

Max Algorithms in Crowdsourcing Environments

Petros Venetis	Hector Garcia-Molina	Kerui Huang	Neoklis Polyzotis
Stanford University	Stanford University	UC Santa Cruz	UC Santa Cruz
venetis@cs.stanford.edu	hector@cs.stanford.edu	khuang@cs.ucsc.edu	alkis@cs.ucsc.edu

Abstract

Our work investigates the problem of retrieving the maximum item from a set in crowdsourcing environments. We first develop parameterized families of max algorithms, that take as input a set of items and output an item from the set that is believed to be the maximum. Such max algorithms could, for instance, select the best Facebook profile that matches a given person or the best photo that describes a given restaurant. Then, we propose strategies that select appropriate max algorithm parameters. Our framework supports various human error and cost models and we consider many of them for our experiments. We evaluate under many metrics, both analytically and via simulations, the tradeoff between three quantities: (1) quality, (2) monetary cost, and (3) execution time. Also, we provide insights on the effectiveness of the strategies in selecting appropriate max algorithm parameters and guidelines for choosing max algorithms and strategies for each application.

1 Introduction

Humans are more effective than computers for many tasks, such as identifying concepts in images, translating natural language, and evaluating the usefulness of products. Thus, there has been a lot of recent interest in *crowdsourcing*, where humans perform tasks for pay or for fun [1, 22].

Crowdsourced applications often require substantial programming: A computer must post tasks for the humans, collect results, and cleanse and aggregate the answers provided by humans. In many cases, a computer program is required to oversee the entire process. For instance, say we have 1,000 photos of a restaurant, and we wish to find the one that people think is most appealing and best describes the restaurant. (We wish to use the photo in an advertisement, for example.) Since it is not practical to show one human all 1,000 photos, we need to break up the problem. For example, as a first step, we may show 100 people 10 photos each, and ask each one of them to select the best one in their set. Then we can select the 100 winners, partition them into 10 new sets, and ask 10 humans to evaluate, and so on. What we have described is a *crowdsourcing algorithm*, analogous to traditional algorithms, except that the base operations or comparisons (e.g., compare 10 photos) are done by humans as opposed to by the computer. Note that we use the term *program* to refer to the computer code that runs the entire process (e.g., finding human workers, displaying photos to the workers, paying workers, and so on). We reserve the term *algorithm* to refer to the logic that performs a specific function of interest (e.g., finding the max).

In this paper we study an important crowdsourcing algorithm, one that finds the best or maximum item in a set, just as illustrated in the previous example. This max algorithm is a fundamental one, needed in many applications. For instance, the max algorithm can be used to find the most relevant URL for a given user query. (The results can subsequently be used as a gold standard for evaluating search engine results.) As another example, a max algorithm can be used in an entity resolution application, e.g., to find the best Facebook profile that matches a target person.

The main challenge in developing an effective max algorithm is in handling user mistakes or variability. Specifically, humans may give different answers in a comparison task (i.e., pick a different item as the best), either because they make a mistake and pick the wrong item, or because the comparison task is subjective and there is no true best item. We can compensate by asking multiple humans to perform each

task (comparison in our setting), but these extra tasks add cost and/or latency. In this paper we address these quality-cost-latency tradeoffs and explore different types of max algorithms, under different evaluation criteria.

There is a substantial body of work in the theory community that explores algorithms with “faulty” comparisons (e.g., [2, 3, 4, 7, 8, 9, 19, 23]), and that work is clearly applicable here. However, those works use relatively simple models that often do not capture all of the crowdsourcing issues (such as the monetary cost required to execute a crowdsourcing algorithm). For example, these works assume that operators (in our case humans) compare only two items at a time, or have a fixed probability of making an error. It is not obvious how to extend the proposed techniques to the common case in crowdsourcing where humans compare more than two items at a time and the error is related to how similar the items are.

At the other extreme, there is recent work [17] that implements algorithms like max in a real crowdsourcing system and considers all practical issues. Their algorithm is highly specialized to a specific crowdsourcing scenario and it is not obvious how to generalize it to handle other scenarios, e.g., different types of item comparisons or different types of human workers.

For our work, we have taken a middle-of-the-ground approach: We formulate a fairly detailed framework that captures, for instance, human tasks that involve sets of items of different sizes, and different human error models. Within this framework, we are able to study, not just a single algorithm or approach, but a wide range of families of parameterized max algorithms, and we are able to study how to tune these algorithms for best performance. Instead of just asking the question “For one given error model (or for one particular crowdsourcing system), what algorithm is best,” we ask the more general question “For what type of workers and tasks does each possible family of max algorithm work best.” Instead of just asking the question “How does algorithm X perform when humans can compare two items,” we ask “What is the impact on cost and answer quality of the number of items a human is given to compare each time.” Instead of just showing that “Algorithm X works well in a given setting,” we show how to tune each family of max algorithms (e.g., how many repetitions of each tasks are needed at each stage of the process). To answer these questions we really need a model that is relatively rich and detailed, but yet not as rigid as an actual implementation.

As we will see, for some of our studies we resort to simulations, since the models are not amenable to detailed analysis. However, in some of our scenarios, we are able to develop analytical expressions that describe performance. Since both our simulations and analysis share the same underlying framework, when we can obtain analytical expressions, we are also able to validate them with simulations. This redundancy gives us greater confidence in both the simulations and the analysis.

Contributions In summary, our contributions are the following:

- We develop a framework for reasoning about max algorithms (Section 2).
- A max algorithm often asks multiple workers to perform the same comparison (task), so it needs a *rule* to aggregate the responses. We formalize the most popular aggregation rule and explore some of its key properties (Section 3.1).
- We present several families of parameterized max algorithms. The parameters for each family include the number of items compared by each human task, and the number of repetitions for each comparison (Sections 3.2 and 3.3). Note that the parameters may vary during an execution, e.g., we may initially perform fewer repetitions and in later stages perform more.
- We study *strategies* for tuning a max algorithm, i.e., for finding the parameter values that lead to lowest overall error, given appropriate constraints (Section 4).
- As part of our framework, we develop a number of models for worker error and for user cost/payments (Sections 5.1 and 5.2).
- We experimentally evaluate our algorithms and strategies under various error and cost models (Section 6).

2 Preliminaries

Our problem’s input is a set of items $\mathcal{E} = \{e_1, e_2, \dots, e_{|\mathcal{E}|}\}$. There is a *smaller than* relation $<$, which forms a strict ordering for the items in \mathcal{E} . That is, for any two items $e_i, e_j \in \mathcal{E}$ (with $i \neq j$) exactly one of the following is true: either $e_i < e_j$ or $e_j < e_i$. (We assume that $e_i \neq e_j$ for $i \neq j$.) The item $e_i \in \mathcal{E}$ for which $e_j < e_i, \forall j \in \{1, 2, \dots, |\mathcal{E}|\} \setminus \{i\}$ is called the *maximum item* in \mathcal{E} . Our goal is to recover the maximum item in \mathcal{E} , given a set of constraints (our problem is formally defined in Section 2.4). (Note that the $<$ relation is only a formalism used to quantify effectiveness. The max algorithms themselves do not use the $<$ relation.)

2.1 Comparisons

To determine the maximum item, an algorithm uses a *comparison operator* termed $\text{Comp}()$. Operator $\text{Comp}(S, r, \mathbf{R})$ asks r humans to compare the items in set $S \subseteq \mathcal{E}$ and combines the responses using aggregation rule \mathbf{R} . (One simple rule \mathbf{R} for example is to pick the item with the most votes, described in detail in Section 3.1.) Each of the r workers selects the item from S that he believes to be the maximum and \mathbf{R} provides the overall winner.

We use a probabilistic model to describe a worker. In particular, our model gives a vector of probabilities $\vec{p} = [p_1, p_2, \dots, p_{|S|}]$, where p_1 is the probability that the worker returns the maximum item, p_2 is the probability he returns the second best, and so on. For example, if $\vec{p} = [0.8, 0.1, 0.1]$, then a worker selects the maximum item in S with probability 0.8, and each of the non-maximum items with 0.1 probability. We study other error models in Section 5.1.

In the crowdsourcing platforms we examine, human work is compensated. We denote the compensation of any human performing a comparison of $|S|$ items by $\text{Cost}(|S|)$. We consider various types of cost functions in Section 5.2. The total cost of $\text{Comp}(S, r, \mathbf{R})$ is the sum of its compensations, which is equal to $r \cdot \text{Cost}(|S|)$.

2.2 Steps

Crowdsourcing algorithms are executed in *steps*. A batch of comparisons \mathcal{C}_1 is submitted during the first step to the marketplace and after some time the human answers are returned. Then, depending on these answers, an appropriate set \mathcal{C}_2 of comparisons is selected for the second step. For the selection of the \mathcal{C}_i comparisons all human answers from previous steps (1, 2, \dots , $i - 1$) can be considered.

Since the operations in a step are likely to be executed concurrently, the number of steps an algorithm requires is a(n) indicator/proxy of the execution time of the algorithm. The number of steps is an especially good indicator when the number of human workers is very large. Thus, we want the execution time proxy (number of steps) to be kept to a minimum.

2.3 Max Algorithms

Our work focuses on *parameterized families of max algorithms*, although many more algorithms can be considered. The input of a *max algorithm* is a set of items \mathcal{E} and the output is an item from \mathcal{E} that is believed to be the maximum in \mathcal{E} . To obtain a max algorithm A from a parameterized family of max algorithms \mathcal{A} , one needs to instantiate a number of parameters: $A = \mathcal{A}(\text{parameters})$. We examine in detail two intuitive families of max algorithm in Section 3.

Example 1 *An example of a max algorithm A is the following: The input for the max algorithm is the set of items $\mathcal{E}_1 = \mathcal{E}$. In step 1, \mathcal{C}_1 contains comparisons $\text{Comp}(S, r_1, \mathbf{R})$: \mathcal{E}_1 is split into non-overlapping sets S of size s_1 (one set can have fewer items), and each set is given to r_1 humans. Each $\text{Comp}()$ (using \mathbf{R}) returns one item (winner). The winners from $\text{Comp}() \in \mathcal{C}_1$ form set \mathcal{E}_2 and are propagated to step 2. In step 2, \mathcal{C}_2 contains comparisons $\text{Comp}(S, r_2, \mathbf{R})$: \mathcal{E}_2 is split into non-overlapping sets S of size s_2 , and each set is given to r_2 humans. The winners from $\text{Comp}() \in \mathcal{C}_2$ form set \mathcal{E}_3 and are propagated to the third step, and so on. At some step, there will only be one winner, which is the output of A .*

As we see in Section 3, A belongs to the family of tournament max algorithms, denoted by \mathcal{A}_T . The parameters for \mathcal{A}_T are the sizes of the comparison sets (s_i) and the number of responses sought by humans (r_i) at each step i . For each pair of sequences $(\{r_i\}, \{s_i\})$, we obtain a different max algorithm $A = \mathcal{A}_T(\{r_i\}, \{s_i\})$.

2.4 Problem Definition

The quantities we are interested in are:

Time(A, \mathcal{E}), the Execution Time: The number of steps required for a max algorithm A to complete (and return one item) for input \mathcal{E} .

Cost(A, \mathcal{E}), the (Monetary) Cost: The total amount of money required for A to complete:

$$\sum_{i=1}^{\text{Time}(A, \mathcal{E})} \sum_{\text{Comp}(S, r, R) \in \mathcal{C}_i} [r \times \text{Cost}(|S|)].$$

Quality(A, \mathcal{E}), the Quality of the Result: The probability that max algorithm A returns the maximum item from input \mathcal{E} .

