Our estimate for how well we are able to predict the maximum object in O is then the probability of the maximum object, given the votes of our new vote matrix, i.e., $a \wedge W$. We denote this value by $P_{max}(a \wedge W)$, i.e., this value is our confidence in the maximum object. The computation, based upon Equation 5, is the following:

$$P_{max}(a \wedge W) = \max_i P(\boldsymbol{\pi}^{-1}(1) = i | a \wedge W)$$

This simplifies to give:

$$P_{max}(a \wedge W) = \frac{\max_i P(\boldsymbol{\pi}^{-1}(1) = i, a \wedge W)}{P(a \wedge W)} \tag{14}$$

Maximum Likelihood Strategy: We can now define the Maximum Likelihood formulation of the Next Votes Problem. We wish to find the multiset Q of b votes such that, on average over all possible answer multisets for Q (and weighted by the probability of those answer multisets), our confidence in the prediction of the maximum object is greatest.

In other words, we want to find the multiset that maximizes:

$$\sum_{a \in A(Q)} P(a|W) \times P_{max}(a \wedge W)$$

which, on using Equations 13 and 14, simplifies to:

$$\frac{1}{P(W)} \times \sum_{a \in A(Q)} \max_i P(\boldsymbol{\pi}^{-1}(1) = i, a \wedge W)$$

Since $P(W)$ is a constant, independent of Q, we have:

ML FORMULATION 2 (NEXT VOTES). *Given b, W, find the vote multiset $Q, |Q| = b$, that maximizes*

$$\sum_{a \in A(Q)} \max_i P(\boldsymbol{\pi}^{-1}(1) = i, a \wedge W) \tag{15}$$

Let $score(Q)$ be the value in Equation 15. We now have an exhaustive strategy to determine the best multiset Q: compute $score(\cdot)$ for all possible multisets of size b, and then choose the multiset with the highest score. Although this strategy is the optimal feasible solution to the Next Votes Problem, it is also computationally infeasible, since a single iteration of ML itself requires enumeration of all $n!$ permutations of the objects in O. Additionally, knowledge of worker accuracy p is required for ML vote selection. This leads us to develop our own vote selection and evaluation framework enabling more efficient heuristics.

3.2 Computational Complexity

As in the Judgment Problem, the Next Votes Problem also turns out to be NP-Hard, while the computation of the probabilities involved also turns out to be #P-Hard. While the proofs use reductions from similar problems, the details are quite different.

Hardness of the Next Votes Problem: We first show that the ML formulation for the Next Votes Problem is NP-Hard, implying that finding the optimal set of next votes to request is intractable.

THEOREM 4 (HARDNESS OF NEXT VOTES). *Finding the vote multiset Q that maximizes $\sum_{a \in A(Q)} \max_i P(\boldsymbol{\pi}^{-1}(1) = i, a \wedge W)$ is NP-Hard, even for a single vote.*

PROOF. (Sketch) Our proof for the Next Votes problem uses a reduction from the same NP-Hard problem described in Section 2.3, i.e., determining Kemeny winners.

We are given a graph G where we wish to find a Kemeny winner. We add an extra vertex v to this graph to create a new graph, G', where v does not have any incoming or outgoing arcs. By definition, v is a Kemeny winner in G', since trivially, v can be placed anywhere in the permutation without changing the number of arcs that are respected. Therefore, there are at least two Kemeny winners in G'. Recall, however, that our goal is to return a Kemeny winner in G', not in G.

Now, consider the solution to the Next Votes problem on G', where an additional vote is requested. We need to show that the two vertices returned by the Next Votes problem are both Kemeny winners in G. Let the two vertices be x, y. As before, recall that

$$P(\boldsymbol{\pi}^{-1}(1) = i, a \wedge W) = \sum_{\pi : i \ wins} P(\pi) P(W \wedge a | \pi)$$

Ignoring $P(\pi)$, which is a constant, we have two terms:

$$F = \max_i \sum_{\pi : i \ wins} P(W \wedge x > y | \pi) + \max_i \sum_{\pi : i \ wins} P(W \wedge y > x | \pi)$$

