The Importance of Being Expert: Efficient Max-Finding in Crowdsourcing*

Aris Anagnostopoulos
Sapienza University
Rome, Italy
aris@dis.uniroma1.it

Luca Becchetti
Sapienza University
Rome, Italy
becchetti@dis.uniroma1.it

Adriano Fazzone
Sapienza University
Rome, Italy
fazzone@dis.uniroma1.it

Ida Mele
Max Planck Institute for Informatics
Saarbrücken, Germany
imele@mpi-inf.mpg.de

Matteo Riondato†
Stanford University
Stanford, CA, USA
rionda@cs.stanford.edu

Οὐ πάνυ ἡμῖν οὕτω φροντιστέον τί ἐροῦσιν οἱ πολλοὶ ἡμᾶς, ἀλλ᾽ ὅτι ὁ ἐπαΐων περὶ τῶν δικαίων καὶ ἀδίκων. [Σωκράτης, Πλάτωνος "Κρίτων"]

We should not care so much about what the many say about us, but about what says whoever is expert on what is right and wrong. [Socrates in Plato's "Crito"]

ABSTRACT

Crowdsourcing is a computational paradigm whose distinctive feature is the involvement of human workers in key steps of the computation. It is used successfully to address problems that would be hard or impossible to solve for machines. As we highlight in this work, the exclusive use of nonexpert individuals may prove ineffective in some cases, especially when the task at hand or the need for accurate solutions demand some degree of specialization to avoid excessive uncertainty and inconsistency in the answers. We address this limitation by proposing an approach that combines the wisdom of the crowd with the educated opinion of experts. We present a computational model for crowdsourcing that envisions two classes of workers with different expertise levels. One of its distinctive features is the adoption of the threshold error model, whose roots are in psychometrics and which we extend from previous theoretical work. Our computational model allows to evaluate the performance of crowdsourcing algorithms with respect to accuracy and cost. We use our model to develop and analyze an algorithm for approximating the best, in a broad sense, of a set of elements. The algorithm uses *naïve* and *expert* workers to find an element that is a constant-factor approximation to the best. We prove upper and lower bounds on the number of comparisons needed to solve this problem, showing that our algorithm uses expert and naïve workers optimally up to a constant factor. Finally, we evaluate our algorithm on real and synthetic datasets using the CrowdFlower crowdsourcing platform, showing that our approach is also effective in practice.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; H.3.3 [**Information Storage and Retrieval**]: Information Filtering; H.1.2 [**Models and Principles**]: User/Machine Factors—*Human information processing*

Keywords

Crowdsourcing; human computation; max algorithm; worker models

1. INTRODUCTION

Crowdsourcing is a computational paradigm that enables outsourcing pieces of the computation to humans, who perform them under monetary compensation. The main rationale for the involvement of humans is the existence of tasks that are easy to perform for a person but very difficult or impossible to accomplish for a machine. Crowdsourcing-based systems [13] and algorithms have been developed to answer important queries for which relational DBMSs have traditionally been used, such as selectivity estimation, counting, ranking, and filtering [7, 22, 23, 30]. Crowdsourcing has applications in various areas, including machine learning, visualization, recommendation systems, computational photography, and data analytics.

One of the main reasons for bringing humans into the computational loop is that the task is underspecified or cannot be specified sufficiently in details for a machine to perform it, whereas humans can use intuition and their background knowledge to understand what is requested from them. Even when the job is well specified, humans may perform the task much better than a machine.

Despite the fact that humans can carry out some operations better or more easily than machines, they do not always perform them correctly. Indeed, it has been observed that the output of crowdsourcing systems can be extremely noisy [15, 35]. There are two main sources of error. The first is *incompleteness of information*: tasks may be underspecified in many respects, so that individual

†Part of the work performed while visiting Sapienza University of Rome.

*This research was partially supported by the Google Focused Research Award "Algorithms for Large-Scale Data Analysis," the EU FET projects MULTIPLEX 317532 and SIMPOL 610704, the Italian MIUR PRIN project "ARS TechnoMedia," and by the NSF award IIS-1247581.

factors come into play when humans perform them. In this case, although a ground truth may exist (e.g., cars come with factory prices), humans may err because they possess only partial information and/or their judgment may be clouded by personal biases. The second source of error, or actually set of sources, includes mistakes due to input errors, misunderstanding of the requirements, and malicious behavior (*crowdsourcing spamming*).

To motivate the topic of this work, assume that we ask you which of the two pictures of Figure 1(a) has *fewer* dots; which one would you select? (Try, if you want, before proceeding!) The correct answer is the first one (180 vs. 200). Most humans can answer questions like this thanks to abilities that are either hard-wired or naturally learnt in the course of time. Thus, if we ask several individuals the same question, we expect that the majority will answer correctly, and the more people we ask, the higher is our confidence that the preferred answer is correct. In cases like this we can successfully apply the paradigm of the *wisdom of crowds* [28]. Indeed, it is for this reason that crowdsourcing platforms offer the possibility to ask the same question to several human workers.

Consider now Figure 1(b) and the question "Which car has a *higher* price?" (Try to guess again!), and similarly for Figure 1(c). In both cases the correct answer is the second car. Yet, unless one is an expert on car pricing or has access to accurate information about the cars, she probably cannot figure out why the Mercedes ($114K) is more expensive than the BMW ($100K), yet cheaper than the Audi ($120K)[1]. In this case the wisdom of crowds will not work: it is hard to guess the correct answer when the price difference is small, unless one is a real expert on U.S. car prices or is able to retrieve accurate information about the prices. For this reason, crowdsourcing platforms have started introducing the concept of *experts*. Amazon mechanical Turk now has *masters* and CrowdFlower allows to employ *skilled workers*. Although the concept of an expert is broad, there are three characteristics that expert workers possess in contrast with *naïve* (i.e., nonexpert) workers: (1) they obtain training or pass tests to certify that they produce higher-quality results for specific application domains with respect to regular workers; (2) they are a much scarcer resource than regular workers; (3) they offer their services at a much higher price.

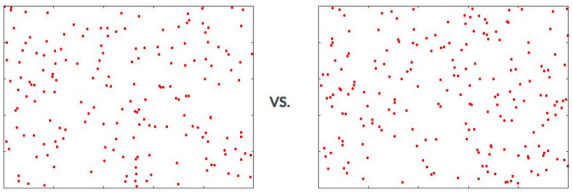
The design and analysis of algorithms that employ human workers on crowdsourcing platforms require computational and cost models that are flexible enough to allow performance analysis and that are realistic enough to reflect the actual runtime and costs of the algorithms. In this respect, two crucial aspects make crowdsourcing different from most other computational paradigms: *human error* and *monetary cost*. The computational and cost models must allow the expression and analysis of these aspects in a realistic way. In addition, as already mentioned, the recent trend in crowdsourcing platforms to organize the available workforce into different classes with different skills demands a suitable modeling of the expert workers, who allow to achieve higher quality results, which cannot be attained by naïve workers but which come at a higher cost.

1.1 Contributions and Roadmap

In this paper we introduce a computational model that captures important aspects of crowdsourcing and present and analyze a max-finding algorithm on top of this model. We introduce the concept of *expert worker*, who can add qualitatively more power to the computations that we are able to perform. We show how expert workers can help in a practical case by presenting an algorithm that leverages on experts to find the maximum among a set of elements. This is a problem often encountered in practice in crowdsourcing scenarios (e.g., ranking of search results, evaluation of web-page relevance to a query, selection of the best labeling for a picture [6]). Whereas this problem is simple enough in the standard computational model, it becomes nontrivial in more complex scenarios as the one we present here. For these reasons, it is one of the most studied problems in the context of crowdsourcing [32, 33] (without considering experts). Our algorithm can be used inside systems like CrowdDB [13] to answer a wider range of queries using the crowd. To summarize our contributions:

- By performing experiments on the CrowdFlower crowdsourcing platform, we identify key characteristics of workers' performance in diverse scenarios (Section 3.1).
- Building on the findings we present models that can capture the behavior in the diverse settings (Section 3.2).
- We introduce the concept of crowdsourcing experts and we incorporate them in our model (Section 3.3).
- Building on our models, we formally define the problem of finding the maximum element. We provide lower bounds on the number of expert and nonexpert comparisons required to solve it. We provide an algorithm for this problem that computes a constant-factor approximation to the maximum ele-

[1] Prices are from August 2013.