Given a family of algorithms \mathcal{A} , our goal is to find parameters that “optimize performance.” The optimization can be formulated in a variety of ways. Here, we focus on maximizing the quality of the result, for a given cost budget B and a given time bound T . The human models are an input to the problem, and thus, for example, Quality(A, \mathcal{E}) and Cost(A, \mathcal{E}) take into account how humans act and how they are compensated.

Problem 1 (Max Algorithm)

$$\begin{aligned} & \underset{A=\mathcal{A}(\text{parameters})}{\text{maximize}} && \text{Quality}(A, \mathcal{E}) \\ & \text{subject to} && \text{Cost}(A, \mathcal{E}) \leq B \\ & && \text{Time}(A, \mathcal{E}) \leq T \end{aligned} \tag{1}$$

Other possible formulations of the problem are:

- Given a desired quality of the result and a budget, optimize the execution time, or
- Given a desired quality of the result and an execution time bound, optimize the (monetary) cost.

The strategies we propose in Section 4 can be adjusted to address these problems, but we do not consider them in our current work.

Although we define our optimization problem using the Quality(A, \mathcal{E}) metric, we propose alternative metrics in Section 5.4. These metrics describe how close the returned item is to the maximum item in \mathcal{E} , while Quality(A, \mathcal{E}) describes whether A returns the maximum item in \mathcal{E} or not. For our evaluations (Section 6) we use both Quality() and the additional metrics.

3 Parameterized Families of Max Algorithms

This section explores the two most natural families of parameterized max algorithms that can be used to solve the max algorithm problem (Problem 1). The parameterized families of max algorithms we propose operate in steps and their parameters are the following:

r_i : the number of human responses sought at step i , and

s_i : the size of the sets compared by Comp() at step i (one set can have fewer items). We assume a human cannot compare more than m items, thus $s_i \in \{2, 3, \dots, m\}$.

We refer to the r_i and s_i values used by one algorithm by $\{r_i\}$ and $\{s_i\}$ respectively.

First, we present an intuitive human responses aggregation rule (Section 3.1) that is a key component of every max algorithm. Then, we describe the bubble max algorithms (Section 3.2) and the tournament max algorithms (Section 3.3). Finally, we present the notion of repetitive algorithms (Section 3.4).

3.1 Plurality Rule

This section describes the most common human responses aggregation rule: the *plurality rule* (denoted by \mathbf{R}_P). In Appendix A we present another popular rule (the majority rule) and compare it to the plurality rule.

When comparison $\text{Comp}(S, r, \mathbf{R}_P)$ is performed, one of the items in S with the most votes is selected: If l items have received n responses each and there is no other item with more responses, then one of the l items (selected at random) is considered to be the maximum item in S . The l items are called *winners*.

Aggregation rules are important because they impact $\text{Quality}(A, \mathcal{E})$. In order to understand their impact, we define $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R})$:

Definition 1 $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R})$ is the probability that \mathbf{R} returns the maximum of s items, assuming we collect r human responses, and each response follows a probabilistic distribution \vec{p} .

To calculate $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_P)$ (for $\mathbf{R} = \mathbf{R}_P$), we assume $S = \{e_1, e_2, \dots, e_{|S|}\}$, with $|S| = s$ and $e_s < e_{s-1} < \dots < e_1$. Thus, if $\vec{p} = [p_1, p_2, \dots, p_s]$, then p_i is the probability that a human response selected item e_i as the maximum in S . We note that the number of received human responses for items e_1, e_2, \dots, e_s follows a multinomial distribution with parameter \vec{p} . Thus, since r votes are provided, the probability that k_i votes have been issued for item e_i is $\frac{r!}{k_1! \cdot k_2! \cdot \dots \cdot k_s!} \cdot \prod_{i=1}^s (p_i^{k_i})$. Based on these probabilities, we can compute the overall quality of the aggregation rule as follows:

$$\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_P) = \sum_{l=1}^s \frac{1}{l} \cdot \sum_{n=1}^r \sum_{L \in \mathcal{L}} \sum_{\substack{0 \leq k_i \leq n-1, i \in \bar{L} \\ \sum_{i \in \bar{L}} k_i + l \cdot n = r}} \left[\frac{r!}{(n!)^l \cdot \prod_{j \in \bar{L}} k_j!} \cdot \prod_{z \in L} p_z^n \cdot \prod_{w \in \bar{L}} p_w^{k_w} \right] \quad (2)$$

where \mathcal{L} is the set of all sets L that are subsets of $\{1, 2, \dots, s\}$ with $|L| = l$ and $1 \in L$. Also, $\bar{L} = \{1, 2, \dots, s\} \setminus L$. (The derivation can be found in Appendix A.2). In the following, $\text{Comp}(S, r)$ is considered equivalent to $\text{Comp}(S, r, \mathbf{R}_P)$.

3.1.1 Non-Decreasing Property

To select appropriate max algorithm parameters, we need $\text{AggrQuality}()$ to be non-decreasing on r , i.e., asking more workers cannot worsen our chances of getting the true maximum (as we will see in Section 4). Fortunately, this property holds in all known cases, although there is no formal proof:

Conjecture 1 $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_P)$ is non-decreasing on r .

In 1785, de Condorcet [6] proved that the conjecture is true in the special case where $s = 2$, r is odd, and $p_1 > 0.5$. In the same work, de Condorcet points out that it would be interesting to have a proof for all s and r values. Only a few years back (in 2001), List and Goodin [16] proved two related theorems in the political philosophy field:

1. $\lim_{r \rightarrow \infty} \text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_P) = 1$, when $p_1 > p_i$ for all $i = 2, 3, \dots, s$, i.e., the rule returns the true maximum item when we ask an infinite number of workers.
2. The plurality aggregation rule returns the most-likely item (e_1) with probability higher than any other item.

While Conjecture 1 has not been proved yet, it is common belief that it is true [16]. We discuss the implications of this conjecture in Section 4.

Algorithm 1: $\mathcal{A}_B(\{r_i\}, \{s_i\})$ operating on \mathcal{E}

```

1 if  $\mathcal{E} == \{e\}$  then
2   return  $e$ 
3  $S'_1 \leftarrow$  random subset of  $\mathcal{E}$  of size  $\min(s_1, |\mathcal{E}|)$ ;
4  $\mathcal{E} \leftarrow \mathcal{E} \setminus S'_1$ ;
   //  $w$  is the winner of the last comparison performed
5  $w \leftarrow \text{Comp}(S'_1, r_1)$ ;
6  $i \leftarrow 2$ ;
7 while  $\mathcal{E} \neq \emptyset$  do
8    $S_i \leftarrow$  random subset of  $\mathcal{E}$  of size  $\min(s_i - 1, |\mathcal{E}|)$ ;
9    $\mathcal{E} \leftarrow \mathcal{E} \setminus S_i$ ;
10   $S'_i \leftarrow S_i \cup \{w\}$ ;
11   $w \leftarrow \text{Comp}(S'_i, r_i, \mathbf{R})$ ;
12   $i \leftarrow i + 1$ ;
13 return  $w$ 

```

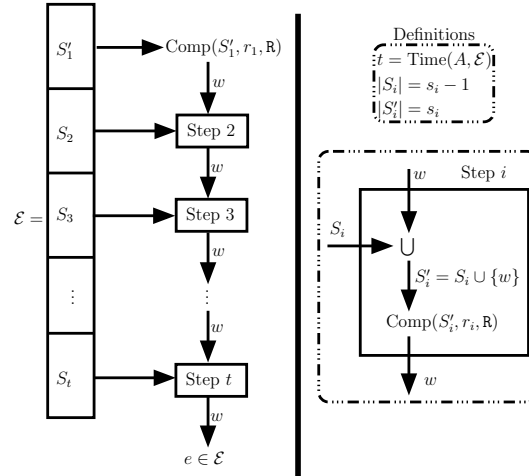


Figure 1: Graphical depiction of the bubble algorithm.

3.2 Bubble Max Algorithms

The family of Bubble max algorithms (denoted by \mathcal{A}_B) is named after the bubble-sort algorithm, because of its similarities to it. Algorithm $A = \mathcal{A}_B(\{r_i\}, \{s_i\})$ is described in Algorithm 1 and depicted in Figure 1.

In summary, A operates on \mathcal{E} as follows. If \mathcal{E} only has one item, this item is returned by the max algorithm A . Otherwise, a comparison with s_1 random items from \mathcal{E} is performed and a winner w is declared. The s_1 random items are removed from \mathcal{E} . If \mathcal{E} is now empty, w is returned. Otherwise, in every step i , $s_i - 1$ items from \mathcal{E} are chosen and unioned with $\{w\}$ forming S'_i . $\text{Comp}(S'_i, r_i, \mathbf{R})$ is “executed” and a winner w is declared, while the items of S'_i are removed from \mathcal{E} . When \mathcal{E} becomes empty, the last winner w is returned by the max algorithm.

To formally define the max algorithm problem (Problem 1) we introduce a few new variables in this section: x_i (the size of the output of step i), \underline{s}_i (the size of the smallest set that is compared at step i), and \bar{x}_i (the number of items that are compared within a set of size s_i at step i).

Example 2 For example, in step i , we have an input of $x_{i-1} = 12$ items and $s_i = 5$. In this case, we will have 2 sets of size $s_i = 5$ compared. Thus, $\bar{x}_i = 10$ items are compared within sets of size s_i . The remaining

$x_{i-1} - \bar{x}_i = \underline{s}_i = 2$ items are included in a smaller set of size \underline{s}_i to be compared.

Now we are able to express the following three quantities:

Time(A, \mathcal{E}): the smallest k for which $|\mathcal{E}| + k - \sum_{i=1}^k s_i \leq 1$.

Cost(A, \mathcal{E}): $\sum_{i=1}^{\text{Time}(A, \mathcal{E})} (r_i \times \text{Cost}(\underline{s}_i))$.

Quality(A, \mathcal{E}): $\frac{s_1}{x_0} \cdot \prod_{i=1}^{\text{Time}(A, \mathcal{E})} \text{AggrQuality}(\underline{s}_i, r_i; \vec{p}_i, \mathbf{R}) + \sum_{j=2}^{\text{Time}(A, \mathcal{E})} \frac{s_j - 1}{x_0} \prod_{i=j}^{\text{Time}(A, \mathcal{E})} \text{AggrQuality}(\underline{s}_i, r_i; \vec{p}_i, \mathbf{R})$.

For this family of max algorithms Quality(A, \mathcal{E}) is the summation of the probabilities of the maximum item taking part in the i^{th} comparison for the first time, multiplied by the probability that it is propagated to all next steps.