Now consider Kemeny permutations of W. Let the set of Kemeny winners be S, and let the number of Kemeny permutations beginning with each of the winners be $s_1 \geq s_2 \geq \ldots s_n$. We also let the probability p be very close to 1 so that only Kemeny permutations form part of F. If we choose two Kemeny winners as x and y, the expression F can be as large as $(s_1 + s_2) \times P$, where P is the probability corresponding to one Kemeny permutation. On the other hand, if both of x and y are not Kemeny winners, then we can show that $F < (s_1 + s_2) \times P$ (since the constraint of $x < y$ and $y > x$ eliminates some non-zero number of permutations from the right hand side of the expression.) Now it remains to be seen if x may be a Kemeny winner while y is not. Clearly, the first term can be as big as $s_1 P$. It remains to be seen if the second term can be $s_2 P$. Note that since x is Kemeny, enforcing that $y > x$ is going to discount all permutations where $max > x > y$. Thus the second term cannot be as big as $s_2 P$. Thus both the vertices returned by the Next Votes problem are Kemeny winners. Thus, the Kemeny winner determination problem on G can be reduced to the Next Votes (with one vote) problem on G'.

□

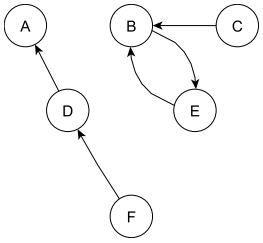


Figure 6: How should we select additional votes to request?

#P-Hardness of Probability Computations: We next show that computing Equation 15 is #P-Hard.

THEOREM 5 (#P-HARDNESS OF NEXT VOTES). *Computing* $\sum_{a \in A(Q)} \max_i P(\boldsymbol{\pi}^{-1}(1) = i, a \wedge W)$ *is #P-Hard, even for a* Q *with a single vote.*

PROOF. (Sketch) Our proof uses a reduction from the #P-Hard problem of counting linear extensions in a DAG. Consider a DAG $G = (V, A)$. We now add two additional vertices, x and y, such that there is an arc from each of the vertices in G to x and to y giving a new graph $G' = (V', A')$.

Consider the computation of $\sum_{a \in A(Q)} \max_i P(\boldsymbol{\pi}^{-1}(1) = i, a \wedge W)$ for $Q = \{\{x, y\}\}$ for G', which simplifies to: $\max_i \sum_{\pi: i \ wins} P(W \wedge (x > y)|\pi) + \max_i \sum_{\pi: i \ wins} P(W \wedge (y > x)|\pi)$.

The first of these two terms is maximized when x is the maximum, and the second term is maximized when y is the maximum. Both terms are identical, since x and y are identical, so we focus on only one of the terms. Let $F(p) = \sum_{\pi: x \ wins} P(W \wedge x > y|\pi)$. Using a calculation similar to that used to derive Equation 7, we have: $F(p) = p^{|A'|} \times \sum_{i=0}^{|A'|} a_i (\frac{1-p}{p})^{|A'|-i}$. We are interested in $a_{|A'|}$, the number of permutations that correspond to linear extensions. Once again, by repeating the trick in Theorem 3, we may use multiple values for p to generate different graphs G', and use the probability computation to derive many equations $F(p)$ corresponding to different p, and then derive the value of $a_{|A'|}$ using Lagrange's interpolation.

Therefore, counting the number of linear extensions in G can be reduced to a polynomial number of instances of computing the probability expression corresponding to the Next Votes problem. □

3.3 Selection and Evaluation of Additional Votes

We next present a general framework to select and evaluate additional votes for the Next Votes Problem. Our approach is the following:

1. score all objects with a scoring function s using initial vote matrix W
2. select a batch of b votes to request
3. evaluate the new matrix W' (initial votes in W and additional b votes) with a scoring function f to predict the maximum object in O.

This framework is displayed in more detail in Algorithm 6. In Step 1, we use a scoring function $s(\cdot)$ to score each object, and in Step 3, we use a scoring function $f(\cdot)$ to evaluate the new matrix W' to predict the maximum object in O. We briefly discuss the choice of these scoring functions when presenting experimental results later in Section 3.4. For now, we assume the use of a scoring function in Step 1 which scores objects proportional to the probability that they are the maximum object in O. It is important to note that our general framework assumes no knowledge of worker accuracy p, unlike in ML vote selection. We next focus our attention upon how to select b additional votes (Step 2).

Heuristic Vote Selection Strategies: How should we select pairs of objects for human workers to compare, when given a vote budget of b votes? Since ML vote selection is computationally infeasible, we consider four efficient polynomial-time vote selection strategies: Paired, Max, Greedy, and Complete Tournament strategies. For ease of explanation, we use the graph in Figure 6 as an example. Before executing a vote selection strategy, we assume that each object has been scored by a scoring function in Step 1 of the framework, denoted by $s[\cdot]$ in Algorithm 6. As a running example to explain our strategies, we assume that our PageRank heuristic (Section 2.4) is used as the scoring function in Step 1: object A has score 0.5, objects B and E each have score 0.25, and objects C, D, and F have score 0. Without loss of generality, assume that the final rank order of the objects, before next vote selection, is (A, B, E, C, D, F).