(a) Task: select the picture with fewer dots.

(b) Task: select the most expensive car.

(c) Task: select the most expensive car.

Figure 1: Examples of crowdsourcing tasks.

ment and makes an optimal (modulo constant factors) use of the workers (Section 4).
- We perform a series of simulations for evaluating the efficiency of our algorithm in practice complementing our theoretical analysis, and we perform a set of experiments on CrowdFlower to show its effectiveness in finding the maximum (Section 5). Among others, we apply our approach on the problem of evaluation of search results to show the applicability of our approach in a real-life scenario.

2. RELATED WORK

Many works in the algorithmic literature have dealt with the problem of sorting or computing the maximum of a set of elements using comparators some of which may be faulty [26, 34], even without considering the crowdsourcing settings. Various different models and solutions have been proposed. A number of works considered the idea that the comparator errs with some probability at each comparison, independently from other comparisons [3, 8, 10, 14, 16, 17, 33]. They presented algorithms to compute the maximum element, studying numerous variants of this purely probabilistic error model. In the simplest variant, each comparator has a fixed probability associated to it, and this remains constant over all comparisons, independently of the values of the elements. This is for example the case for the basic models considered by Feige et al. [10] and Davidson et al. [8] (in both works, more sophisticated probabilistic models are also considered, but they do not incorporate the concept of experts, crucial in our work). An important consequence of this probabilistic modeling of the error is the following: *If the error probabilities are independent and less than* $1/2$ *then, given two items to compare, independently of their mutual distance, it is possible to identify the element with higher value with arbitrarily high probability by performing the same comparison multiple times, and taking the element that won the majority of the comparisons (or an arbitrary element in case of a tie).* In our work, we consider a different model (the *threshold* model) where such conclusion is not possible: an expert has more capabilities than a naïve worker and her answers *cannot* be simulated by aggregating the answers of multiple naïve workers. Another model is presented by Aigner [1], where at each step of the computation a fraction p of the answers returned so far could be wrong. This setting is in part related to past work on comparisons with faulty memories [11, 12] and more generally to computational tasks involving communication across noisy channels. Pelc [24] presents a survey of this broader area of research. Other works considered a threshold model, where the comparator may err if the elements have values very close to each other [2]. We extend and adapt some of the results from these previous contributions to the crowdsourcing model of computation. None of these consider the main idea of the present work, namely that comparators may belong to different classes with different error parameters and different costs, and one cannot be used to simulate the other.

In the crowdsourcing environment, Marcus et al. [19] looked at how to select the maximal element and sort a sequence of elements by splitting the input into nonoverlapping sets with the same size and sort these sets recursively. No guarantee is given on the running time and accuracy of the algorithms.

Venetis and Garcia-Molina [32] and Venetis et al. [33] present algorithms for finding the maximum in crowdsourcing environments based on static and dynamic tournaments. They consider error models taken from the psychometrics literature and, given a specific error model and a budget of computational resources, they compute the optimal parameters for the algorithms using simulated annealing. They do not discuss, though, about the possibility of having experts and the tradeoffs between accuracy and costs that this possibility would allow.

The need for experts is pointed out by Sun et al. [27]: a single majority vote from all the workers is only accurate for low difficulty tasks, but it is insufficient as the difficulty of tasks increases.

Some works [13, 23**?**] took into account the probability of mistakes and different cost measure to optimize various filtering tasks using crowdsourcing, but without analyzing their effect in a formal framework. We present a different model and algorithm, giving precise guarantees on the performances.

Karger et al. [16] presented an algorithm for crowdsourcing that learns the reliability of each worker and assigns tasks to workers according to their reliability, using this piece of information to minimize the number of questions asked to the workers. Other works [4, 5, 9, 18, 25, 31] presented methods to identify sets of well performing workers or to detect badly performing workers and rule them out, whereas we assume to know precisely who the experts are, as they may reside outside a classical crowdsourcing system. As an example, consider the case where the task requires to select the best picture representing the Colosseum. A professional photographer would be an expert in this case, to whom we ask to solve the task offline. She is hired for the precise reason that she is an expert, hence her status is known in advance. Alternatively, it is possible to apply the algorithms presented in those works to detect a set of experts and then use our algorithm to leverage their additional expertise. In this sense, our work is orthogonal and complementary to that from [4, 5, 9, 18, 25, 31].

Mason and Watts [20] investigated the effect of increasing the financial incentives for workers on the quality of the performed work. They found out that there is no improvement in quality as the incentive grows. This implies that one cannot just pay some workers more than others and use them as experts. Instead, in our model we pay experts more than others for the only reason that they are experts and are going to perform the work with higher precision.

Mo et al. [21] proposed algorithms to compute the number of workers whom to ask the same question such as to achieve the best accuracy with a fixed available budget. The workers all belong to the same class and they perform a task correctly or incorrectly with a probability that depends on the task but not on the workers.

Davidson et al. [8] introduced algorithms inspired by Feige et al. [10] to solve max, top-k, and group-by queries under a new error model where the probability of error when comparing two elements depends on the distance between the elements. The probability of error is the same for all workers, so there is no concept of experts and nonexperts.

3. MODELING CROWDSOURCING

In this section we formalize crowdsourcing computation for computing the maximum (or best) among a set of elements. We perform some crowdsourcing experiments and we use the findings to justify a class of models that we introduce.

Finding the maximum over a set of elements. Let $\mathcal{U}$ be a universe of elements and let $\mathsf{v}(\cdot)$ be a function from $\mathcal{U}$ to the reals: $\mathsf{v} : \mathcal{U} \to \mathbb{R}$. For an element $e \in \mathcal{U}$ we call $\mathsf{v}(e)$ the *value of* e. The function v establishes a partial[2] order over the elements of $\mathcal{U}$. We define the *distance between two elements* $u, v \in \mathcal{U}$ as $\mathsf{d}(u, v) = |\mathsf{v}(u) - \mathsf{v}(v)|$. Given a set L of n elements from $\mathcal{U}$, let $V_L = \max_{e \in L} \mathsf{v}(e)$. We denote with M_L any of the elements from L with value V_L, so that, $\mathsf{v}(M_L) = V_L$ by definition, and we refer to it as the *maximum element of* L. The set L will often be clear from

[2]The order is partial because it is possible to have $\mathsf{v}(e_1) = \mathsf{v}(e_2)$ for $e_1 \neq e_2$.

the context and we will drop it from the notation of M_L and only use M. The problem of interest in this work is the selection of an element $e \in L$ whose value $\mathsf{v}(e)$ is equal to or closely approximates V_L, as formally defined in Section 4.

Human workers and crowdsourcing algorithms. Both the universe $\mathcal{U}$ and the value function v are arbitrary. In particular, v may be very difficult or time consuming to evaluate or even approximate for computers but very easy to evaluate or approximate for humans. In this work we develop a crowdsourcing algorithm to compute an element from L whose value is close to that of the maximum element M, using a set of human *workers* W. We assume that a worker can only compare two elements at a time (this assumption is often present also in prior works as it simplifies the formalization) and returns the one that she believes has the maximum value. Following Venetis et al. [33], the algorithms we consider are organized in *logical time steps*. In the s-th logical step, a batch B_s of pairwise comparisons is sent to the crowdsourcing platform, which, after some time, returns the corresponding answers from the human workers. Depending on these answers, the algorithm selects the next batch B_{s+1} of comparisons, and so on, until the algorithm terminates. Depending on the size of W, each logical step s in general corresponds to a sequence $\mathsf{F}(s)$ of consecutive *physical time steps*. In particular, in the generic physical time step $t \in \mathsf{F}(s)$, a subset $W_t \subseteq W$ of the workers is active. Each active worker $w \in W_t$ receives a pair (k, j) of *distinct*[3] elements from $L \times L$. Worker w then "computes" a *comparison function* $\mathsf{m}_w(k, j)$, which returns the element from k, j that she *believes* has the maximum value. We say that the returned element *wins* the comparison.

Remark. Venetis et al. [33] simply call *steps* what we call *logical steps* here. The reason is that they consider the number of logical time steps (according to our terminology) a reasonable measure of the time complexity.

Given the generality of the function v and of the universe $\mathcal{U}$, the worker w may not be able to compute v exactly, but only to somehow approximate it. Hence, the element returned by the function m_w may not be the one with maximum value. To delve more into this question, we start by performing a set of crowdsourcing experiments, which allow us to observe how humans err in different max-finding tasks and how we can use their collective knowledge.