Our goal is to select the appropriate sequences $\{r_i\}$ and $\{s_i\}$ in order to maximize Quality(), given constraints on Cost() and Time(). The optimization problem described in Equation (1) becomes for the bubble max algorithms:

$$\begin{aligned}
& \underset{A=\mathcal{A}(\{r_i\}, \{s_i\})}{\text{maximize}} && \frac{\underline{s}_1}{x_0} \cdot \prod_{i=1}^{\text{Time}(A, \mathcal{E})} \text{AggrQuality}(\underline{s}_i, r_i; \vec{p}_i, \mathbf{R}) + \sum_{j=2}^{\text{Time}(A, \mathcal{E})} \frac{s_j - 1}{x_0} \times \prod_{i=j}^{\text{Time}(A, \mathcal{E})} \text{AggrQuality}(\underline{s}_i, r_i; \vec{p}_i, \mathbf{R}) \\
& \text{subject to} && x_0 = |\mathcal{E}| \\
& && x_{j+1} = \max(1, x_j - (s_j - 1)) \\
& && \underline{s}_j = \min(s_j, x_{j-1}) \\
& && x_{\text{Time}(A, \mathcal{E})} = 1 \\
& && \sum_{i=1}^{\text{Time}(A, \mathcal{E})} (r_i \cdot \text{Cost}(\underline{s}_i)) \leq B \\
& && \text{Time}(A, \mathcal{E}) \leq T \\
& && s_i \in \{2, 3, \dots, m\}
\end{aligned} \tag{3}$$

3.3 Tournament Max Algorithms

The Tournament max algorithms (\mathcal{A}_T) are named after tournament-like algorithms that declare best teams in sports. Given the parameters described at the beginning of this section (sequences $\{r_i\}$ and $\{s_i\}$), we obtain a max algorithm $A = \mathcal{A}_T(\{r_i\}, \{s_i\})$. A operates on some input \mathcal{E} and returns some item from \mathcal{E} , as described in Algorithm 2 and graphically depicted in Figure 2.

Algorithm 2: $\mathcal{A}_T(\{r_i\}, \{s_i\})$ operating on \mathcal{E}

```

1  $i \leftarrow 1$  ;
2  $\mathcal{E}_i \leftarrow \mathcal{E}$  ;
3 while  $|\mathcal{E}_i| \neq 1$  do
4   partition  $\mathcal{E}_i$  in non-overlapping sets  $S_j$ , with  $|S_j| = s_i$  (the last set can have fewer items);
5    $i \leftarrow i + 1$  ;
6    $\mathcal{E}_i \leftarrow \bigcup_j \{C(S_j, r_i, \mathbf{R})\}$  ;
7 return  $e \in \mathcal{E}_i$ 

```

In summary, A operates on \mathcal{E} as follows. Set \mathcal{E}_i contains the items that are compared at step i ($\mathcal{E}_1 = \mathcal{E}$). At step i , \mathcal{E}_i is partitioned in non-overlapping sets (S_j , for $j = 1, 2, \dots, \lceil \frac{|\mathcal{E}_i|}{s_i} \rceil$) of size s_i ($|S_j| = s_i$) and

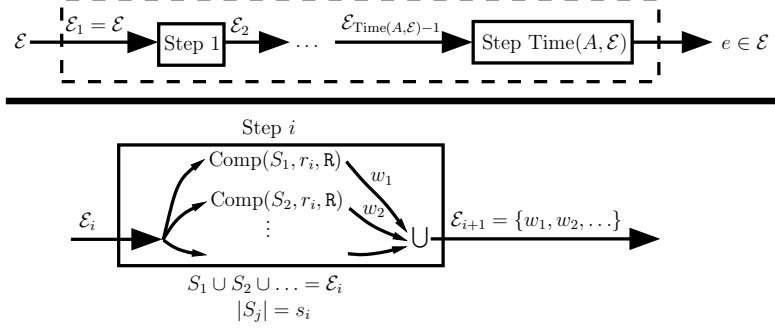


Figure 2: Graphical depiction of the tournament algorithm.

the winners of each comparison $\text{Comp}(S_j, r_i, \mathbf{R})$ form set \mathcal{E}_{i+1} . If \mathcal{E}_{i+1} has exactly one item, this item is the output of A . Otherwise, the process is repeated for one more step.

To formally define the max algorithm problem (Problem 1) we need to be able to express three quantities:

$\text{Time}(A, \mathcal{E})$: the minimum k for which $x_k = 1$ (see Equation (4)).

$\text{Cost}(A, \mathcal{E})$: $\sum_{j=1}^{\text{Time}(A, \mathcal{E})} r_i \cdot \left(\lfloor \frac{x_{j-1}}{s_j} \rfloor \times \text{Cost}(s_j) + \text{Cost}(\underline{s}_j) \right)$.

$\text{Quality}(A, \mathcal{E})$: $\prod_{j=1}^{\text{Time}(A, \mathcal{E})} \left(\frac{\bar{x}_j}{x_{j-1}} \text{AggrQuality}(s_j, r_j; \vec{p}_j, \mathbf{R}) + \frac{\underline{s}_j}{x_{j-1}} \text{AggrQuality}(\underline{s}_j, r_j; \vec{p}_j, \mathbf{R}) \right)$, which is the product of the probabilities that the maximum item in each step is propagated to the next step.

We again aim to select appropriate sequences $\{r_i\}$ and $\{s_i\}$ in order to maximize $\text{Quality}()$, given constraints on $\text{Cost}()$ and $\text{Time}()$. The optimization problem described in Equation (1) becomes for the tournament max algorithms:

$$\begin{aligned}
& \underset{A=\mathcal{A}_T(\{r_i\}, \{s_i\})}{\text{maximize}} && \prod_{j=1}^{\text{Time}(A, \mathcal{E})} \left(\frac{\bar{x}_j}{x_{j-1}} \text{AggrQuality}(s_j, r_j; \vec{p}_j, \mathbf{R}) + \frac{\underline{s}_j}{x_{j-1}} \text{AggrQuality}(\underline{s}_j, r_j; \vec{p}_j, \mathbf{R}) \right) \\
& \text{subject to} && x_0 = |\mathcal{E}| \\
& && x_{j+1} = \lceil \frac{x_j}{s_j} \rceil \\
& && \underline{s}_j = x_{j-1} - \lceil \frac{x_{j-1}}{s_j} \rceil \times s_j \\
& && \bar{x}_j = x_{j-1} - \underline{s}_j \\
& && x_{\text{Time}(A, \mathcal{E})} = 1 \\
& && \sum_{j=1}^{\text{Time}(A, \mathcal{E})} \left(\lfloor \frac{x_{j-1}}{s_j} \rfloor \cdot \text{Cost}(s_j) + \text{Cost}(\underline{s}_j) \right) \cdot r_i \leq B \\
& && \text{Time}(A, \mathcal{E}) \leq T \\
& && s_j \in \{2, 3, \dots, m\}
\end{aligned} \tag{4}$$

3.4 Repetitive Max Algorithms

One natural question is the following: Is it beneficial to apply a cheaper max algorithm on an input more times and aggregate the results of these applications, or should budget be allocated to *one* (more) expensive max algorithm? In order to answer this question, we define *repetitive max algorithms*:

Definition 2 (Repetitive Max Algorithm) *A repetitive max algorithm is a max algorithm A that is applied multiple times on some input set of items \mathcal{E} . The output of the repetitive max algorithm is the item $e \in \mathcal{E}$ that is returned the most times by the multiple applications of A on \mathcal{E} (ties are broken arbitrarily).*

We investigate whether repetitive max algorithms are useful in Section 6.4.

4 Strategies for Tuning Max Algorithms

This section describes strategies that optimize max algorithms with parameters $\{r_i\}$ and $\{s_i\}$ that run in steps (like the ones we described in Section 3). Our strategies are not guaranteed to be optimal, but they heuristically determine parameter choices ($\{r_i\}$ and $\{s_i\}$) that improve performance while satisfying the (budget and time) constraints. We present the strategies by increasing complexity.

ConstantSequences

The first strategy we examine is the one most commonly used in practice today: The practitioner decides how big the sets of items are and how many human responses he seeks per set of items. **ConstantSequences** emulates this behaviour and selects parameters $\{r_i\}$ and $\{s_i\}$ among the constant sequences search space: For all i we have $r_i = r$ and $s_i = s$. This way, we only have two parameters (r and s) to determine.

Algorithm 3: `ConstantSequences($x_0, \vec{p}, B, T, \mathcal{A}$)`

```

//  $\hat{p}$ ,  $\hat{r}$ , and  $\hat{s}$  is the best quality of result, best  $r$ , and best  $s$  seen so far for
  constant sequences
1 ( $\hat{p}, \hat{r}, \hat{s}$ )  $\leftarrow$  (0.0, NULL, NULL);
2 for  $s \leftarrow 2$  to  $m$  do
3    $s_i \leftarrow s, \forall i$ ;
4    $r \leftarrow$  maximum possible repetitions for the selected  $s$  and  $B$ ;
5    $r_i \leftarrow r, \forall i$ ;
6   if  $\text{Time}(\mathcal{A}(\{r_i\}, \{s_i\}), \mathcal{E}) > T$  then
7     continue;
8    $p \leftarrow \text{Quality}(\mathcal{A}(\{r_i\}, \{s_i\}), \mathcal{E})$ ;
9   if  $p > \hat{p}$  then
10    ( $\hat{p}, \hat{r}, \hat{s}$ )  $\leftarrow$  ( $p, r, s$ );
11 return ( $\hat{p}, \hat{r}, \hat{s}$ );

```

ConstantSequences operates as described in Algorithm 3. It goes through all acceptable s values (from 2 to m — line 2), finds the maximum possible r for each s that keeps the cost below the budget B (line 4), and computes `Quality()` (line 8). The execution time constraint is considered in lines 6–7. Finally, **ConstantSequences** returns the “optimal” probability of returning the maximum item and the r and s that give this “optimal” probability $(\hat{p}, \hat{r}, \hat{s})$ (line 11).

Proposition 1 *If $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R})$ is non-decreasing on r , Algorithm 3 returns the optimal selection of r and s for constant sequences (for the bubble and tournament max algorithms).*

Proof: **ConstantSequences** explores all possible s values (line 2). Thus, we only need to prove that we make the appropriate selections of r values for each s we examine.

For both the bubble and the tournament max algorithms, $\text{Quality}(A, \mathcal{E})$ is increasing over each of the values of $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R})$ that come in play in the calculations: In both cases we have (summations of) multiplications of positive quantities AQ . The positive quantities AQ ’s are directly related to the $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R})$ values. Since $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R})$ is non-decreasing on r , AQ ’s are non-decreasing

on r . Thus, $\text{Quality}(A, \mathcal{E})$ is non-decreasing as r increases, and thus it is enough to only examine that highest possible value of r for each s . \square

As mentioned in Section 3.1.1 we do not know whether $\text{AggrQuality}()$ is non-decreasing on r or not, but it is a common assumption in the political philosophy literature [16]. Thus, we are confident that Algorithm 3 returns the optimal r and s values.

RandomHillclimb

RandomHillclimb builds on top of **ConstantSequences** and operates as described in Algorithm 4. It first determines the optimal r and s values using **ConstantSequences** (lines 1–3); the s value does not change after this point for any step. Then, it attempts to adjust the r values across the steps, such that $\text{Quality}()$ increases (lines 4–18). A random *source* step c (line 5) is selected and r_c (repetitions at step c) is decreased by 1. Thus, some additional comparisons can be applied in other steps. We check each possible other step t (line 7) and we attempt to add all the additional comparisons to this *target* step. If by “moving” the repetitions from c to t we manage to improve the quality of the tournament we adjust the sequence $\{r_i\}$ (line 15–17). We perform this greedily, i.e., we make the adjustment for the step t that maximizes the improvement in $\text{Quality}()$ every time. If at some point we fail to improve the performance of the max algorithm, we stop (line 18).