Algorithm 6 General Vote Selection Framework

Require: n objects, vote matrix W, budget b
Ensure: *ans* = predicted maximum object
compute score $s[\cdot]$ for all objects using function s {Step 1}
initialize multiset Q {of votes to request}
sort all objects by $s[\cdot]$, store object indices in $index[\cdot]$
select b votes for Q using a vote selection strategy {Step 2}
submit batch Q
update W with new votes from workers
compute final score $f[\cdot]$ for all objects using function f {Step 3}
$ans \leftarrow \text{argmax}_i \, f[i]$

Strategy 7 Paired Vote Selection

Require: n objects, budget b, vote multiset Q, scores $s[\cdot]$, sorted object indices $index[\cdot]$
Ensure: Q = selected b votes
for $i : 1 \ldots b$ **do**
$Q \leftarrow Q \cup (index[2i-1], index[2i])$ {$index[1]$ has largest score s}
end for

The first strategy we consider is Paired vote selection (PAIR), displayed in Strategy 7. In this strategy, pairs of objects are selected greedily, such that no object is included in more than one of the selected pairs. For example, with a budget of $b = 2$, the strategy asks human workers to compare the rank 1 and rank 2 objects, and the rank 3 and rank 4 objects, where rank is determined by the scoring function from Step 1 in Algorithm 6. The idea behind this strategy is to restrict each object to be involved in at most one of the additional votes, thus distributing the b votes among the largest possible set of objects. This can be anticipated to perform well when there are many objects with similar scores, e.g., when there are many objects in the initial vote graph G_v which have equally high chances of being the maximum object. Considering the example in Figure 6, for $b = 2$, this strategy requests the votes (A, B) and (E, C).

Strategy 8 Max Vote Selection

Require: n objects, budget b, vote multiset Q, scores $s[\cdot]$, sorted object indices $index[\cdot]$
Ensure: Q = selected b votes
for $i : 2 \ldots (b+1)$ **do**
$Q \leftarrow Q \cup (index[1], index[i])$ {$index[1]$ has largest score s}
end for

The second strategy we consider is Max vote selection (MAX), displayed in Strategy 8. In this strategy, human workers are asked

to compare the top-ranked object against other objects greedily. For example, with a budget of $b = 2$, this strategy asks human workers to compare the rank 1 and rank 2 objects, and the rank 1 and rank 3 objects, where rank is determined by the scoring function in Step 1 in Algorithm 6. Considering again the example in Figure 6, for $b = 2$, this strategy requests the votes (A, B) and (A, E).

Strategy 9 Greedy Vote Selection

Require: n objects, budget b, vote multiset Q, scores $s[\cdot]$, sorted object indices $index[\cdot]$
Ensure: Q = selected b votes
  initialize priority queue S {storing unordered object pairs}
  for $i : 1 \ldots b$ **do**
    for $j : (i + 1) \ldots b$ **do**
      insert object pair (i, j) into S with priority $(s[i] \times s[j])$
    end for
  end for
  for $i : 1 \ldots b$ **do**
    Remove highest priority object pair (x, y) from S
    $Q \leftarrow Q \cup (x, y)$
  end for

The third strategy we consider is Greedy vote selection (GREEDY), displayed in Strategy 9. In this strategy, all possible comparisons (unordered object pairs) are weighted by the product of the scores of the two objects, where the scores are determined in Step 1 of Algorithm 6. In other words, a distribution is constructed across all possible object pairs, with higher weights assigned to object pairs involving high scoring objects (which are more likely to be the maximum object in O). After weighting all possible object pairs, this strategy submits the b highest weight pairs for human comparison. Considering the example in Figure 6, object pairs (A, B) and (A, E) has weight 0.125, (B, E) has weight 0.0625, and all other pairs have weight 0. For a budget $b = 2$, this strategy requests the votes (A, B) and (A, E).