3.1 Workers' Accuracy in Crowdsourcing

To find the properties required for our models, we performed a series of experiments on the CrowdFlower crowdsourcing platform.[4] We describe first the two main datasets that we created and then we provide some details about the CrowdFlower setup. The task at hand is to ask workers to compare two items. The main question that we want to address with our experiment is: *is it always possible to improve accuracy by asking multiple workers to answer the same question independently, or is there a* cognitive barrier*, at least for certain types of questions, that does not allow to achieve arbitrarily high precision?*

Datasets. We created two main datasets:

- DOTS (inspired by [31]): It consists of a collection of images containing randomly placed dots. The number of dots in each picture ranges from 100 to 1500, with steps of 20.
- CARS: This dataset contains descriptions of cars. We downloaded a set of approximately 5000 new cars from the `cars.com` web site. For each car we collected a *photo*, the *make*, *model*, *body style* (e.g., SUV, sedan, coupe, etc.), *engine informations*, *number of doors*, and its *price* (in August 2013). We cleaned the dataset and created a set of 110 cars with price between 14K and 130K. For every pair of cars the difference in price is at least $500.

  CARS is a very noisy dataset. For instance we found several sets of cars of the same make and model that varied significantly in the price, often in the order of several thousands dollars. Most of the times this difference is a result of the differences in equipment, but sometimes different dealers also sell the same car at different prices. Showing the entire equipment to workers is impractical and would lead to higher error rate, so we decided to show only limited information about each car. We ensured that for the same brand and year the car models are not repeated, by selecting a representative that was in the middle of the price range.

Measuring Workers' Accuracy. We used the CrowdFlower platform, a paid crowdsourcing service, available since 2009. It offers quality-ensured results at massive scale, good APIs, and multiple channels.

For each dataset we used 50 elements for comparisons, and some additional ones for *gold* comparisons, which are comparisons for which the ground-truth value is provided and which are used by CrowdFlower to evaluate the performance of workers and reduce the effect of spam (responses of workers whose performance on gold comparisons has accuracy less than 70% are ignored). In total, 15% of the queries that we performed are gold queries. We selected pairs covering the overall range of values and differences. We submitted 105 pairs from DOTS and 154 from CARS (we found that for the latter we needed more data points). For each pair to be compared we requested at least 21 answers. Figure 1 shows snapshots of the pairs presented to the workers.

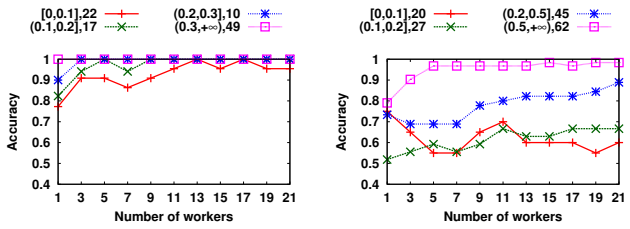
Our goal is twofold. First, we want to measure the accuracy of workers as the difference of the value of the two items under comparison varies. Second, we want to study to what extent we can improve the accuracy by increasing the number of workers.

We summarize our findings in Figure 2. In Figure 2(a), which refers to DOTS, each line corresponds to responses obtained for a given range of difference between the actual number of dots. For example, the red (lowest) line accumulates the responses of queries where the relative difference between the number of dots ranged from 0 to 10% (these are the hardest questions), whereas the green (second lowest) corresponds to the range 10–20%. On the x-axis we vary the number of workers whose (independent) responses we observe (we consider the votes of 1 worker to 21 workers, ordered by time of response) and on the y-axis we report the aggregate accuracy of the workers when we take a *majority vote*. Note that the accuracy is generally quite low when considering a single worker but it improves as we ask more workers, arriving very close to 1. Figure 2(b) shows the same plot for CARS. When the relative difference between the price of the two cars is relatively large, the accuracy approaches 1 as the number of workers increases. However, for smaller differences (up to 20%) the accuracy of the workers plateaus: it does not surpass 0.6 or 0.7, depending on the difference.

The results allow us to draw two conclusions. First, we need different error models to capture the different behaviors that we observed. For the behavior in DOTS we need a model in which the accuracy increases with the number of workers. Instead, for CARS we need a model where, for small values of the difference, an increase to the number of workers is not sufficient to increase the accuracy. This is the topic of the next section.

[3] By distinct we just mean that a worker does not receive two copies of the same element, not that $\mathsf{d}(k, j) \neq 0$.

[4] `http://crowdflower.com`



(a) DOTS dataset (b) CARS dataset

Figure 2: Relative accuracy of responses varying the number of workers for different deltas.

Second, these results show a key difference in the tasks: one of them allows to leverage on innate expertise and collective wisdom (counting dots), whereas the other seems to require acquired knowledge at some point (must know car prices). This observation led us to the introduction and modeling of *experts*, skilled workers who are able to rank elements with close values correctly, but who require a higher monetary compensation. Higher compensation can either reflect an immediate cost, or it can model scenarios in which experts are a scarce resource and thus their time is more valuable. This is the topic of Section 3.3.

3.2 Simple Error Models

There are a number of reasons and scenarios in which a worker may commit a mistake as we mentioned in the introduction, and in the previous section we saw that in different settings these can lead to qualitatively different behavior. We next present the probabilistic model, common in prior works, which can model the behavior in DOTS, but not the one in CARS. Then we present a threshold-based model, which captures the behavior in the CARS experiment.

Probabilistic Error Model. A common approach is to assume that an error occurs with some probability, not necessarily fixed: when a worker is given two elements to compare, she chooses the one with highest value with some probability, and the one with lower value with the residual probability, independently of any other comparison she or other workers might perform [3, 8, 10, 14, 16, 17, 33]. The error probability may depend on the difference of the values of the elements to compare, and grows as the difference shrinks, even though for purposes of analysis a common assumption is that it is fixed and independent from the difference.

Assume that for a given question, the probability of error is $p < 0.5$. Then one can show that if we ask the same question to k workers, the probability that the element with *lower* value receives the *majority* of votes is bounded by $e^{-\frac{(1-2p)^2}{8(1-p)}k}$. It decreases exponentially fast to 0 as k grows, implying that we can get arbitrarily good precision by increasing the number of workers; this is qualitatively the behavior that we observed in Figure 2(a). Of course, one can create instances for DOTS on which we cannot aim to reach accuracy equal to 1: e.g., we can create one image with n dots (for large n) and another with $n+1$ dots. In this case, the probability p of error is essentially 0.5 and it will not decrease no matter how many workers we use.

The CARS experiment exemplifies situations in which the comparison tasks may require an expertise not possessed by unskilled (*naïve* in the following) workers. Hence, it is not possible to achieve arbitrary precision and confidence levels by just aggregating the answers from a suitable number of naïve workers. To capture such scenarios, we developed a generalization of the probabilistic model, which we present next.

Threshold Model. To capture scenarios like those mentioned at the end of the previous section, we consider the *threshold model* $\mathcal{T}(\delta, \varepsilon)$, an extension of the model introduced by Ajtai et al. [2] to formalize the concept of *Just Noticeable Difference* by Weber and Fechner (then generalized by Thurstone [29]). Here, whenever a worker is presented with two elements k, j to compare, she chooses the less valuable one (i.e., errs) with a probability that depends on their distance $\mathsf{d}(k, j)$ as follows: let δ be a nonnegative parameter, which models the *discernment ability* of the workers, and let $\varepsilon \in [0, 1)$ be a *residual error probability*. If $\mathsf{d}(k, j) > \delta$ and (without loss of generality) $\mathsf{v}(k) > \mathsf{v}(j)$, the worker returns k with probability $1 - \varepsilon$ and j with probability ε.[5] Instead, if the two elements have values close to each other ($\mathsf{d}(k, j) \leq \delta$) the worker returns either k or j *completely arbitrarily*. In particular, if asked multiple times to compare k and j, the worker may return k on some occasions and j in others, or always k or j. As a result, when the question is hard, asking a lot of workers does not help to increase the accuracy, which is the behavior observed in CARS.

The parameter δ acts as a distance *threshold*, hence the name of the model. In other words, if $\mathsf{d}(k, j) > \delta$, the worker is able to discriminate between the two elements k and j, but for a number of reasons she may still, with low probability, return the one with the lower value. Modeling this uncertainty when the elements are farther apart than δ allows us to take into account workers input errors, malicious workers, or prior bias that the workers may have.