Algorithm 4: $\text{RandomHillclimb}(x_0, \vec{p}, B, T, \mathcal{A})$

```

1  $(\hat{p}, \hat{r}, \hat{s}) \leftarrow \text{ConstantSequences}(x_0, \vec{p}, B, T, \mathcal{A});$ 
   //  $\hat{p}$  is the highest probability after a random source step is examined
2  $\hat{r}_i \leftarrow \hat{r}, \forall i;$ 
3  $\hat{s}_i \leftarrow \hat{s}, \forall i;$ 
4 repeat
   //  $c$  is a random source step
5    $c \leftarrow$  random step from tournament;
   //  $\bar{p}$  is the highest probability seen until after each target node is examined
6    $\bar{p} \leftarrow 0.0;$ 
   //  $t$  is a random target step
7   for  $t \leftarrow 1$  to  $\text{Time}(\mathcal{A}(\{\hat{r}_i\}, \{\hat{s}_i\}), \mathcal{E})$  do
8      $\{r_i\} \leftarrow \{\hat{r}_i\};$ 
     // reducing the repetitions for the source step
9      $r_c \leftarrow r_c - 1;$ 
     // increasing repetitions for the target step
10     $r_t \leftarrow$  maximum value that does not violate budget  $B;$ 
11     $p \leftarrow \text{Quality}(\mathcal{A}(\{r_i\}, \{\hat{s}_i\}), \mathcal{E});$ 
12    if  $p > \bar{p}$  then
13       $\bar{p} \leftarrow p;$ 
14       $\{\bar{r}_i\} \leftarrow \{r_i\};$ 
15  if  $\bar{p} > \hat{p}$  then
16     $\hat{p} \leftarrow \bar{p};$ 
17     $\{\hat{r}_i\} \leftarrow \{\bar{r}_i\};$ 
18 until  $\bar{p} < \hat{p};$ 
19 return  $(\hat{p}, \{\hat{r}_i\}, \{\hat{s}_i\});$ 

```

AllPairsHillclimb

AllPairsHillclimb is an extension of RandomHillclimb with one important difference: This strategy considers all possible steps as sources c (not just a random one). We again greedily change the r 's by keeping each time the source c and the target t that maximize the improvement in $\text{Quality}(A, \mathcal{E})$.

AllSAllPairsHillclimb

AllSAllPairsHillclimb is a generalization of the AllPairsHillclimb. The difference is that *all* possible s 's are considered (not only the one that ConstantSequences returns). This optimization can only lead to improvements for $\text{Quality}()$. We observe the significance of these improvements in our experiments in Section 6.

VaryingS

VaryingS is a generalization of AllSAllPairsHillclimb and it is the only strategy we examine that allows different s values in different steps. VaryingS operates as seen in Algorithm 5. The parameters for step 1 are the same as the parameters the AllSAllPairsHillclimb returns for step 1 (lines 1–5). If the parameters for steps 1, 2, \dots , $u - 1$ have been decided, the parameters for step u are decided as follows: r_u and s_u are set equal to the parameters for step 1 of AllSAllPairsHillclimb with the non-consumed budget and execution time and the appropriate number of input items (lines 8–12). The process stops when the selection of $\{s_i\}$ returns exactly one item in some step (line 7). After all the selections have occurred, $\text{Quality}()$ is computed and returned (lines 13–14).

Algorithm 5: $\text{VaryingS}(x_0, \vec{p}, B, T, \mathcal{A})$

```
// global optimal values
1  $(\hat{p}, \{\hat{r}_i\}, \{\hat{s}_i\}) \leftarrow (0.0, \text{NULL}, \text{NULL})$  ;
2  $(p, \{r_i\}, \{s_i\}) \leftarrow \text{AllSAllPairsHillclimb}(x_0, \vec{p}, B, T, \mathcal{A})$  ;
3  $u \leftarrow 1$  ;
4  $\hat{r}_u \leftarrow r_1$  ;
5  $\hat{s}_u \leftarrow s_1$  ;
6  $b_1 \leftarrow$  budget consumed from the selection of  $\hat{r}_1$  and  $\hat{s}_1$  on an input of size  $x_0$  ;
7 while  $x_u > 1$  do
8    $u \leftarrow u + 1$  ;
9    $(p, \{r_i\}, \{s_i\}) \leftarrow \text{AllSAllPairsHillclimb}(x_{u-1}, \vec{p}, B - b_{u-1}, T - (u - 1), \mathcal{A})$  ;
10   $\hat{r}_u \leftarrow r_1$  ;
11   $\hat{s}_u \leftarrow s_1$  ;
12   $b_u \leftarrow$  budget consumed from the selection of  $\hat{r}_1, \hat{r}_2, \dots, \hat{r}_u$  and  $\hat{s}_1, \hat{s}_2, \dots, \hat{s}_u$  on an input of size
     $x_0$  ;
13  $\hat{p} \leftarrow \text{Quality}(\mathcal{A}(\{\hat{r}_i\}, \{\hat{s}_i\}), \mathcal{E})$  ;
14 return  $(\hat{p}, \{\hat{r}_i\}, \{\hat{s}_i\})$  ;
```

5 Evaluation Framework

Max algorithms can be used by various applications and each application gives rise to different human behaviours. Also, each application may focus on different evaluation metrics. To understand the quality/time/cost tradeoff, we explore various concrete error and compensation worker models (still a fraction of what our framework supports) in Sections 5.1 and 5.2, explain how these models can be applied in other domains in Section 5.3, and propose various evaluation metrics in Section 5.4 (extending the already defined $\text{Quality}(A, \mathcal{E})$ metric).

5.1 Human Error Models

This section describes various models for human errors. We present our models from the most detailed one to the simplest one. As we have seen in Section 2.1, a set of items S is given to a human and the error model assigns probabilities to each (possible) response of a human. We assume there are no malicious workers, and thus $p_1 \geq \frac{1}{|S|}$ (i.e., the probability of returning the maximum item in S is at least as high as the probability of returning a random item from S). Appendix B describes how an appropriate model (and parameters) can be selected in practice.

Let the input set of items of each comparison be $S = \{e_1, e_2, \dots, e_{|S|}\}$, with e_i being the i^{th} best item in \mathcal{E} (or equivalently, $e_{|S|} < e_{|S|-1} < \dots < e_1$). As a reminder, a human response has probability p_i of returning item e_i . The rank of an item is defined as follows:

Definition 3 (Item’s Rank) *The rank of item $e_i \in S$, denoted as $\text{rank}(e_i, S)$, is defined as the number of items in S that are better than e_i plus 1.*

For example, $\text{rank}(e_1, S) = 0 + 1 = 1$ and $\text{rank}(e_i, S) = i$.

Proximity Error Model The proximity error model takes one parameter $p \in [\frac{1}{|S|}, 1]$ and assumes there is a distance function $d(\cdot, \cdot)$ that compares how different two items are. Formally, $d(e_i, e_j) \in (0, 1)$ (since $e_i \neq e_j$ for $i \neq j$). When $d(e_i, e_j)$ is close to 0, e_i is similar to e_j . On the contrary, when $d(e_i, e_j)$ is close to 1, e_i and e_j are very different.

A worker returns e_i with probability:

$$\begin{cases} p_1 = p \\ p_i = (1 - p) \cdot \frac{1 - d(e_i, e_1)}{\sum_{j=2}^{|S|} [1 - d(e_j, e_1)]}, i \in \{2, 3, \dots, |S|\} \end{cases}$$

The intuition is the following: Humans find it hard to distinguish similar items. Thus, the closer one item is to the maximum item in S , the higher its probability of being selected by the human as the maximum.

Example 3 *If the task of a human is to select the photograph with the highest contrast from a set of 3 photographs: e_1 , e_2 , and e_3 , with $d(e_2, e_1) = 0.1$, $d(e_3, e_1) = 0.9$. If $p = 0.6$, then $p_1 = 0.6$, $p_2 = (1 - 0.6) \times \frac{0.9}{0.1 + 0.9} = 0.36$, and $p_3 = (1 - 0.6) \times \frac{0.1}{0.1 + 0.9} = 0.04$. Effectively, we favor items with values closest to the maximum item in this model.*

Order-Based Error Model This model is similar to the previous model with one main difference: The distance function is defined as $d(e_i, e_j) = \frac{|\text{rank}(e_i, S) - \text{rank}(e_j, S)|}{|S|} \in (0, 1)$. A worker returns e_i with probability:

$$\begin{cases} p_1 = p \\ p_i = (1 - p) \cdot \frac{1 - \frac{|\text{rank}(e_i, S) - \text{rank}(e_1, S)|}{|S|}}{\sum_{j=2}^{|S|} \left[1 - \frac{|\text{rank}(e_j, S) - \text{rank}(e_1, S)|}{|S|} \right]} = (1 - p) \cdot \frac{|S| - |\text{rank}(e_i, S) - \text{rank}(e_1, S)|}{\sum_{j=2}^{|S|} [|S| - |\text{rank}(e_j, S) - \text{rank}(e_1, S)|]}, i \in \{2, 3, \dots, |S|\} \end{cases}$$

In this model the order of the items is important for the p_i values.

Example 4 *For the setting of Example 3, we have $\text{rank}(e_1, S) = 1$, $\text{rank}(e_2, S) = 2$, and $\text{rank}(e_3, S) = 3$. Thus, if $p = 0.6$ again, the order-based error model gives: $p_1 = 0.6$, $p_2 = (1 - p) \times \frac{\frac{2}{3}}{\frac{2}{3} + \frac{1}{3}} \approx 0.27$, and $p_3 = (1 - p) \times \frac{\frac{1}{3}}{\frac{2}{3} + \frac{1}{3}} \approx 0.13$.*

Linear Error Model This model assumes that the human worker is able to determine the maximum item in S with a probability that depends on the number of items $|S|$ to be examined. Intuitively, the more items to examine, the harder the task becomes.

The probability that a worker selects the maximum item is $p_1 = 1 - p_e - s_e \cdot (|S| - 2)$, where p_e, s_e are model parameters. Effectively, $p_1 = 1 - p_e$ when $|S| = 2$ and it decreases by s_e for each additional item in S . When the worker fails to return the maximum item (with probability $p_e + s_e \cdot (|S| - 2)$), he returns a random (but not the maximum) item from S . Thus, each item in $\{e_2, e_3, \dots, e_{|S|}\}$ has probability $\frac{1 - p_e - s_e \cdot (|S| - 2)}{|S| - 1}$ to be selected. Values p_e and s_e take values in such a way that $p_e + s_e \cdot (|S| - 2) \in [0, 1 - \frac{1}{|S|}]$ for all values of $|S|$ used in practice.

Example 5 For the setting of Example 4, with parameters $p_e = 0.6$ and $s_e = 0.05$, we have for the linear error model $p_1 = 0.6 - 0.05 \times (3 - 2) = 0.55$ and $p_2 = p_3 = \frac{1 - 0.55}{3 - 1} = 0.225$.

Constant Error Model This model assumes that the human worker is able to determine the maximum item from S with probability $p \in [\frac{1}{|S|}, 1]$, for any S (with $|S| \geq 2$). In the event that the worker is not able to determine the maximum item from S (which occurs with probability $1 - p$), he returns one random non-maximum item from S . Thus, each item in $\{e_2, e_3, \dots, e_{|S|}\}$ has probability $\frac{1 - p}{|S| - 1}$ to be selected.

Example 6 For the setting of Example 5, we have for the constant model if $p = 0.6$ the following: $p_1 = 0.6$, $p_2 = p_3 = \frac{1 - 0.6}{3 - 1} = 0.2$.

5.2 Human Cost Models

This section describes the models we explore for human work cost. The function $\text{Cost}(\cdot)$ we defined in Section 2.1 takes as input the number of items that are compared at a time.

Constant Cost Model In this model, each task has a fixed price. That is, $\text{Cost}(|S|) = c$ for some constant c .

Linear Cost Model In this model, each task has a price that depends on $|S|$. If the human worker works on two items, he is compensated with c , if he works on three items, he is compensated with $c + s_c$, if he works on four items, he is compensated with $c + 2 \times s_c$, and so on. Formally, $\text{Cost}(|S|) = c + s_c \times (|S| - 2)$, for some constants c and s_c .