Strategy 10 Complete (Round-Robin) Vote Selection

Require: n objects, budget b, vote multiset Q, scores $s[\cdot]$, sorted object indices $index[\cdot]$
Ensure: Q = selected b votes
  $K \leftarrow 0$ {K is the size of the round-robin tournament}
  while $\frac{K*(K+1)}{2} \leq b$ **do**
    $K \leftarrow K + 1$
  end while
  $K \leftarrow K - 1$
  for $i : 1 \ldots K$ **do**
    for $j : (i + 1) \ldots K$ **do**
      $Q \leftarrow Q \cup (index[i], index[j])$ {$index[1]$ has largest score s}
    end for
  end for
  initialize priority queue S {storing unordered object pairs}
  for $i : 1 \ldots K$ **do**
    insert object pair $(i, K + 1)$ into S with priority $(s[i] \times s[K + 1])$
  end for
  for $i : 1 \ldots (b - \frac{K*(K+1)}{2})$ **do**
    Remove highest priority object pair (x, y) from S
    $Q \leftarrow Q \cup (x, y)$ {select remaining votes greedily}
  end for

The fourth strategy we consider is Complete Tournament vote selection (COMPLETE), displayed in Strategy 10. In this strategy,

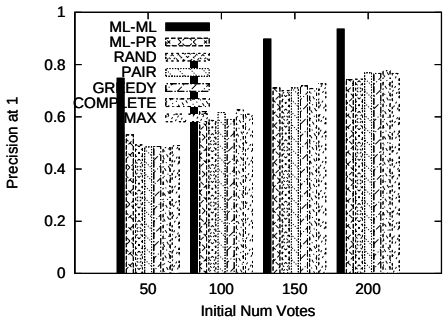


Figure 7: Precision at 1 versus number of initial votes. 1 additional vote, 7 objects, p=0.75.

we construct a single round-robin tournament among the K objects with the highest scores from Step 1 of Algorithm 6, where K is the largest number such that $\frac{K*(K+1)}{2} \leq b$. In a single round-robin tournament, each of the K objects is compared against every other exactly once. For the remaining $r = b - \frac{K*(K+1)}{2}$ votes, we consider all object pairs containing the $(K + 1)st$ (largest scoring) object and one of the first K objects, and weight each of these K object pairs by the product of the scores of the two objects (as we did with Greedy vote selection). We then select the r object pairs with highest weight.

The idea behind the Complete Tournament strategy is that a round-robin tournament will likely determine the largest object among the set of K objects. If the set of K objects contains the true max, this strategy can be anticipated to perform well. Regarding the selection of the remaining votes, the strategy can be thought of as augmenting the K object tournament to become an incomplete $K+1$ object tournament, where the remaining votes are selected greedily to best determine if the $(K+1)st$ object can possibly be the maximum object in O. Considering the example in Figure 6, for $b = 2$, there is a 2-object tournament among objects A and B and vote (A, B) is requested. Then, for the remaining vote, the strategy greedily scores object pairs which contain both the next highest ranked object not in the tournament, object E, and one of the initial 2 objects. Object pair (A, E) will be scored 0.125 and (B, E) will be scored 0.0625, so the second vote requested is (A, E).

3.4 Experiments

Which of our four vote selection heuristics (PAIR, MAX, GREEDY, or COMPLETE) is the best strategy? We now describe a set of experiments measuring the prediction performance of our heuristics for various sets of parameters. When evaluating our vote selection strategies, we utilized a uniform vote sampling procedure, described previously in Section 2.5, to generate an initial vote matrix W. Then, in Step 1 of our vote selection framework (Algorithm 6), we adopted our PageRank heuristic (Section 2.4) as our scoring function $s(\cdot)$ to score each object in O. In Step 2, we executed each of our vote selection strategies using these scores. In Step 3, we used our PageRank heuristic as our scoring function $f(\cdot)$ to score each object in the new matrix W' (composed of both the initial votes in vote matrix W and the b requested additional votes), and generate final predictions for the maximum object in O. Note that we performed several experiments contrasting prediction performance of PageRank versus other possible scoring functions and found PageRank to be superior to the other functions. Hence, we selected PageRank as the scoring function for both Step 1 and Step 3 of our vote selection framework.

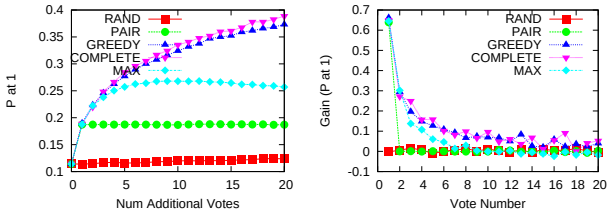


Figure 8: Precision at 1 versus number of additional votes (left). Incremental Gain (P@1) of each vote relative to a 0 additional votes baseline (right). 100 objects, p=0.95, 200 initial votes.