We say that two elements u, v with $\mathsf{d}(u, v) \leq \delta$ are *indistinguishable*. A set of indistinguishable elements is a set of elements such that each two of them are indistinguishable. Note that, assuming the comparison model we just described, it is impossible to exactly compute the element M of L with maximum value if there is another element $e \in L$ indistinguishable from M.

For simplicity, we assume that if the difference is above the threshold then the probability of error is the fixed value ε. This can be generalized to an error probability that depends on the difference, as in the probabilistic model. In addition, δ can be 0, so the threshold model is a generalization of the probabilistic model.

3.3 Threshold Model with Experts

The experimental findings of Section 3.1 demonstrate that there are types of questions that typically require knowledge or skills that are neither innate nor naturally learnt; these questions exhibit an accuracy barrier which cannot be overcome without involving skilled workers with the required expertise. Here, a skilled worker (*expert* in the following) should be seen as an abstraction. In some scenarios, an expert is an actual skilled worker, with the necessary expertise to perform the required comparison tasks with an accuracy that is not achievable by naïve workers. In other settings, an expert models the extra effort needed to perform a comparison task with higher accuracy, for example, by accessing authoritative, external information sources on the topic of the comparison. Yet another setting is where the naïve worker corresponds, for example, to a machine-learning algorithm and an expert to a human. The main characteristic is that the presence of an actual expert or the acquisition of authoritative information about the topic of a comparison *cannot be simulated by simply increasing the number of naïve workers*. Crowdsourcing platforms have become aware of this necessity and they have introduced the concept of experts.

The extension of the threshold model to capture experts is natural. The workers from W are split into two classes, one of *naïve* workers and one of *expert* workers. Naïve workers follow the threshold model $\mathcal{T}(\delta_\mathrm{n}, \varepsilon_\mathrm{n})$, whereas experts follow $\mathcal{T}(\delta_\mathrm{e}, \varepsilon_\mathrm{e})$, with $\delta_\mathrm{n} \gg$

[5]The probabilistic error model is a special case of the threshold model when $\delta = 0$.

δ_e and $\varepsilon_e \leq \varepsilon_n$ (possibly $\varepsilon_e = 0$). Given that there are now two thresholds, we talk of *naïve-indistinguishable* elements for elements u, v with $\mathsf{d}(u,v) \leq \delta_n$, and of *expert-indistinguishable* if $\mathsf{d}(u,v) \leq \delta_e$. Indeed, under our model, two elements that are expert-indistinguishable are also naïve-indistinguishable.

Remarks. We assume to know in advance whether a worker is an expert or not. This is orthogonal and complementary to the approach taken in previous related works that tackle the problem of finding the experts [4, 5, 9, 18, 25, 31]. Our assumption is intuitive given the definition of an expert worker, which is someone who is hired for the specific reason that she is an expert in the task at hand. As we mention in Section 2, an example of this could be a photographer hired to select the best picture of a monument. We may have thousands of pictures to choose from and given the much higher cost of the professional photographer we want to use the cheap naïve workers to filter out the least interesting ones, so that the photographer only has to look at few of them. Alternatively, one can use the aforementioned algorithms to find a group of experts and then use our algorithm to exploit their additional skills and computational power.

In our model we consider two classes of workers, but a natural extension models multiple classes of workers with different expertise levels, or even a continuous measure of expertise for ranking workers. We leave these extensions as future work.

3.4 Cost Model

We analyze our algorithms with respect to monetary cost. We express the cost of the algorithm as a function of the size of the input $n = |L|$.

The main measure of resource consumption that is usually of interest in crowdsourcing applications is the number of operations performed by workers, as they correspond directly to monetary costs, given that workers are paid for each operation they perform. In the presence of experts, we assume that naïve and expert workers have different costs: experts have an associated cost c_e per operation that is much greater than the cost c_n per operation associated to naïve workers ($c_e \gg c_n$). Therefore, if an algorithm performs $x_e(n)$ expert comparisons and $x_n(n)$ naïve comparisons, the total *monetary cost* of the algorithm is

$$\mathsf{C}(n) = x_e(n) \cdot c_e + x_n(n) \cdot c_n.$$

4. FINDING THE MAXIMUM ELEMENT

In this section we delve into the problem of finding the maximum among a set of elements. Apart from the foundational nature of the problem, we study it for the following reasons: (1) many common crowdsourcing tasks are essentially problems of finding the maximum according to some criterion (e.g., finding the best result to a query, the most relevant ad to a page or query, or the best design among a set of candidates [6]); (2) indeed many past works on crowdsourcing algorithms studied the problem of finding the maximum [19, 32, 33]; (3) it is well specified and amenable to (albeit nontrivial) theoretical analysis. Furthermore, we hope that this work will stimulate the rigorous and analytical study of yet more complicated crowdsourcing problems (e.g., evaluation of classification or other machine-learning algorithms).

Given a multiset L of $|L| = n$ elements from a universe $\mathcal{U}$, our algorithm selects an element $e \in L$ whose value is close to the maximum value among the elements in L. We consider the threshold comparison model with experts. Recall that $M \in L$ is an element with maximum value among those in L. The algorithm finds an element $e \in L$ such that $\mathsf{d}(M,e) \leq 3\delta_e$. Its cost depends on the value $\mathsf{u}_n(n)$, which represents the number of elements in L that are naïve-indistinguishable from M: $\mathsf{u}_n(n) = |\{e : \mathsf{d}(M,e) \leq \delta_n\}|$. We assume that $\mathsf{u}_n(n) = o(n)$, giving power to naïve users to discriminate a large part of the input from the maximum element; this assumption is reasonable for typical crowdsourcing datasets and tasks.

Remark. For the sake of presentation we assume that both residual errors ε_n and ε_e (see Section 3.3). are equal to 0 Our results can be extended to any value less than $1/2$.

4.1 An Expert-Aware Max-Finding Algorithm

The algorithm consists of two phases, summed up in Algorithm 1. In the first phase, it uses naïve workers to filter out the majority of the elements that cannot possibly be the maximum, leaving a small set S of candidate elements containing M. In more detail, in the first phase we solve the following problem:

PROBLEM 1. *Given an initial set L of n elements, return a subset $S \subset L$ of size $O(\mathsf{u}_n(n))$ that contains M, using only naïve workers to perform comparisons.*[6]

Algorithm 1: Find an element close to the maximum M

Input : A set L of n elements, a function $\mathsf{u}_n(n) = o(n)$.
Output: An approximation of the maximum element M in L.

1 Obtain a set $S \subset L$ using Algorithm 2 with naïve workers
2 Return the output of Algorithm 3 with expert workers on input S

In the second phase, we apply a max-finding algorithm to the set S found in the first phase, using only experts to perform the comparisons. More precisely, we solve the following problem:

PROBLEM 2. *Given an input set S of size $O(\mathsf{u}_n(n))$ containing M, return an element e, such that $\mathsf{d}(M,e) = O(\delta_e)$, using only experts and performing as few comparisons as possible.*[7]

One could solve this problem with $\Theta(|S|^2)$ experts comparisons by performing a simple *all-play-all tournament*[8] among the elements in S. To reduce the number of comparisons performed by experts, we instead use one of the algorithms proposed in [2, Section 3] and return the element found by this algorithm. More details about the second phase are given in Section 4.1.2.

4.1.1 First phase

In the first phase we want to solve Problem 1 using only naïve workers and requesting as few comparisons as possible to minimize monetary cost. We outline our algorithmic approach in Algorithm 2. It relies on some combinatorial properties of all-play-all tournaments, which we prove below. In particular, Lemma 1 shows a key property of the maximum element M in all-play-all tournaments.

LEMMA 1. *In an all-play-all tournament among the elements of L, the maximum element M wins at least $n - \mathsf{u}_n(n)$ comparisons.*

[6]The exact value of $|S|$ and the reason for such a choice are motivated in Section 4.1.1. In a nutshell, this choice allows to achieve an asymptotically optimal number of (naïve) comparisons, as shown in Section 4.3.

[7]In practice, $\mathsf{d}(M,e) \leq 3\delta_e$ or $\mathsf{d}(M,e) \leq 2\delta_e$, according to the algorithm used to solve Problem 2; see Section 4.1.2.