5.3 Note on Models

Our proposed models are tailored to the max algorithm problem, but can be applied (with some extensions) to other problems. For example, the cost models can be used in tasks such as sorting, entity resolution, and top- k retrieval. Also, the error models can be applied to practically any task that has the following characteristics:

- has a fixed set of possible outcomes (e.g., multiple choice), and
- exactly one of the outcomes can be considered correct.

Of course, our models cannot model any possible task in a crowdsourcing environment, but they can be the basis for other kinds of tasks.

5.4 Evaluation Metrics

We defined $\text{Quality}(A, \mathcal{E})$ as the main optimization criterion, but there are other metrics of interest. $\text{Quality}(A, \mathcal{E})$ describes how often A returns the maximum item in \mathcal{E} . Other metrics evaluate how often A returns one of the top- k items in \mathcal{E} , or how close the returned items are to the maximum item. This section describes some standard metrics we have used for our evaluations. We note that we only optimize for the $\text{Quality}(A, \mathcal{E})$ metric, but evaluate max algorithms using the metrics we define in this section also.

We define the metrics of interest in the context of simulations. We simulate the application $A(\mathcal{E})$ (max algorithm A operates on input set \mathcal{E}) E times and then obtain the following values:

Top- k : The fraction of the E simulations for which $A(\mathcal{E})$ belonged in the top- k items of \mathcal{E} (i.e., $\text{rank}(A(\mathcal{E}), \mathcal{E}) \leq k$).

Mean Reciprocal Rank (MRR): The value: $\frac{1}{E} \sum_{i=1}^E \frac{1}{\text{rank}_i}$, where rank_i is the rank of the returned item in the i^{th} simulation.

Both metrics yield values in $[0, 1]$: Values close to 0 indicate that A returns items that are not close to the maximum item, while values close to 1 indicate that A returns items that are close to the maximum. Also, both metrics have a probabilistic meaning: For the top- k metric for example, if we have a significantly large number of simulations, we converge to the *probability* that A returns an item in the top- k items. Finally, we experimentally observe exceptionally low variance values for our metrics (lower than 10^{-3} in all cases), and only report the expected metric values in Section 6.

6 Experiments

This section summarizes various experiments we have performed, organized by results. We first explore which max algorithms and which strategies perform best. Surprisingly, we see that the same algorithm and strategy perform best under all combinations of human models and parameters we have tested (Section 6.1). Nevertheless, we explore in more detail:

- the effect of different parameters (like B , T , m , p , p_e , s_e , c , and s_c) on the performance of the max algorithms (Sections 6.1, 6.2, and 6.3),
- how useful repetitive max algorithms are (Section 6.4),
- characterizations of the “optimal” solutions (Section 6.5),
- the effectiveness of our proposed optimization strategies, by exploring various evaluation metrics (Section 6.6) and comparing our heuristics to exhaustive search (Section 6.7), and
- the sensitivity of our algorithms to inaccurate estimations of the crowdsourcing system error model parameters (Section 6.8), and

Most of our experimental results were analytically calculated (and confirmed with simulations). In a few cases, we resorted to simulations only, since analysis was not possible. When we use simulations, we point out why analysis was hard and how our simulations were run.

6.1 Max Algorithms and Strategies Performance

This section compares the effect of various strategies to the quality of the result `Quality()` for the two parameterized max algorithms we have defined in Section 3 (bubble and tournament).

We experimented with various human models and parameters: budget (B), time bound (T), error model parameters (p , p_e , s_e), and cost model parameters (c , s_c). All parameters tested gave qualitatively similar results. For succinctness, we present one selection of human models and parameters. Figure 3 shows `Quality()` on the y-axis and the budget B on the x-axis, for each of the five strategies we described in Section 4 (`ConstantSequences`, `RandomHillclimb`, `AllPairsHillclimb`, `AllSAllPairsHillclimb`, and `VaryingS`). The graph on the left refers to the tournament algorithm and the graph at the right to the bubble algorithm; the legend in the middle is common for both graphs. To generate Figure 3 we used $|\mathcal{E}| = 100$, the order-based error model with parameter $p = 0.78$ (one of the many values we tried in the range $[0.6, 1]$), the linear cost model with parameters $c = 1$ and $s_c = 0.1$, and allowed sets S participating in some `Comp()` to have size at most $m = 10$. Finally, we set the time bound T to infinity and varied the budget B between 20 and 100 with a step of 20.

We observe in Figure 3 that there are “discontinuities” (jumps) for `Quality()` as budget B increases. These discontinuities exist because: (1) B needs to be increased by a particular amount to increase r_i for

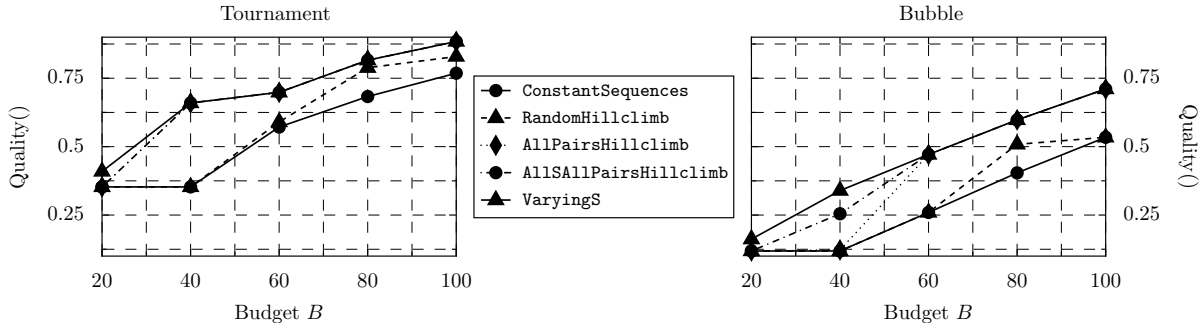


Figure 3: Evaluating five strategies’ (seen in the legend in the middle) effect on the tournament (left) and bubble (right) max algorithms.

some step i (e.g., a tournament algorithm performs $r_1 \times \lceil \frac{|\mathcal{E}|}{s_1} \rceil$ comparisons at step 1, thus to increase r_1 by 1 requires an additional budget for $\lceil \frac{|\mathcal{E}|}{s_1} \rceil$ new comparisons), and (2) the r_i increase does not have the same Quality() impact all times (e.g., increasing any r_i from 1 to 2 has no impact to Quality() by the plurality rule properties, but increasing r_i from 2 to 3 increases Quality()).

From Figure 3, we obtain the following:

Result 1: As expected, as the budget B increases, the quality of the result Quality() increases (for all strategies).

Result 1 states the intuitive fact that the more money one is willing to spend, the higher the quality of the result is. The increase is almost linear for the range of budgets we examined. Furthermore, we do not observe topping off of gains for the values of budget B we present, but of course it takes place at higher values.

When comparing the five strategies with each other we obtain the following:

Result 2: For all budgets examined and for both the bubble and the tournament max algorithms there is a clear ordering of the strategies from best to worst: VaryingS, AllSAllPairsHillclimb, AllPairsHillclimb, RandomHillclimb, and ConstantSequences. Thus, VaryingS is the strategy that should be used in all cases in practice.

Most applications today use strategies like ConstantSequences, i.e., $r_i = r$ and $s_i = s$ for all steps i . Result 2 suggests that the quality of the result improves when the number of repetitions and the set size are allowed to vary across steps. Interestingly, comparing to ConstantSequences, VaryingS increases Quality() by $\sim 100\%$ for the same budget in some cases (e.g., for the tournament algorithm for $B = 40$)! Thus, applications can benefit by our strategies in two ways: either increase Quality() for a fixed budget, or produce a desired Quality() for a lower budget.

Comparing the two max algorithms (bubble and tournament) we obtain the following:

Result 3: Tournament max algorithms perform better than bubble max algorithms for the same budget.

The performance difference between the tournament and the bubble max algorithms can be explained in the following way. The maximum item has to “survive” (not erroneously be dropped by a comparison Comp()) by all the comparisons it participates in for it to be the output of the max algorithm. In bubble max algorithms the worst case scenario is that the maximum item survives a very long chain of comparisons

($\Theta(|\mathcal{E}|)$), while in the tournament max algorithms the number of comparisons is small in all cases ($\Theta(\log |\mathcal{E}|)$). Finally, it is important to note that in *no* combination of human models and parameters that we tested, the bubble max algorithms performed better than the tournament max algorithms.

Now, observing each of the families of max algorithms separately, we observe that:

Result 4: For very small or very large budgets B , the differences between the five strategies are not significant. For middle values of B though we observe large differences between the strategies. We generally observe large differences between the strategies for $B \approx \frac{|\mathcal{E}|}{2}$ shifted appropriately as the cost parameters change. Thus, it is “indifferent” which strategy is used for very small or very large values of B , but for the middle values **VaryingS** should be used when possible.

The differences between the `Quality()` values the strategies achieve, can be explained if we think of each strategy as a search in the parameter space of sequences $\{r_i\}$ and $\{s_i\}$. For small values of B the differences between the resulting `Quality()` values are not significant, because there are not many choices of $\{r_i\}$ and $\{s_i\}$ permitted by budget B . As B increases, there are more parameter choices and the differences between the strategies’ performance increase: Some strategies are able to determine good parameter choices and some are not. For very high values of B the differences between the performance of the algorithms decreases again. There are many $\{r_i\}$ and $\{s_i\}$ choices and all strategies perform very well: If there is enough budget, one can increase r_i ’s significantly and observe high `Quality()` for any strategy. Browsing through the resulting $\{r_i\}$ and $\{s_i\}$ from the various strategies for large B ’s indicates that the actual selected r_i and s_i values may be significantly different, but the `Quality()` values are very close. We also note that the differences between strategies are not significant when p_1 is close to 1.

Attempting to characterize the sequences $\{s_i\}$ that led to optimal `Quality()` values we see that:

Result 5: There is no clear pattern for the s_i values.

We give some intuition why a clear pattern is hard to observe for the s_i values. For the linear error model, for example, there is a tradeoff that the size of the sets s_i creates: Having large sets (large s_i ’s) leads to larger errors per question, but we are able to ask more questions (if the budget is fixed). Whether that is beneficial or not depends on the parameters of the models. Such tradeoffs often arise in our setting, and our framework allows us to make the appropriate selections of parameters in all cases we explored.

Interestingly, there are clear patterns for the $\{r_i\}$ sequence, that we discuss in Section 6.5.

6.2 Error Model Parameters

We now investigate the performance of the bubble and tournament max algorithms for various values of $1 - p_1$, i.e., the probability of a human making an error.

Figure 4 shows `Quality()` on the y-axis vs. the probability of a human worker making a mistake in his responses $1 - p_1$ on the x-axis. The graph includes two lines: one for the tournament max algorithms and one for the bubble max algorithms. For both lines we used the **VaryingS** strategy to select $\{r_i\}$ and $\{s_i\}$ since, as we have seen (Result 2), it is the best strategy. We used $|\mathcal{E}| = 100$ items, with budget $B = 40$, the linear cost model $c = 1$ and $s_c = 0.1$, allowing at most $m = 7$ items per comparison. We experimented with the order-based error model and varied $1 - p_1$ between 0.1 to 0.3. Our results were similar for other parameters and models.

From Figure 4 we observe that:

Result 6: As the probability of a human to make mistakes increases, the quality of the result decreases almost linearly for fixed budget B . The slope for the bubble max algorithms is (absolutely) higher than the one for the tournament max algorithms. Thus, running a max algorithm in marketplaces with higher-quality workers gives better results.