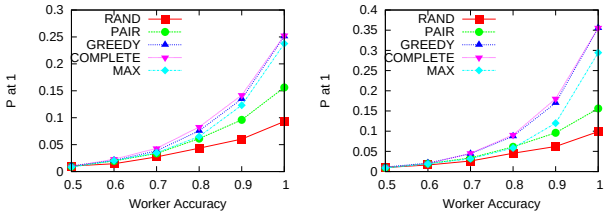


Figure 9: Precision at 1 versus worker accuracy. 100 objects, 100 initial votes. 5 additional votes (left), 15 additional votes (right)

- ML vote selection outperforms heuristic strategies when results are evaluated with ML scoring.
- However, when ML vote selection is evaluated with PageRank (e.g., like the heuristics), prediction performances of all methods are similar.

For a first experiment, we compare the prediction performance (Precision at 1) of our four vote selection heuristics (and random initial vote selection (RAND)) against the "optimal" strategy, i.e., the Maximum Likelihood (ML) vote selection procedure described in Section 3.1. Recall that ML can be used in two places: when selecting additional votes (as in Section 3.1), and when predicting the max given the initial plus additional votes (e.g., ML evaluation in Section 2.2). We use ML-ML to refer to using ML for both tasks, this gives the best possible strategy. To gain additional insights, we also consider ML-PR, a strategy where ML is used to select the additional votes, and PageRank is used to select the winner. Since ML is computationally very expensive, for this experiment we consider a small problem: select *one* additional vote given a set of 50 (2.5x Edge Coverage) to 200 initial votes (10x Edge Coverage) among a set of 7 objects, $p = 0.75$.

Our experimental results are displayed in Figure 7. First, as expected, ML-ML has the best performance. Clearly, ML-ML is doing a better job at selecting the additional vote and in selecting the winner. Of course, keep in mind that ML-ML is not feasible in most scenarios, and it also requires knowledge of the worker accuracy p. Nevertheless, the gap between ML-ML and the other strategies indicates there is potential room for future improvement beyond the heuristics we have developed.

Second, we observe in Figure 7 that all other strategies, including ML-PR, perform similarly. The relative performance of ML-PR indicates that the gain achieved by ML-ML is due to its better prediction of the winner, as opposed to its choice for the next vote. In hindsight, this result is not surprising, since the selection of a single vote cannot be expected to have a large impact. (We will observe larger impacts when we select multiple additional votes.) The results also demonstrate that our vote selection heuristics show promise, since they seem to be doing equally well as ML, and since they often perform slightly better than RAND, at least for the selection of a single next vote.

To evaluate our heuristics in larger scenarios, we conducted a series of experiments, and the results of some of those are summarized in Figures 8, 9 and 10. To begin, we summarize some of the general trends that can be observed in these figures.

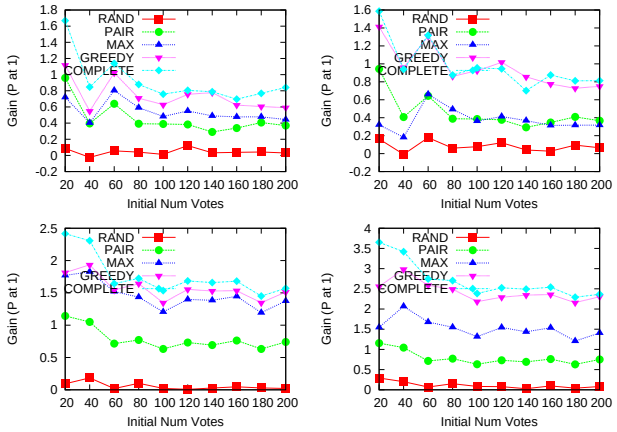


Figure 10: Gain (P@1) relative to a 0 additional votes baseline vs number of initial votes. 100 objects. 5 add. votes and p=0.75 (top left), 15 add. votes and p=0.75 (top right), 5 add. votes and p=0.95 (bottom left), 15 add. votes and p=0.95 (bottom right).

General observations regarding all strategies:

- As the number of additional votes increases, prediction performance increases.
- As the number of additional votes increases, the gain from additional votes decreases (though the decrease is not very dramatic).
- As worker accuracy increases, prediction performance increases.
- As worker accuracy increases, the gain from additional votes increases.

We only explain the graph in Figure 8(right), since the others are self-explanatory. In this graph, the vertical axis shows the incremental P@1 gain at k additional votes, defined as (P@1 with k additional votes - P@1 with $k-1$ additional votes) / (P@1 with 0 additional votes). As we can see, the information provided by additional votes is more valuable when there are fewer initial votes (second bullet above).