[8]In an all-play-all tournament, each element is compared against every other element. This is also sometimes known as a round-robin tournament.

PROOF. By definition of $\mathsf{u_n}(n)$, there are at most $\mathsf{u_n}(n)$ elements e such that $\mathsf{d}(e, M) \leq \delta_\mathrm{n}$, whereas M wins every element e such that $\mathsf{d}(e, M) > \delta_\mathrm{n}$. □

The previous lemma suggests a way (described below) to filter out elements that are certainly *not* the maximum and eventually obtain a set of candidates for further processing by expert workers. In the following lemma (whose proof is deferred to Appendix B) we prove that the size of this set is small. Actually, we prove a slightly more general result that holds for any set of elements playing an all-play-all tournament and any minimum number of wins, and does not depend on the error model.

LEMMA 2. *Let A be a set of elements and let $r < |A|$. In an all-play-all tournament among the elements of A, there are at most $2r - 1$ elements that win at least $|A| - r$ comparisons each .*

Lemmas 1 and 2 combined lead to an efficient algorithm to solve Problem 1, that is, to compute a set $S \subsetneq L$ of size at most $O(\mathsf{u_n}(n))$, such that $M \in S$. In particular, Lemma 1 suggests that we should make sure that S contains all the elements that would win at least $n - \mathsf{u_n}(n)$ comparisons in an all-play-all tournament, otherwise we may miss M. We could easily find S by performing an all-play-all tournament among all elements in L and then picking those that win at least $n - \mathsf{u_n}(n)$ times; this would require $\binom{n}{2} = \Theta(n^2)$ comparisons. With the help of Lemma 2 we can find S more efficiently (i.e., with fewer comparisons) as follows: We partition L into small subsets of size $g = 4\mathsf{u_n}(n)$ (except for one subset, which may have fewer), and then perform an all-play-all tournament within each subset. We discard elements that lose at least $\mathsf{u_n}(n)$ comparisons in the all-play-all tournament of their subset and we keep those that win at least $g - \mathsf{u_n}(n)$ comparisons. In the next level, we partition the set of all surviving elements into subsets of size g and perform all-play-all tournaments within each of the subsets, and so on, until the set of survivors contains fewer than $2\mathsf{u_n}(n)$ elements. The pseudocode of the algorithm is presented as Algorithm 2. In Lemma 3 (proof in Appendix B) we prove its correctness and we show that it requires only $O(n\mathsf{u_n}(n))$ comparisons. Later, in Section 4.3 we prove that this bound is optimal, within constant factors.

LEMMA 3. *Algorithm 2 computes a set S such that $M \in S$ and $|S| \leq 2\mathsf{u_n}(n) - 1$ by performing at most $4n\mathsf{u_n}(n)$ comparisons.*

4.1.2 Second phase

The outcome of the first phase is a set S of size at most $2\mathsf{u_n}(n) - 1$ containing M; the second phase is devoted to solving Problem 2, that is, retrieving M (or a nearby element) from S, using experts to perform comparisons. We have three options to solve Problem 2:

1. Perform an all-play-all tournament on the set S and pick the element e with the most wins; it is guaranteed that $\mathsf{d}(M, e) \leq 2\delta_\mathrm{e}$. This method requires $\Theta(\mathsf{u_n}(n)^2)$ expert comparisons.
2. Use the deterministic algorithm 2-MaxFind [2, Section 3.1] (pseudocode in Algorithm 3); it performs $O(\mathsf{u_n}(n)^{3/2})$ expert comparisons to return an element e such that $\mathsf{d}(M, e) \leq 2\delta_\mathrm{e}$.
3. Use the randomized algorithm from [2, Section 3.2] (pseudocode in Algorithm 5 in Appendix B); this performs $\Theta(\mathsf{u_n}(n))$ expert comparisons and it returns an element e with the guarantee that $\mathsf{d}(M, e) \leq 3\delta_\mathrm{e}$ whp.

Clearly, we do not consider the first option as it is dominated by the second one (assuming that we memorize results and we do not repeat comparisons that we have already performed). For the theoretical analysis we assume that we use the third one. This allows us to obtain asymptotically optimal results in terms of expert comparisons, with the downside that the value returned can be up to $3\delta_\mathrm{e}$ far from the maximum, whp. In practice though it turns out that the second option is superior to the third one for the values of n (and $\mathsf{u_n}(n)$) that we consider: even though the third option is a linear algorithm, the constants are so high that for the values of n of our interest they lead to a much higher cost. The second option has also the advantage that it guarantees to return an element that is closer to the maximum (only $2\delta_\mathrm{e}$ far, the best possible for the model [2]). For this reason we use the 2-MaxFind algorithm for the simulations in Section 5.

Algorithm 2: Find a set of candidates containing the maximum element M

Input : A set L of n elements, a function $\mathsf{u_n}(n) = o(n)$.
Output: A set of size at most $2\mathsf{u_n}(n) - 1$ containing the maximum element M of L.

```
g ← 4u_n(n)
i ← 0
L_i ← L
while |L_i| ≥ 2u_n(n) do
    L_{i+1} ← ∅
    Partition L_i into subsets G_1, ..., G_ℓ of size g (the last one may be smaller)
    forall G_j, 1 ≤ j ≤ ℓ − 1 do
        Perform an all-play-all tournament among the elements of G_j
        Let W_j be the set of the elements of G_j that win at least g − u_n(n) comparisons in the all-play-all tournament.
        L_{i+1} ← L_{i+1} ∪ W_j
    end
    if |G_ℓ| ≤ u_n(n) then
        L_{i+1} ← L_{i+1} ∪ G_ℓ
    else
        Let W_ℓ be the set of the elements of G_j that win at least |G_ℓ| − u_n(n) comparisons in the all-play-all tournament
        L_{i+1} ← L_{i+1} ∪ W_ℓ
    end
    i ← i + 1
end
return L_i
```

For the sake of completeness, we now outline the algorithm and present its pseudocode in Algorithm 3 (see also [2, Section 3.1]). Consider the candidate set S returned by Algorithm 2 and let $s = |S| \leq 2\mathsf{u_n}(n) - 1$. Algorithm 2-MaxFind works by iteratively selecting an arbitrary subset of $\sqrt{s}$ elements and then performing an all-play-all tournament among them. All the elements are then compared to the winner of the tournament (i.e., one of the elements with the highest number of wins). Those that lose against the winner are removed. This process is iterated on the remaining elements until only $\sqrt{s}$ elements are left. A final all-play-all tournament among these elements determines the winner.

4.2 Analysis of the Algorithm

In this section we analyze the correctness and efficiency of the algorithm presented in the previous section. As we mentioned previously, for the purpose of the analysis, we assume that, in the second phase, Algorithm 3 is replaced by its randomized counterpart from [2, Section 3.2] (pseudocode in Algorithm 5 in Appendix B).

Algorithm 3: 2MaxFind: find (approximation of) maximum element M

Input : A set S of s elements.
Output: An estimate of the maximum element M in S.

```
Label all items as candidates
while more than ⌈√s⌉ candidates do
    Pick an arbitrary set of ⌈√s⌉ candidate elements and
      play them in an all-play-all tournament. Let x have the
      most number of wins
    Compare x against all candidate elements and eliminate
      all elements that lose to x
end
Play a final all-play-all tournament among the at most ⌈√s⌉
  elements survived and return the element with the most
  wins
```

Correctness. The following lemma shows the correctness of our algorithm, assuming that we use the randomized algorithm from [2, Section 3.2][9] (i.e., option 3 from the discussion in the previous section).

LEMMA 4. *With probability at least $1-|S|^{-c}$, for any constant c and S large enough, our algorithm returns an element e such that $\mathsf{d}(M,e) \leq 3\delta_{\mathrm{e}}$.*

PROOF. Immediate from the fact that S contains M and [2, Theorem 4]. □

Cost analysis. The following lemma quantifies the cost of our algorithm assuming, as we mentioned, that in the second phase we apply the randomized algorithm in [2, Section 3.2].