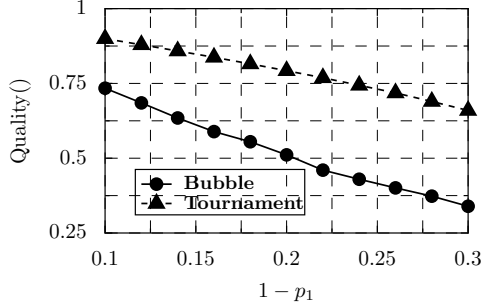


Figure 4: Evaluating the bubble and the tournament max algorithms for various probabilities that a human makes mistakes.

The fact that tournament max algorithms perform better than bubble max algorithms was explained in Result 3. The fact that bubble max algorithms get worse and worse than the tournament max algorithms as $1 - p_1$ increases can be explained in a similar way. For the tournament max algorithms the number of steps where the maximum item is included is small. Thus, we expect a small decrease in $\text{Quality}()$ as $1 - p_1$ increases. On the contrary, for the bubble algorithms the number of steps where the maximum item is included (on average) is larger. Thus, we expect a larger decrease in $\text{Quality}()$ as $1 - p_1$ increases.

6.3 Time Bound Study

This section studies the effect of the time bound constraint ($\text{Time}(A, \mathcal{E}) \leq T$) on $\text{Quality}()$.

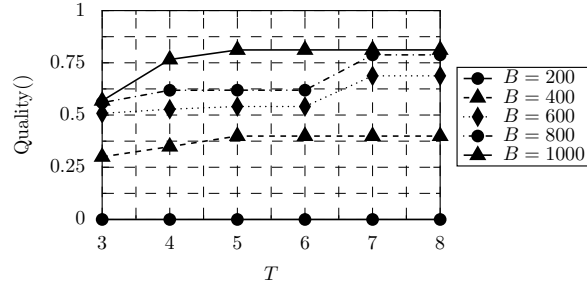


Figure 5: Observing the effect of different time bounds on the quality of the result.

Figure 5 shows the time bound T (the number of steps in which A has to complete) on the x-axis and $\text{Quality}()$ on the y-axis. We used the linear error model with parameters $p_e = 0.03$, $s_e = 0.05$, and the linear cost model with parameters $c = 1$ and $s_c = 0.3$ on $|\mathcal{E}| = 1,000$ items. The budget B varied between 200 and 1,000, and m was set to 10.

We observe that:

Result 7: The more relaxed the time bound constraint is, the better the quality of result is. Thus, if our time constraints allow it, letting a task run for more steps can yield significantly better results.

For small budgets ($B = 200$), we our max algorithms cannot run (i.e., $\text{Quality}() = 0$). As the budget gets higher, $\text{Quality}()$ gets improves. For any particular budget we see that as we relax the time bound constraint (increase T), the value of $\text{Quality}()$ increases. For example, for $B = 1,000$ we see $\text{Quality}()$ increasing from 0.56 it goes up to 0.79, an improvement of 0.23! A similar behavior is observed for all other budgets.

6.4 Repetitive Max Algorithms

This section investigates repetitive max algorithms (Section 3.4). In detail, we performed the following experiment:

Let R be the number of repetitions and B be the total budget. We execute an algorithm A_R a total of R times, and return the most frequent output (ties broken arbitrarily). To determine A_R , we use strategy `VaryingS` with a budget of $\frac{B}{R}$ and the desired input size $|\mathcal{E}|$.

For each run (defined by a set size $|\mathcal{E}|$, a budget B , and repetitions R) we generated 100,000 input sets and counted how many times the repetitive max algorithm retrieved the correct maximum item.

In our experiment we found that $R = 1$ always provides the best results:

Result 8: Repetitive max algorithms are not helpful in our setting. Thus, allocating the full budget to a single max algorithm is the best approach.

Running an algorithm A_R , for $R \geq 2$ on some input \mathcal{E} has $\text{Quality}(A_R, \mathcal{E}) \leq \text{Quality}(A_1, \mathcal{E})$. Our experiments suggest that repeatedly (R times) applying A_R on input \mathcal{E} improves the probability that the maximum item in \mathcal{E} is returned. Even though, the R repetitions are not enough to surpass $\text{Quality}(A_1, \mathcal{E})$. Thus, the repetitive algorithms are not useful in any setting we have examined.

6.5 Distribution of r 's

This section explores how the r_i values are distributed across the steps, when all steps have the same s_i . We experiment with the `AllPairsHillclimb` strategy and the tournament max algorithm.

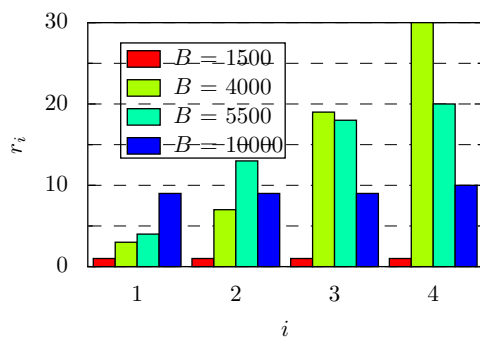


Figure 6: Exploring how `AllPairsHillclimb` algorithm distributes repetitions (r_i 's) across steps.

Figure 6 shows the r_i values on the y-axis achieved for each step i (x-axis). For simplicity, we used $s_i = 10$ for each step i and observed the r_i 's. We used $|\mathcal{E}| = 10,000$ (thus, there are exactly 4 steps for the tournament max algorithm), infinite time bound ($T = \infty$), the constant error model with $p = 0.8$, and the constant human cost model with $c = 1$. We used various budgets (from $B = 1,500$ up to $B = 10,000$).

From Figure 6 we observe that:

Result 9: Balanced numbers of repetitions across steps are beneficial when the budget allows it. Thus, when r_i 's are unbalanced, it is because the budget does not allow more repetitions to be allocated in some steps.

We observe in Figure 6 that when the budget is low ($B = 1,500$) all 4 steps are assigned one repetition each ($r_i = 1$ for all i 's). As budget increases ($B = 4,000$ or $B = 5,500$) mainly the last steps are benefited

from more repetitions. (The number of comparisons performed in the initial steps is very high, and thus it costs a lot to allocate comparisons in them.) As we increase the budget even more ($B = 10,000$), we observe that the repetitions across steps are again balanced (approximately 10 repetitions per step for all steps).

6.6 Various Metrics

This section evaluates the `VaryingS` strategy on the tournament max algorithms for the metrics we defined in Section 5.4: (1) MRR and (2) top- k success. The optimization of the max algorithm parameters occurs with `Quality()` as the objective. (`Quality()` and top-1 are the same: The probability that an algorithm returns the maximum/top-1 item.) After we optimize the max algorithm we examine the other metrics via simulations. All metrics follow a common behaviour: the closer they are to 1, the better the results are.

To generate Figure 7, we used the constant error model for various values of p . We used the `VaryingS` strategy to optimize the tournament max algorithms with an input of $|\mathcal{E}| = 1,000$ items, $T = \infty$, $B = 600$, $m = 10$, and linear cost model with $c = 1$ and $s_c = 0.4$. The x-axis is the probability of a human worker making a mistake ($1 - p_1$) and the y-axis is the metric value for the corresponding $1 - p_1$ value.

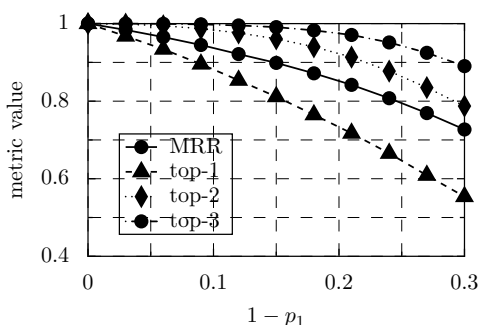


Figure 7: Comparing the effect that $1 - p_1$ for various different evaluation metrics.

From Figure 7 we observe:

Result 10: Even though the optimization occurs for `Quality()`, the optimized max algorithms have similar behaviour for all metrics examined. Thus, optimizing over `Quality()` *only*, seems enough for any sensible scenario.

In Figure 7 we see that the `Quality()` metric (top-1 in the graph) performs worse than all other metrics. Of course, top-1 is the most penalizing metric by definition: Either the max algorithm returns the maximum item or the max algorithm receives no credit. For the top- k metrics, as k increases, the higher the metric value is.

More than that, we observe that:

Result 11: Obtaining items very close to the maximum item is *considerably* more likely than obtaining *the* maximum item. Thus, if an application asks for an item close to the maximum, using our max algorithms and strategies is enough.

We can see that even when the probability of obtaining *the* maximum item from a set is close to 0.55 (for $1 - p_1 = 0.3$), the algorithm returned one of the top-3 items in 89% of the experiments. Furthermore, the algorithm returned an item in top-5 in more than 99% of the experiments!

6.7 Comparison with Exhaustive Search

This section evaluates how effective our strategies are with respect to an exhaustive search of the parameter space. In order to evaluate our strategies, we perform an exhaustive search over the space of $\{r_i\}$ and $\{s_i\}$. The exhaustive search is very expensive, and thus we focus on a small instance of the problem: We consider $|\mathcal{E}| = 10$, $m = 4$, and set a maximum value for r_i to 5. We use the linear error model (with $p_e = 0.2$ and $s_e = 0.04$), and the linear cost model (with $c = 1$ and $s_c = 0.2$). We set the budget B to 10, and the time bound T to ∞ . For the exhaustive search, we find all $\{r_i\}$ and $\{s_i\}$ that lead to $\text{Cost}(A, \mathcal{E})$ below B and for each $A = \mathcal{A}_T(\{r_i\}, \{s_i\})$ we simulate (averaging over 500,000 simulation runs) the value for each of the following metrics: MRR and top- k for $k = 1, 2$, and 3. (As mentioned earlier, $\text{Quality}()$ and top-1 are the same: They both measure the probability that an algorithm returns the maximum item.) We only evaluate the **VaryingS** strategy for the tournament max algorithms, which is the best strategy (Result 2).

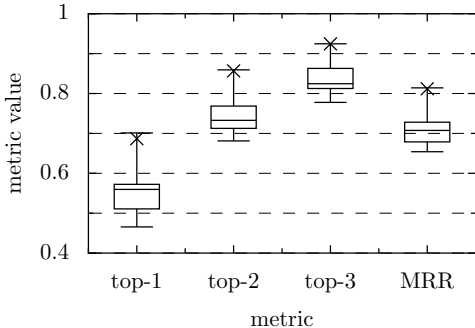


Figure 8: Comparing the **VaryingS** strategy for the tournament max algorithms with an exhaustive search over the parameter space.

Figure 8 shows the distribution of the metric values using box/whiskers plots and evaluates the effectiveness of **VaryingS**. Each box/whiskers plot shows the range of metric values observed as we exhaustively examine all $\{r_i\}$ and $\{s_i\}$ values (that meet the budget constraint). The top bar represents the maximum value observed, the bottom bar is the lowest value. The bottom of the box is the first quartile (minimum value above the 25% percent of the observed values), the top of the box is the third quartile (minimum value above 75% percent of the observed values), and the line in the middle of the box is the median value observed. The x-axis presents the metrics we examined (top- k and MRR). The y-axis presents the metric values. For each metric, we present the achieved value by **VaryingS** using the \times symbol.

By observing the range of metric values the exhaustive search achieves, we see that:

Result 12: For all metrics, there is a wide range of values that are achieved by different parameter selections. Thus, simple strategies are not likely to work well.