The Complete Tournament and Greedy strategies are significantly better than the Max and Paired strategies.

We can also use Figures 8, 9 and 10 to compare our heuristics. First, notice that the difference between heuristics can be very significant. For instance, in Figure 10(bottom left) we see that the Paired (PAIR) strategy provides a 0.7x P@1 gain for 5 additional votes (100 initial votes, $p = 0.95$), while the Complete Tournament (COMPLETE) strategy provides a 1.5x P@1 gain, where we measure P@1 gain as (P@1 with k votes - P@1 with 0 votes) / (P@1 with 0 votes). Second, we observe that the Complete Tournament and Greedy (GREEDY) vote selection strategies consistently outperform the Max (MAX) and Paired strategies in all scenarios. In particular, the performance gap between the Complete Tournament or Greedy strategies and the Max or Paired strategies is greater when selecting 15 additional votes, as compared to when selecting 5 additional votes. This indicates that when a larger vote budget b is available for additional votes, the additional votes will be better utilized by the more sophisticated strategies (Complete Tournament and Greedy) as compared to the simpler strategies (Max and Paired). Also, note in Figure 8(left) that the prediction performances of the Complete Tournament and Greedy strategies steadily improve with additional votes, while the Max and Paired strategies taper off.

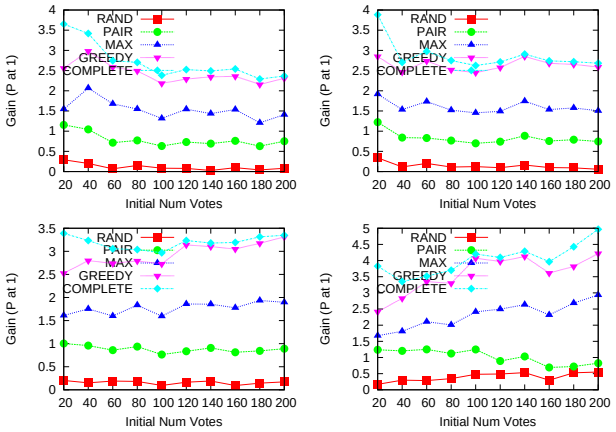


Figure 11: Objects are divided into k initial object types. Gain (P@1) relative to a 0 additional votes baseline vs number of initial votes. 100 objects, p=0.95, 15 additional votes. 1 type (top left), 5 types (top right), 10 types (bottom left), 20 types (bottom right).

Given only votes between objects of the same type:

- The value of additional votes is greater when it is more difficult to predict the maximum object.
- The Complete Tournament strategy is the best strategy.

In our scenarios so far, the Complete Tournament and Greedy strategies perform similarly. To differentiate between the two, we explored different ways in which the initial votes could be generated. (Recall that up to this point we have been randomly selecting the pairs of objects that are compared by the initial votes.) We next discuss one of these possible different vote generation schemes. Suppose that our objects are of different types (e.g., soft-cover books, hardcover books, e-books, etc.), and for some reason initial votes between objects of the same type are much more likely than across types. For example, it is more likely that two e-books have been compared, rather than one e-book and one hard-cover book. (The situation is analogous to sporting events, where intra-league games are more likely than inter-league games.)

For our experiment, we consider an extreme instance where there are *no* initial votes involving objects of different types. In particular, we divide our set O of n objects into k disjoint object types. When votes are sampled for the initial vote matrix W, sampling of votes is only permitted between objects of the same type. Keep in mind that predicting the maximum object in O is more difficult when there are more object types because each object type will likely have a leader (greatest object), each of these leaders will have on average similar probabilities of being the maximum object (since object type groups are likely of similar size), and the initial vote matrix W provides no information regarding comparisons between these leaders.

We perform experiments for different values of k (e.g., different numbers of initial object types), Figure 11 displays Precision at 1 gain relative to a 0 additional votes baseline. We observe that the P@1 gain increases for the Complete Tournament and Greedy strategies as k increases, implying that the value of additional votes is greater when it is more difficult to predict the maximum object from the initial vote matrix. That is, in the harder problem instances (larger k), the additional votes play a more critical role in comparing the object type leaders. More importantly, the Complete Tournament strategy outperforms the Greedy strategy (and the others too) in this more challenging scenario.