LEMMA 5. *Our algorithm performs $O(n\mathsf{u}_{\mathrm{n}}(n))$ naïve and $O((\mathsf{u}_{\mathrm{n}}(n))^{1.7} + (\mathsf{u}_{\mathrm{n}}(n))^{0.6}\log^2 \mathsf{u}_{\mathrm{n}}(n))$ expert comparisons. Accordingly, its monetary cost $\mathsf{C}(n)$ is*

$$\mathsf{C}(n) = O(c_{\mathrm{n}} n\mathsf{u}_{\mathrm{n}}(n) + c_{\mathrm{e}}((\mathsf{u}_{\mathrm{n}}(n))^{1.7} + (\mathsf{u}_{\mathrm{n}}(n))^{0.6}\log^2 \mathsf{u}_{\mathrm{n}}(n))) \ .$$

If we use algorithm 2-MaxFind to perform the second phase, the following theorem follows from Lemma 3 and from [2, Lemma 1]:

THEOREM 1. *There exists an algorithm that computes an element e such that $\mathsf{d}(e,M) \leq 2\delta_{\mathrm{e}}$ and that performs at most $4n\mathsf{u}_{\mathrm{n}}(n)$ naïve comparisons and $2\mathsf{u}_{\mathrm{n}}(n)^{3/2}$ expert ones.*

4.3 Lower Bounds

In this section, we study the inherent complexity of Problems 1 and 2, into which we have divided the task of the maximum element of a set: (1) identifying a small candidate set containing the maximum using cheap, naïve workers and (2) selecting the maximum or a nearby element out of S, using experts to perform comparisons.

We start with Problem 2, which is easier to analyze. It is simple to conceive instances of the problem for which $\mathsf{u}_{\mathrm{n}}(n)$ elements are naïve indistinguishable from the maximum. This implies that the number of expert comparisons required are in the worst case $\Omega(\mathsf{u}_{\mathrm{n}}(n))$. Actually, it is possible to prove stronger results. The following theorem follows from [2].

LEMMA 6. *Any deterministic algorithm that computes an element e such that $\mathsf{d}(e,M) \leq 2\delta_{\mathrm{e}}$ must perform at least $\Omega(\mathsf{u}_{\mathrm{n}}(n)^{4/3})$ expert comparisons.*

[9]An analogous result holds deterministically if we use the 2-MaxFind algorithm.

Next, we turn our attention to Problem 1. Here, we prove in Corollary 1 below that the number of comparisons performed by Algorithm 2 in Phase 1 is optimal. To prove this corollary, we first show the key fact that, if we want to identify a subset of elements containing the maximum out of the initial set L of size n using naïve workers, we need to identify all elements that win at least $n - \mathsf{u}_{\mathrm{n}}(n)$ comparisons in an all-play-all tournament among the elements of L, because any of them could be the maximum.

LEMMA 7. *Let C be any set of comparisons by naïve workers. If there exists an element e that takes part in at most $\mathsf{u}_{\mathrm{n}}(n)$ comparisons in C, then there exists an assignment of values to the elements such that (1) the assignment of values is compatible with the outcomes of C and (2) e is the element with maximum value.*

PROOF. Let element e have the maximum value, and let E_1 be the set of (at least $n - \mathsf{u}_{\mathrm{n}}(n)$, by Lemma 1) elements that the maximum wins and E_2 the other (at most $\mathsf{u}_{\mathrm{n}}(n)$) elements. We construct the following instance (see Figure 8 in the appendix). We arrange the elements in E_1 at distance about $1.5\delta_{\mathrm{n}}$ from e such that they are distinct (say, we arrange them evenly in an interval of length $0.1\delta_{\mathrm{n}}$, centered at distance $1.5\delta_{\mathrm{n}}$ from element e). We arrange the elements in E_2 in a similar way at distance $0.8\delta_{\mathrm{n}}$ from element e. This is a valid instance for the model because there are at most $\mathsf{u}_{\mathrm{n}}(n)$ elements with distance at most δ_{n} from the maximum, and the maximum wins against all the elements at distance more than δ_{n}. With the exception of e, all the other elements are at a distance δ_{n} from each other, so any comparison result among them is compatible with the results in C. □

As a corollary, we obtain the fact that if we want to return a small set of candidates for the maximum, then we must make at least $\Omega(n\mathsf{u}_{\mathrm{n}}(n))$ comparisons. This number of comparisons is required even if the algorithm returns a large (up to a constant fraction) set.

COROLLARY 1. *Any algorithm that uses only comparisons by naïve workers, and that returns a set S that is guaranteed to contain the maximum and such that $|S| \leq n/2$, must perform at least $n\mathsf{u}_{\mathrm{n}}(n)/4$ comparisons.*

PROOF. The algorithm returns the set S with candidates for the maximum, so it deduces that no element in the set $L \setminus S$ is the maximum. By Lemma 7, each of the elements in $L \setminus S$ must take part in at least $\mathsf{u}_{\mathrm{n}}(n)$ comparisons (otherwise it is impossible to deduce for sure that it is not the maximum). We have $|S| \leq n/2$, therefore $|L \setminus S| \geq n/2$. Each comparison involves two elements, so the required number of comparisons is at least $n\mathsf{u}_{\mathrm{n}}(n)/4$. □

4.4 Estimation of $\mathsf{u}_{\mathrm{n}}(n)$

A nice feature of the algorithm is that it only requires one parameter, namely $\mathsf{u}_{\mathrm{n}}(n)$, which we assume to be $o(n)$. Overestimating this parameter can only harm in cost but not in accuracy. We next show how we can estimate an upper bound to $\mathsf{u}_{\mathrm{n}}(n)$ using a *training set*, that is, a set of $\hat{n}$ elements of which we know the one with highest value (we denote this element with $\hat{M}$). Training data like this (or even richer; for example, assuming that the results of all pairwise comparisons are known) are typically used in crowdsourcing platforms to evaluate the workers and are sometimes referred to as *gold data*.

First, we note that without additional assumptions on the model it is impossible to estimate the value of $\mathsf{u}_{\mathrm{n}}(n)$. Indeed, the model allows for the workers to reply correctly when the difference in the values of the elements presented to them is under the threshold δ_{n}, thus providing no information about δ_{n}. Therefore we are required to perform some additional assumptions:

Assumption 1. We assume that the training set reflects the statistical properties of the actual dataset we will work with. In particular, we assume that the number of items within distance δ_{n} from M is distributed like the number of items within distance δ_{n} from $\hat{M}$, the element with maximum value in the training set, so that $\frac{n}{\hat{n}}\mathsf{u}_{\mathrm{n}}(\hat{n})$ is a reasonable estimator of $\mathsf{u}_{\mathrm{n}}(n)$.

Assumption 2. We assume that, for comparisons between elements whose difference in value is below the (naïve) threshold, workers err with some probability $p_{\mathrm{err}} > 0$. We assume that the empirical error rate in the training set (i.e., the fraction of times that the workers select the wrong element in a comparison) is a good estimator for p_{err}.

For example, if we look at Figure 2(b), we might assume $p_{\mathrm{err}} \approx 0.4$ for accuracies up to 20%. We further assume that comparisons performed by different workers or involving different item pairs are statistically independent. This assumption, except for being required as we noted previously, seems to be reasonable. Indeed, if a worker is being tested and he never errs then we can consider him an expert!

We remark that the assumptions are only needed to estimate $\mathsf{u}_{\mathrm{n}}(n)$; the algorithms discussed in Section 4 are completely oblivious to these assumptions. We also stress once more that overestimating $\mathsf{u}_{\mathrm{n}}(n)$ has no impact on the correctness of the algorithm or on the quality guarantees, only on the cost.

Below we describe the procedure to estimate $\mathsf{u}_{\mathrm{n}}(n)$ as Algorithm 4. Note that this algorithm does not assume any knowledge of the naïve threshold δ_{n}. In this algorithm, c is a suitable constant that is used to tune the level of confidence with which we return an upper bound on $\mathsf{u}_{\mathrm{n}}(n)$.

Algorithm 4: Estimating $\mathsf{u}_{\mathrm{n}}(n)$ from training data.

Input : Training set containing $\hat{n}$ elements.
Output: An estimate of $\mathsf{u}_{\mathrm{n}}(n)$.

$\#errors \leftarrow 0$
forall x *in training set* **do**
  Ask a worker to compare pair $(x, \hat{M})$.
  if *the worker made an error* **then**
    $\#errors \leftarrow \#errors + 1$
  end
end
return $\frac{n}{\hat{n}} \max\left\{c \ln n, \frac{2(\#errors)}{p_{err}}\right\}$

Next we show that, under Assumption 2, we have

$$\mathsf{u}_{\mathrm{n}}(\hat{n}) \leq \max\left\{c \ln n, \frac{2(\#errors)}{p_{\mathrm{err}}}\right\}$$

whp, which, using Assumption 1, implies that the above procedure returns an upper bound for $\mathsf{u}_{\mathrm{n}}(n)$.