Observing the performance of the **VaryingS** strategy, we see that:

Result 13: Even though **VaryingS** optimized over only one objective function (top-1 or $\text{Quality}()$), it produces very good results in all metrics we examined. Thus, it is enough to only optimize over $\text{Quality}()$ using **VaryingS**, to obtain high quality results.

Result 13 suggests that using **VaryingS** to tune algorithms is enough for any metric of interest.

6.8 Error Model Parameters Sensitivity

Modeling humans is a very difficult task, and thus it is vital for an algorithm to be tolerant in erroneous estimations of the error model parameters. This section explores how sensitive our algorithms are to the error

model parameters. For our study, we assume that a crowdsourcing marketplace has error model parameters \bar{P} , but the algorithm $A = \mathcal{A}_T(\{r_i\}, \{s_i\})$ that is run was generated by strategy **VaryingS** with parameters $\bar{P}' \neq \bar{P}$.

We have experimented with all the error models we have described in Section 5.1 obtaining similar results. We now describe one of the many experiments we have performed. For this experiment, we used $|\mathcal{E}| = 1,000$ items with the linear cost model ($c = 1$ and $s_c = 0.2$), our budget was $B = 1,400$, the time bound was $T = \infty$, and m was set to 10. We used the linear error model with parameters $p_e = 0.15$ and $s_e = 0.02$. We generated various algorithms A using the **VaryingS** strategy, assuming that the crowdsourcing marketplace had parameters $p'_e \in (0, 0.3)$ and $s'_e = s_e = 0.02$. In Figure 9 we see the difference $\Delta p_e = p'_e - p_e$ (x-axis) and the $\text{Quality}()$ achieved by the algorithm A (y-axis) produced with the corresponding Δp_e (i.e., the algorithm returned by **VaryingS** when $p'_e = p_e + \Delta p_e = 0.15 + \Delta p_e$).

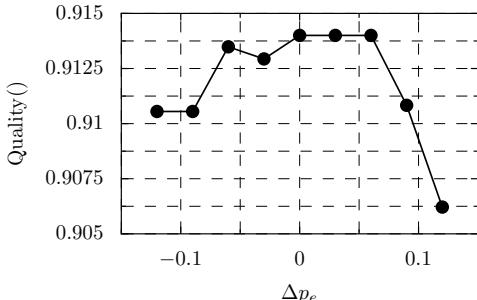


Figure 9: Sensitivity of a tournament algorithm to the linear error model parameter p_e .

We see that:

Result 14: As expected, the highest value for $\text{Quality}()$ is observed for $\Delta p_e = 0$. Negative and positive values of Δp_e (i.e., $p'_e \neq p_e$) give slightly (but not significantly) smaller values for $\text{Quality}()$. Thus, small parameter errors do not significantly impact the performance of the algorithms.

Because of Result 14, the estimations of the actual parameters of the error models followed in practice do not need to be perfectly accurate. Thus, the cost for estimating the model parameters of a marketplace can be kept to a minimum with small accuracy losses in the algorithm’s application.

7 Related work

A lot of work has been done in the context of our problem. In summary, comparing to previous work, our paper is different in at least three aspects:

1. **Error/cost models:** The error and cost models we consider are more general than previously considered error models.
2. **Notion of Steps:** Crowdsourcing environments force algorithms to be executed in steps, making the design of algorithms harder.
3. **Number of items compared:** We allow the flexibility of comparing “any” number of items per operation. All other techniques we are aware of only perform binary comparisons.

In the rest of this section we present work related to our paper.

Resilient algorithms Some related work to ours has extensively examined so-called resilient algorithms. These algorithms assume that some items from the input are corrupted (e.g., because of memory faults). The goal is to produce the correct result (for computing the maximum item in a set for example) for the uncorrupted items in the input. One of the first papers in this area is [9], where resilient algorithms for sorting and the max algorithm problem are provided. An overview paper on resilient algorithms is [13], which presents work done in resilient counting, resilient sorting, and the resilient max algorithm problem. Along these lines [10, 11] attempt to understand for a given algorithm, how many corruptions can be tolerated for it to run correctly.

In our work we assume that no item from the input may be corrupted, but that humans (who are effectively “comparators”) can make mistakes. Also, only binary comparisons are considered in [9, 10, 11, 13]; we consider any comparison of size 2 or more.

Sorting/Max algorithms with errors Another line of work similar to ours involves sorting networks, in which some comparators can be faulty. One of the first works in this area [23] proposes sorting algorithms that deal with two scenarios of faults: (1) up to k comparators are faulty and (2) each comparator has a probability δ (δ is small) of failing and yet, the result is retrieved correctly with probability greater than $1 - \varepsilon$ (ε is small). In [4] sorting networks that return the correct result with high probability are constructed from any network that uses unreliable comparators (each comparator can be faulty with some probability smaller than $\frac{1}{2}$). The depth of the resulting network is equal to a constant multiplied by the depth of the original network.

In [19] the max algorithm problem is considered under two error models: (1) up to e comparisons are wrong, and (2) all “yes” answers are correct and up to e “no” answers are wrong. The max (and the sorting) problem is also considered in [3] under a different error model: If the two items compared have very similar values (their absolute difference is below a threshold), then a random one is returned; otherwise, the correct item is returned. The max algorithm problem is solved using $2 \cdot n^{\frac{3}{2}}$ comparisons. An extension of the error model used in [19] is given in [2] where up to a fraction of p of the questions are answered erroneously at any point. In [2] it is proved that if $p \geq \frac{1}{2}$, then the maximum item may not be retrieved; if $p < \frac{1}{2}$, then the maximum item can be retrieved with $\Theta\left(\frac{1}{(1-p)^n}\right)$ binary comparisons (where n is the size of the input). Even more elaborate error models are considered in [2]: For example, the number of wrong answers we have received may be at most p fraction of the number of questions *only at the end of the process* (and for example the first 5 questions may all be answered incorrectly). In this model if $p \geq \frac{1}{n-1}$, then the maximum item may not be found; if $p < \frac{1}{n-1}$, then the maximum item can be found in $n - 1$ questions.

The complexity of four different problems is explored in [7, 8]. The problems considered are: (1) binary search, (2) sorting, (3) merging, and (4) selecting the k^{th} best item. The error model considered is the following: For any comparison of items there is a probability p that the correct result is returned and $1 - p$ that the wrong result is returned.

Again, this line of work also does not have the notion of steps, nor the complexity of the error models we are exploring. Also, the only type of comparisons that has been used is that of two items per comparison.

Crowdsourcing Marcus et al. [17] consider the problems of sorting and joining datasets in real crowdsourcing system, considering many practical issues. Their work implements max algorithms (as instances of sorting algorithms) by splitting the input set of items \mathcal{E} in non-overlapping sets of equal size and asking workers to sort these sets. Our work on the contrary allows us to answer more general questions regarding different scenarios, such as different types of human workers and compensation schemes, different algorithms and strategies. For instance, we can use our framework to tune the tournament algorithm in the setting described in [17].

Tournament algorithms have been used in crowdsourcing to determine the correct answer to a difficult task [21]. Answers provided by n workers to a task (not necessarily a comparison) are compared (in pairs) by workers. The winners of the pairwise comparisons are compared with each other in the next step. This process continues for a fixed number of steps, after which the majority answer is returned. The intuition is that workers are better in identifying the correct answer (when comparing it to other answers) than in producing the correct answer.

Sorting in steps Work has also been done in sorting in a particular number of rounds (or steps as we defined them). Häggvist and Hell prove bounds on the complexity of sorting n items in r rounds [12]. In the same spirit, [5] answers the following question: How many processing units are required if one wants to sort n items in r rounds? In these two works there is no uncertainty about the data nor the comparisons.

Voting systems Voting systems have been used to declare a winner (maximum item in our case) according to votes (comparisons to other items in our case). One work in this line of research is [20], which declares a winner such that it creates the least disagreement with the votes of the crowd. Furthermore, de Condorcet [6] and List and Goodin [16] provide useful insights for the plurality rule from the standpoint of political philosophy.

8 Conclusions

We investigated methods for retrieving the maximum item from a set in a crowdsourcing environment. We developed parameterized families of algorithms to retrieve the maximum item and proposed strategies to tune these algorithms under various human error and cost models. We concluded, among other things, the following:

- The type of worker errors impact results accuracy, but not what is the best algorithm/strategy.
- It pays off to vary the size of a task.
- It pays off to vary/optimize the number of repetitions.
- Finding the maximum item with high accuracy is expensive, unless workers are very reliable. However, it is much easier (costs less) to find an item in the top-2 or top-3.

Our work is the first step in the development of rigorous max algorithms in crowdsourcing environments. We have started exploring extensions in the following directions: Determining the minimum cost and execution time required for achieving a desired quality of the result. Furthermore, we develop more elaborate models for the execution time that take into account the “size” of a marketplace (i.e., number of workers). In crowdsourcing marketplaces workers are not always interested to work in a particular task. Thus, we need to take into account the event of “no answers” in the max algorithms, an extension we already consider. Finally, although our models provide key insights in crowdsourcing marketplaces, we are developing non-uniform worker models.

While the max algorithm problem is very interesting and important in the crowdsourcing setting, there are other types of problems that still need to be tackled. Natural extensions of our work include the retrieval of the top- k items from a set and sorting sets of items. More than that, in the practitioner’s front, we plan to evaluate multiple models for different application, propose appropriate models for each applications, and suggest guidelines for running programs on crowdsourcing platforms.

9 Acknowledgements

We thank Aris Anagnostopoulos and Aditya G. Parameswaran for useful early discussions on this work.

References

- [1] Amazon Mechanical Turk. <http://www.mturk.com/>, sep 2011.
- [2] Martin Aigner. Finding the Maximum and Minimum. *Discrete Applied Mathematics*, 74(1):1–12, 1997.
- [3] Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. Sorting and Selection with Imprecise Comparisons. In *Automata, Languages and Programming*, pages 37–48. 2009.

- [4] Shay Assaf and Eli Upfal. Fault Tolerant Sorting Network. *FOCS*, pages 275–284, 1990.
- [5] Béla Bollobás and Andrew Thomason. Parallel Sorting. *Discrete Applied Mathematics*, 6(1):1–11, 1983.
- [6] Marquis de Condorcet. *Essai Sur L' Application De L' Analyse À La Probabilité Des Décisions Rendues À La Pluralité Des Voix*. 1785.
- [7] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Computing with Unreliable Information. In *STOC*, pages 128–137, 1990.
- [8] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with Noisy Information. *SIAM J. Comput.*, pages 1001–1018, 1994.
- [9] Irene Finocchi, Fabrizio Grandoni, and Giuseppe Italiano. Designing Reliable Algorithms in Unreliable Memories. In *Algorithms — ESA*, pages 1–8. 2005.
- [10] Irene Finocchi, Fabrizio Grandoni, and Giuseppe Italiano. Optimal Resilient Sorting and Searching in the Presence of Memory Faults. In *Automata, Languages and Programming*, pages 286–298. 2006.
- [11] Irene Finocchi and Giuseppe F. Italiano. Sorting and Searching in the Presence of Memory Faults (without Redundancy). In *STOC*, pages 101–110, 2004.
- [12] Roland Häggkvist and Pavol Hell. Sorting and Merging in Rounds. *SIAM J. on Matrix Analysis and Applications*, 3:465–473, 1981.
- [13] Giuseppe Italiano. Resilient Algorithms and Data Structures. In *Algorithms and Complexity*, pages 13–24. 2010.
- [14] Erich L. Lehmann and George Casella. *Theory of Point Estimation*. Springer Texts in Statistics. Springer, August 1998.
- [15] Erich L. Lehmann and Joseph P. Romano. *Testing Statistical Hypotheses*. Springer Texts in Statistics. Springer, New York, third edition, 2005.
- [16] Christian List and Robert E. Goodin. Epistemic democracy: Generalizing the Condorcet Jury Theorem. *Journal of Political Philosophy*, 9:277–306, 2001.
- [17] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-Powered Sorts and Joins. *PVLDB*, 5:13–24, September 2011.
- [18] National Scientific Corporation. *Voting Systems*. National Scientific Corporation, 1974.
- [19] Bala Ravikumar, K. Ganesan, and K. B. Lakshmanan. On Selecting the Largest Element in spite of Erroneous Information. In *STACS*, pages 88–99. 1987.
- [20] Ronald L. Rivest and Emily Shen. An Optimal Single-Winner Preferential Voting System based on Game Theory. In *COMSOC*, pages 399–410, 2010.
- [21] Yu-An Sun, Christopher R. Dance, Shourya Roy, and Greg Little. How to Assure the Quality of Human Computation Tasks When Majority Voting Fails? Workshop on Computational Social Science and the Wisdom of Crowds (NIPS), 2011.
- [22] Luis von Ahn. Games with a purpose. *Computer*, 39:92–94, 2006.
- [23] Andrew C. Yao and F. F. Yao. On Fault-Tolerant Networks for Sorting. Technical report, Stanford, CA, USA, 1979.