Finally, we conduct a more in-depth study of the Complete Tournament vote selection strategy and examine the benefit of vote redundancy. Given a limited budget, should the Complete Tournament strategy select fewer top objects and propose more redundant votes, or should it select more objects and ask fewer votes per pair? For instance, the Complete Tournmament strategy could select the top three objects and submit four votes for each pair, for a total of 12 additional votes. Or it could select the top 4 objects, and for each of the possible 6 comparisons, request 2 votes (for the same 12 total additional votes). What is the best approach?

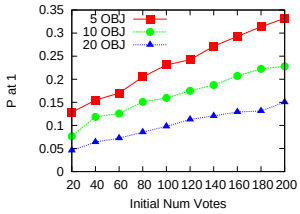


Figure 12: Variations of the Complete Tournament strategy, 10 additional votes. 5 OBJ = compare 5 objects 4 times each. 10 OBJ = compare 10 objects 2 times each. 20 OBJ = compare 20 objects 1 time each. 100 objects, p=0.95.

Figure 12 displays the Precision at 1 of the Complete Tournament strategy for 10 additional votes, where the votes are uniformly and randomly distributed among the 5, 10, or 20 objects in O with highest score (as provided in Step 1 of Algorithm 6). We find that distributing the 10 votes among 5 objects, where each object is compared against every other, leads to the best prediction performance. That is, we do not observe any benefit for distributing votes among a larger set of objects when using the Complete Tournament strategy. The strategy performs well only when additional votes provide the ability to rank the objects in a set. Assuming that votes are distributed randomly among object pairs, the Complete Tournament strategy is able to order the set only when most objects in the set are compared against each other. Note that Figure 12 is only an illustration of the interaction between the number of top objects selected and the redundancy of votes. The results will vary depending upon worker accuracy and the vote budget b.

4. RELATED WORK

As far as we know, we are the first to address the Next Votes Problem, and there is no relevant literature that directly addresses this problem. Thus, in this section, we review work related to the Judgment Problem. The algorithms and heuristics we presented for the Judgment Problem are primarily drawn from three diverse topic areas: paired comparisons, social choice, and ranking.

The Judgment Problem has its roots in the *paired comparisons* problem, first studied by statisticians decades ago [20, 12]. In the paired comparisons problem, given a set of pairwise observations regarding a set of objects, it is desired to obtain a ranking of the objects. In contrast, in the Judgment Problem, we are interested in predicting the maximum object.

The Judgment Problem also draws upon classical work in the economic and social choice literature regarding *Winner Determination* in elections [25, 31]. Numerous voting rules have been used (Borda, Condorcet, Dodgson, etc.) to determine winners in elections [28]. The voting rules most closely related to our work are the Kemeny rule [19] and Slater rule [29]. A *Kemeny permutation* minimizes the total number of pairwise inconsistencies among all votes, whereas a *Slater permutation* minimizes the total number of pairwise inconsistencies in the majority-vote graph [6]. An object is considered a *Kemeny winner* or *Slater winner* if it is the greatest object in a Kemeny permutation or Slater permutation.

We believe our ML formulation is more principled than these voting rules, since ML aggregates information across all possible permutations. For example, in the graph of Figure 1, while C and

D are both admissible solutions for the Kemeny rule, ML returns D as an answer, since D has almost one and a half more times the probability of being the maximum object compared to C. No prior work about the Judgment Problem, to our knowledge, uses the same approach as our ML formulation.

In the recent social choice literature, the research most closely related to ours has been work by Conitzer et al. regarding Kemeny permutations and Maximum Likelihood [9, 10]. Conitzer has studied various voting rules and determined for which of them there exist voter error models where the rules are ML estimators [8]. In our study, we focused upon the opposite question: for a specific voter error model, we presented both Maximum Likelihood, as well as heuristic solutions, to predict the winner.

Our work is also related to research in the theory community regarding ranking in the presence of errors [21, 1] and noisy computation [14, 2]. Both Kenyon et al. and Ailon et al. present randomized polynomial-time algorithms for feedback arc set in tournament graphs. Their algorithms are intended to approximate the optimal permutation, whereas we seek to predict the optimal winner. Feige et al. and Ajtai et al. present algorithms to solve a variety of problems, including the Max Problem, but their scenarios involve different comparison models or error models than ours.

More generally, in the last several years, there has been a significant amount of work regarding crowdsourcing systems, both inside [15, 16] and outside [22, 23] the database community. Of note is recent work by Tamuz et al. [30] on a crowdsourcing system that learns a similarity matrix across objects, while adaptively requesting votes. Not as much work has been done regarding general crowdsourcing algorithms [24, 27]. Instead, most algorithmic work in crowdsourcing has focused upon quality control [3, 18].