If $\mathsf{u}_{\mathrm{n}}(\hat{n}) < c \ln n$ we are done, so in the following we assume that $\mathsf{u}_{\mathrm{n}}(\hat{n}) \geq c \ln n$. Because there exist $\mathsf{u}_{\mathrm{n}}(\hat{n})$ elements in the training set that workers can confuse with the maximum, we have

$$\mathbf{E}\left[\#errors\right] = p_{\mathrm{err}}\mathsf{u}_{\mathrm{n}}(\hat{n}).$$

Then, under Assumption 2 above, an application of a Chernoff bound [**?** , Section 1.6] implies that:

$$\mathbf{P}\left(\#errors < \frac{p_{\mathrm{err}}}{2}\mathsf{u}_{\mathrm{n}}(\hat{n})\right) < e^{-\frac{p_{\mathrm{err}}}{8}\mathsf{u}_{\mathrm{n}}(\hat{n})} \leq n^{-\frac{cp_{\mathrm{err}}}{8}},$$

which implies that whp we have that $\mathsf{u}_{\mathrm{n}}(\hat{n}) \leq \frac{2(\#errors)}{p_{\mathrm{err}}}$, and proves our claim.

Notice that we now need to estimate the value p_{err}. We next discuss how this value is much easier to estimate than $\mathsf{u}_{\mathrm{n}}(n)$ if we assume our model reflects workers' behavior. To this purpose, we consider a subset of element pairs from our training set and we compare them using naïve workers, assigning each pair to multiple workers for comparison. For a given pair, if there is consensus among the workers it was assigned to, we take this as an indication that the difference in the values between the two elements of the pair is at least δ_{n}, up to a residual probability that decreases exponentially in the number of workers, following the same arguments as in Section 3.2. On the other hand, for pairs in which the values of the two elements to compare differ by less than δ_{n}, the error probability on these pairs is exactly p_{err}, and a simple application of a Chernoff bound shows that, with high probability, we can achieve an accurate estimate of p_{err} with a small number of workers. For issues concerning the estimation of the value p_{err} in real life and for a more general discussion, we refer the reader to Appendix A.

5. EXPERIMENTS

To evaluate the efficiency of our algorithm we performed a series of simulations, which we present next. In Section 5.1 we perform experiments to compare the accuracy of our algorithm with simpler ones. In Section 5.2 we measure the loss in accuracy and cost if we err in the estimation of $\mathsf{u}_{\mathrm{n}}(n)$. Finally, in Section 5.3 we show how our algorithm performs when using real workers from the CrowdFlower platform, first in two controlled applications and then on the more realistic problem of evaluating search results.

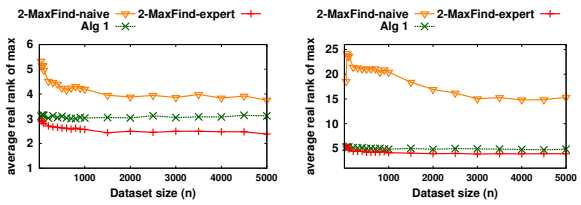
We study the performance of the algorithms both on randomly and on adversarially generated inputs. For the former, we selected n random values independently and uniformly at random from a range. We experimented with various values for the parameters n, δ_{n}, and δ_{e}; the last two, in particular, define the values of $\mathsf{u}_{\mathrm{n}}(n)$ and $\mathsf{u}_{\mathrm{e}}(n)$, respectively. When a worker is asked to rank a pair of elements whose value difference is below her threshold (δ_{n} or δ_{e}), each element is chosen as the answer with probability 1/2.

The adversarial data were created so as to maximize the number of comparisons of 2-MaxFind algorithm. Specifically, in all the comparisons of step 4 of Algorithm 3, whenever the difference is below the threshold, we make element x lose, such as to maximize the number of elements that go to the next round. Furthermore, whenever the algorithm compares two elements whose values have a difference smaller than the threshold, the response is such that it maximizes the number of comparisons of the algorithm. For our algorithm we considered the upper bound predicted by the theory, even though the actual bound may be lower.

5.1 Comparison of Accuracy and Cost

Our algorithm is optimal asymptotically in the sense that it minimizes both naïve and expert comparisons (see Section 4). In this section, we compare our algorithm against two other approaches, (1) 2-MaxFind [2] (see also Section 4.1.2) when it uses only naïve users and (2) 2-MaxFind when it uses only expert users. We will refer to the former approach as 2-MaxFind-naïve and to the latter as 2-MaxFind-expert. In particular, we compare the three approaches with respect to their cost and accuracy. By *accuracy* we mean the rank of the element returned. If the rank is 1 then we have perfect accuracy, and the higher is the rank the lower is the accuracy.

We start by comparing the accuracy of the three approaches. In Figure 3 we depict the true rank of the element returned for each of them. As expected, we can observe that the best approach is 2-MaxFind-expert, with our Algorithm following closely, whereas 2-MaxFind-naïve returns an element with a much lower rank, which worsens as $\mathsf{u}_{\mathrm{n}}(n)$ increases.

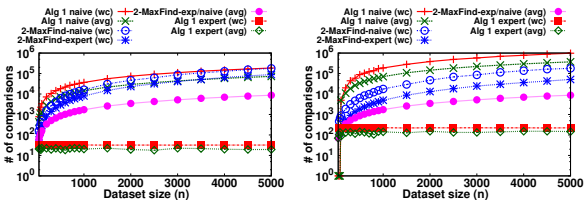


(a) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (b) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

Figure 3: Accuracy as a function of n for different values of $\mathsf{u}_\mathrm{n}(n)$ and $\mathsf{u}_\mathrm{e}(n)$. We compare our approach (Alg 1) with the 2-MaxFind algorithm that uses either only expert or only naïve users.

Thus, unless $\mathsf{u}_\mathrm{n}(n)$ is very small, which would mean that the naïve users perform very well or that the dataset is small, if we require high accuracy we need to resort either to our algorithm or to 2-MaxFind-expert. Which one is preferable depends also on the cost, which we explore next.

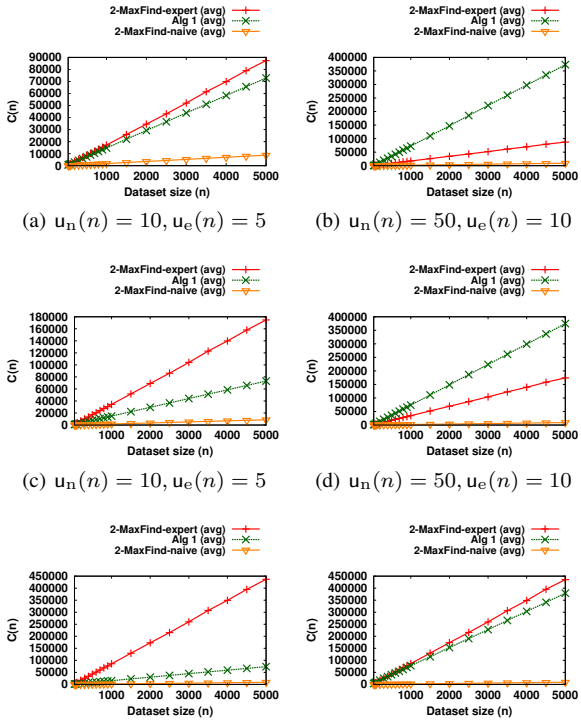
We next compare the number of naïve and expert comparisons of the three approaches in Figure 4, on average and in the worst case. First note how much smaller is the number of expert comparisons for our algorithm; it only depends on the leftover set, and is expected to stay constant as n grows. On the other hand, this comes at a price. We now perform a high number of naïve comparisons, actually higher than the number of expert comparisons that 2-MaxFind-expert performs, in the average case. If the cost of an expert and of a naïve worker is comparable (obviously the exact values will depend on the platform and on the application), then the cost of our algorithm will be higher than 2-MaxFind-expert, and it is better to simply use the latter. On the other hand, in settings where expert comparisons have a much higher cost than naïve comparisons, our algorithm leads to a significantly lower cost. In particular, in the case where naïve comparisons are performed by machines and expert ones by humans, the cost savings can be tremendous. Similarly, in cases where experts are a limited resource, as is often the case when the experts are hired outside of the crowdsourcing platform, it may be a necessity to use cheaper naïve users to prefilter the items before eventually resorting to experts.