A Human Responses Aggregation Rules

This section describes two (among many) rules \mathbf{R} that can be used in practice for deciding which item from a set S should be declared the maximum item if we have r human responses for set S . These rules are inspired by voting systems [18].

We assume that operation $\text{Comp}(S, r, \mathbf{R})$ is executed, where $S = \{e_1, e_2, \dots, e_s\}$ and $|S| = s$. The items in S are such that $e_s < e_{s-1} < \dots < e_1$. The probability that item e_i is selected by a human worker is p_i (with $\sum_{i=1}^{|S|} p_i = 1$). We refer to the probabilities p_1, p_2, \dots, p_s collectively by \vec{p} .

The number of human responses (or votes) that items e_1, e_2, \dots, e_s each receive in this setting follows a multinomial distribution with parameters \vec{p} . Thus, if r votes are provided, the probability that k_i votes have been issued for item e_i is $\frac{r!}{k_1! \cdot k_2! \cdot \dots \cdot k_s!} \cdot \prod_{i=1}^s \binom{r}{k_i} p_i^{k_i}$.

We denote by $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R})$ the probability that a particular rule \mathbf{R} returns the maximum item from a set S , when operation $\text{Comp}(S, r, \mathbf{R})$ is executed and $|S| = s$, assuming that one knows the probabilities \vec{p} .

A.1 Majority rule

The *majority rule* (denoted by \mathbf{R}_M) chooses the item that it considers to be the maximum item in set S in the following way:

Case 1: if there is an item that has received more than $\lfloor \frac{r}{2} \rfloor$ responses by humans (majority or more votes) it is returned by $\text{Comp}(S, r, \mathbf{R}_M)$;

Case 2: if there is no item that has received more than $\lfloor \frac{r}{2} \rfloor$ responses by humans (there is no majority) a random item from S is returned by $\text{Comp}(S, r, \mathbf{R}_M)$.

The probability $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_M)$ in this case is

$$\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_M) = \Pr[e_1 \text{ is returned}] = \Pr[e_1 \text{ is returned in case 1}] + \Pr[e_1 \text{ is returned in case 2}] \quad (5)$$

We can only be correct in case 1 if item e_1 is selected at least $\lfloor \frac{r}{2} \rfloor + 1$ times:

$$\Pr[e_1 \text{ is returned in case 1}] = \sum_{k=\lfloor \frac{r}{2} \rfloor + 1}^r \binom{r}{k} p_1^k (1 - p_1)^{r-k} \quad (6)$$

Also, if we are in case 2, we select one of the items in S at random, and thus:

$$\Pr[e_1 \text{ is returned} \mid \text{case 2}] = \frac{1}{s} \quad (7)$$

Moreover, case 2 occurs when all items have received up to $\lfloor \frac{r}{2} \rfloor$ human responses. Thus:

$$\Pr[\text{case 2}] = \sum_{\substack{0 \leq k_1, k_2, \dots, k_s \leq \lfloor \frac{r}{2} \rfloor \\ \sum_{i=1}^s k_i = r}} \left[\frac{r!}{k_1! \cdot k_2! \cdot \dots \cdot k_s!} \prod_{i=1}^s \binom{r}{k_i} p_i^{k_i} \right] \quad (8)$$

Thus, from Equations (7) and (8) we have:

$$\begin{aligned} \Pr[e_1 \text{ is returned in case 2}] &= \Pr[e_1 \text{ is returned} \mid \text{case 2}] \times \Pr[\text{case 2}] = \\ &= \frac{1}{s} \cdot \sum_{\substack{0 \leq k_1, k_2, \dots, k_s \leq \lfloor \frac{r}{2} \rfloor \\ \sum_{i=1}^s k_i = r}} \left[\frac{r!}{k_1! \cdot k_2! \cdot \dots \cdot k_s!} \prod_{i=1}^s \binom{r}{k_i} p_i^{k_i} \right] \end{aligned} \quad (9)$$

From Equations (5), (6), and (9) we obtain:

$$\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_M) = \sum_{k=\lfloor \frac{r}{2} \rfloor + 1}^r \left[\binom{r}{k} p_1^k (1 - p_1)^{r-k} \right] + \frac{1}{s} \cdot \sum_{\substack{0 \leq k_1, k_2, \dots, k_s \leq \lfloor \frac{r}{2} \rfloor \\ \sum_{i=1}^s k_i = r}} \left[\frac{r!}{k_1! \cdot k_2! \cdot \dots \cdot k_s!} \prod_{i=1}^s (p_i^{k_i}) \right] \quad (10)$$

In Section A.3 we compare this rule to rule described in the following section (plurality rule).

A.2 Plurality rule

We saw the plurality rule in Section 3.1, and we remind how it operates here. The *plurality rule* (denoted by \mathbf{R}_P) chooses the item that it considers to be the maximum in set S in the following way:

If l items have received a number of n responses each and there is no other item with more responses than $n - 1$, then one of the l items is considered to be the maximum item in S (at random). The l items are called *winners*.

We now provide the derivation for Equation (2). The probability $\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_P)$ in this case is

$$\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_P) = \Pr [e_1 \text{ is returned}] = \sum_{l=1}^s \Pr [e_1 \text{ is returned} \mid e_1 \in l \text{ winners}] \times \Pr [e_1 \in l \text{ winners}] \quad (11)$$

Now, since one of the winners is selected at random:

$$\Pr [e_1 \text{ is returned} \mid e_1 \in l \text{ winners}] = \frac{1}{l} \quad (12)$$

Also,

$$\Pr [e_1 \in l \text{ winners}] = \sum_{n=1}^r \Pr [e_1 \in l \text{ winners, with } n \text{ responses each}].$$

and in the summation, each of the l winners receives n votes. Because the number of human responses per item follows the multinomial distribution, we have:

$$\Pr [e_1 \in l \text{ winners}] = \sum_{n=1}^r \sum_{L \in \mathcal{L}} \sum_{\substack{0 \leq k_i \leq n-1, i \in \bar{L} \\ \sum_{i \in \bar{L}} k_i + l \cdot n = r}} \left[\frac{r!}{(n!)^l \cdot \prod_{j \in \bar{L}} k_j!} \cdot \prod_{z \in L} p_z^n \cdot \prod_{w \in \bar{L}} p_w^{k_w} \right] \quad (13)$$

where \mathcal{L} is the set of all sets L that are subsets of $\{1, 2, \dots, s\}$ with $|L| = l$ and $1 \in L$. Also, $\bar{L} = \{1, 2, \dots, s\} \setminus L$. The second summation is over all such subsets. There are $\binom{s-1}{l-1}$ such sets. Index 1 is always included because in order to be the returned item, it must belong in the winners and must have received n responses.

Thus, from Equations (11), (12), and (13) we have:

$$\text{AggrQuality}(s, r; \vec{p}, \mathbf{R}_P) = \sum_{l=1}^s \frac{1}{l} \cdot \sum_{n=1}^r \sum_{L \in \mathcal{L}} \sum_{\substack{0 \leq k_i \leq n-1, i \in \bar{L} \\ \sum_{i \in \bar{L}} k_i + l \cdot n = r}} \left[\frac{r!}{(n!)^l \cdot \prod_{j \in \bar{L}} k_j!} \cdot \prod_{z \in L} p_z^n \cdot \prod_{w \in \bar{L}} p_w^{k_w} \right] \quad (14)$$

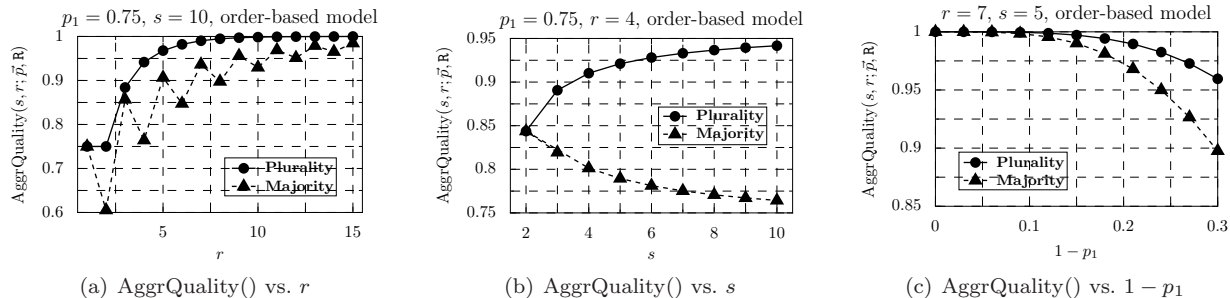


Figure 10: Comparing R_M and R_P .

A.3 Rules Comparison

This section compares the two rules we have described: majority and plurality. The main conclusion is that the plurality rule seems to perform better than the majority rule in all cases.

In Figure 10 we see some sample comparisons of the two rules we described, for the case of the order-based error model (the graphs look qualitatively similar for all models). The results were obtained analytically, by calculating Equations (2) and (10) (and also verifying via simulations). We can see that R_P is at least as good as R_M in Figure 10 (but also for all combinations of the parameters that are not depicted in Figure 10 and we have tested). Also, in many cases the differences between the two rules is significant (approximately up to 0.2 in some cases). Since plurality rule always performs better according to Figure 10, we will use this rule in the following.

B Selecting Error Models

Selecting an appropriate model is a standard statistical task. The methodology is the following: Select one set of items for which the values of the items are known (or can be estimated with high accuracy). Give this set to multiple workers and collect their responses. Based on the responses, two steps have to be taken:

1. Determine the optimal parameters that seem to best fit each model [14], and
2. For the optimal parameters, use a standard statistic test (such as the χ^2 -test, the Anderson-Darline test, or the Shapiro-Wilk test) to determine if the responses actually follow the given error model with the optimal parameters [15].

This process should be repeated for various sets of different sizes in order to examine if a model fits the data. Especially for the linear error model, we need sets of different sizes to be tested (one set of a particular size can be handled by the constant error model) in order to find the optimal parameters and examine if this is the model that fits the data.

This process allows the selection of the best fitting error model for the task in mind, especially for high numbers of sets examined and responses sought. The accuracy of the selected model depends on how much money the practitioner wants to spend on this exploratory step (before running the actual max algorithm task) in order to get statistics that allow the discovery of good models.