5. CONCLUSION

In a conventional database system, finding the maximum element in a set is a relatively simple procedure. It is thus somewhat surprising that in a crowdsourcing database system, finding the maximum is quite challenging, and there are many issues to consider. The main reason for the complexity, as we have seen, is that our underlying comparison operation may give an incorrect answer, or it may even not complete. Thus, we need to decide which is the "most likely" max (Judgment Problem), and which additional votes to request to improve our answer (Next Votes Problem).

Our results show that solving either one of these problems optimally is very hard, but fortunately we have proposed effective heuristics that do well. There is a gap between the optimal solution and what our heuristics find (as seen for example in Figure 7), but we believe that it will be very hard to close this gap without incurring high computational costs. Among the heuristics, we observed significant differences in their predictive ability, indicating that it is very important to carefully select a good heuristic. Our results indicate that in many cases (but not all) our proposed PageRank heuristic is the best for the Judgment Problem, while the Complete Tournament heuristic is the best for the Next Votes Problem.

Our results are based on a relatively simple model where object comparisons are pairwise, and worker errors are independent. Of course, in a real crowdsourced database system these assumptions may not hold. Yet we believe it is important to know that even with the simple model, the optimal strategies for the Judgment Problem and Next Votes Problem are NP-Hard. Furthermore, our heuristics can be used even in more complex scenarios, since they do not depend on the evaluation model. Even though they can be used when the model assumptions do not hold, we believe it is important to understand how the heuristics perform in the more tractable scenario we have considered here.

6. REFERENCES

[1] N. Ailon et al. Aggregating inconsistent information: ranking and clustering. *JACM*, 55(5), 2008.
[2] M. Ajtai et al. Sorting and selection with imprecise comparisons. *Automata, Languages and Programming*, 2009.
[3] O. Alonso et al. Crowdsourcing for relevance evaluation. In *SIGIR Forum*, 2008.
[4] P. Boldi et al. PageRank as a function of the damping factor. In *WWW*, 2005.
[5] F. Brandt et al. PageRank as a weak tournament solution. In *WINE*, 2007.
[6] Y. Chevaleyre et al. A short introduction to computational social choice. In *SOFSEM*, 2007.
[7] W. Cohen et al. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.
[8] V. Conitzer et al. Common voting rules as maximum likelihood estimators. In *UAI*, 2005.
[9] V. Conitzer et al. Improved bounds for computing kemeny rankings. In *AAAI*, 2006.
[10] V. Conitzer et al. Preference functions that score rankings and maximum likelihood estimation. In *IJCAI*, 2009.
[11] D. Coppersmith et al. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *SODA*, 2006.
[12] H.A. David. The method of paired comparisons. 1963.
[13] H.A. David. Ranking from unbalanced paired-comparison data. *Biometrika*, 74(2):432–436, 1987.
[14] U. Feige et al. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.
[15] M. Franklin et al. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.
[16] P. Heymann et al. Turkalytics: analytics for human computation. In *WWW*, 2011.
[17] O. Hudry. On the complexity of Slater's problems. *European Journal of Operational Research*, 203(1):216–221, 2010.
[18] P. Ipeirotis et al. Quality management on amazon mechanical turk. In *KDD Workshop on Human Computation*, 2010.
[19] J. Kemeny. Mathematics without numbers. *Daedalus*, 1959.
[20] M.G. Kendall et al. On the method of paired comparisons. *Biometrika*, 31(3/4):324–345, 1940.
[21] C. Kenyon-Mathieu et al. How to rank with few errors. In *STOC*, 2007.
[22] A. Kittur et al. Crowdsourcing user studies with mechanical turk. In *CHI*, 2008.
[23] G. Little et al. TurKit: human computation algorithms on mechanical turk. In *UIST*, 2010.
[24] A. Marcus et al. Human-powered sorts and joins. *PVLDB*, 2011.
[25] H. Moulin. Choosing from a tournament. *Social Choice and Welfare*, 3(4):271–291, 1986.
[26] L. Page et al. The PageRank citation ranking: Bringing order to the web. *Technical Report*, 1998.
[27] A. Parameswaran et al. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 2011.
[28] D. Saari. Basic geometry of voting. 1995.
[29] P. Slater. Inconsistencies in a schedule of paired comparisons. *Biometrika*, 48(3/4):303–312, 1961.
[30] O. Tamuz et al. Adaptively learning the crowd kernel. In *ICML*, 2011.
[31] P. Young. Optimal voting rules. *J. Econ. Perspectives*, 1995.