(a) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (b) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

Figure 4: Number of comparisons as a function of n for two values of $\mathsf{u}_\mathrm{n}(n)$ and $\mathsf{u}_\mathrm{e}(n)$. Note that the y-axis is on a log scale. We depict the number of expert and naïve comparisons for our algorithms, the number of expert comparisons of 2-MaxFind-expert and the number of naïve comparisons of 2-MaxFind-naïve. In each, we present the number of comparisons in both the average-case and the worst-case scenario. The average number of expert comparisons of 2-MaxFind-expert and the average number of naïve comparisons of 2-MaxFind-naïve are very close to each other, and we depict them with a single curve.

To quantify the savings in cost, we apply the model of Section 3.4, with unit cost for c_n and various values for c_e. In Figure 5 we observe how the average cost increases in the input size, for various parameters. Figure 9 contains the same information for the worst-case cost, calculated as we mentioned previously. We



(a) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (b) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

(c) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (d) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

(e) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (f) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

Figure 5: Average cost as a function of n with $c_\mathrm{n} = 1$ and different values of c_e, $\mathsf{u}_\mathrm{n}(n)$ and $\mathsf{u}_\mathrm{e}(n)$. We have $c_\mathrm{e} = 10$ (row 1), 20 (row 2), 50 (row 3). We compare our approach (Alg 1) with the 2-MaxFind algorithm that uses either only expert or only naïve users.

can observe that the savings of our two-phase algorithm become higher as the the ratio $c_\mathrm{e}/c_\mathrm{n}$ of the expert cost over the naïve cost becomes higher. Notice that the average cost of 2-MaxFind-expert is lower in cases in which the ratio $c_\mathrm{e}/c_\mathrm{n}$ is relatively small (but the worst-case cost is still comparable or higher). In such cases, it is probably beneficial to use the 2-MaxFind-expert—the exact scenarios depend on the values of $\mathsf{u}_\mathrm{n}(n)$, $\mathsf{u}_\mathrm{e}(n)$, and the aforementioned ratio. In particular, we found that if the ratio is less than 10, then our algorithm has a higher cost in the average case. As the cost of an expert worker becomes much higher than the cost of a naïve one, the savings can become tremendous. A typical case for these scenarios is when we would use a crowdsourcing service as a provider for naïve workers, and use external professionals as experts, as we expect to be the case in the application of evaluation of search results that we describe in Section 5.3. The ratio can be even higher when a naïve worker corresponds to a machine-learning algorithm, whereas the expert is a human professional.

The overall conclusion of this set of experiments is that unless the cost of an expert is comparable to the cost of a naïve worker (less than 10 times more expensive), we can achieve great cost savings by our approach taking advantage of both naïve and expert users.

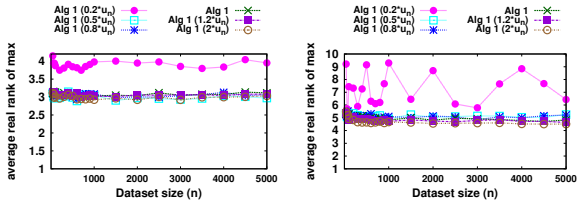
5.2 Under/Over Estimation of $\mathsf{u}_\mathrm{n}(n)$

In this section we analyze the effects of inaccurate estimation of parameter $\mathsf{u}_\mathrm{n}(n)$ on the accuracy and cost of our algorithm.

To this end, we vary the input size and we measure the accuracy as the average real rank of the maximum element returned by the algorithms we consider, where the average is taken over a set of randomly generated instance of the given size, for the three ap-

proaches that we considered in the previous section: our algorithm (Alg 1), 2-MaxFind-expert, and 2-MaxFind-naïve. We describe the inaccuracy in the estimation of $\mathsf{u}_\mathrm{n}(n)$ by the *estimation factor*, defined as the ratio between the estimated and the true value of $\mathsf{u}_\mathrm{n}(n)$. We consider the values $\{0.2, 0.5, 0.8, 1, 1.2, 2\}$ for the estimation factor.

The effects of inaccurate estimation of $\mathsf{u}_\mathrm{n}(n)$ on the accuracy are summarized in Figure 6. As expected, when $\mathsf{u}_\mathrm{n}(n)$ is overestimated the results do not worsen w.r.t. the case that we have the exact value of $\mathsf{u}_\mathrm{n}(n)$ because in the end of the first phase of the algorithm we end up with a larger set (though this comes at a higher cost). When $\mathsf{u}_\mathrm{n}(n)$ is underestimated, the accuracy of our algorithm potentially decreases, because it could return an empty set of elements or a set of elements that does not contain the real max. In practice, we find that the results are not much worse compared to the case that the underestimation is not very large. For instance, if the estimation factor is 0.8 then the set returned in the first round contains the real max in 99% of the times, whereas for an estimation factor of 0.5 results start to worsen with the max appearing in 82% of the sets. When the estimation factor drops to 0.2 the number of times the maximum arrives in the second round is only 38%. Yet the element recovered is not very bad, as we see in Figure 6.



(a) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (b) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

Figure 6: Accuracy as a function of n for different values of $\mathsf{u}_\mathrm{n}(n)$ and $\mathsf{u}_\mathrm{e}(n)$. We compare the accuracy of our 2-phase algorithm for various values of the estimation factor.

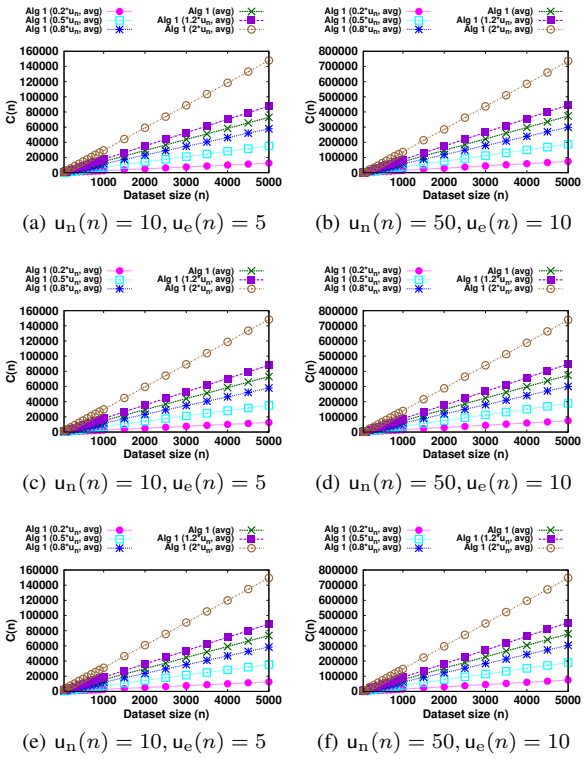
Next we examine the effect on the cost. figures 7 and 10 depict the change of the cost for different estimation factors as we vary the dataset size. As in the Section 5.1 we consider both the average and the worst-case datasets. Figure 7 shows that the cost has a smooth linear behavior; for instance, an estimation factor of 2 doubles the cost both in the average and the worst case.

As a general conclusion, a small overestimate or underestimate of the value $\mathsf{u}_\mathrm{n}(n)$ does not affect radically, neither the accuracy, nor the cost of our algorithm, at least for the datasets that we considered, indicating a robust behavior of our algorithm.

5.3 Experiments on CrowdFlower

We finally conducted a series of experiments on CrowdFlower, whose goal was twofold: (1) Assessing the ability of our algorithm to compute a value close to the maximum; (2) investigating the accuracy of a naïve-only approach based on 2-MaxFind on real problem instances. The first two problems are somewhat artificial, but were chosen as they admit objective, numerical answers, which allow the evaluation of our approach. The third problem we considered is representative of a class of potential applications in which the presence of experts may make a difference.

Settings. We used the datasets DOTS and CARS described in Section 3.1. For each dataset we conducted two identical experiments, namely, they have equal configuration and receive the same input data. From each dataset we downsampled a set of $n = 50$ elements. This volume of data allows to draw conclusions on the performance of the algorithm on real data at reasonable cost. The real data we



(a) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (b) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

(c) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (d) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

(e) $\mathsf{u}_\mathrm{n}(n) = 10, \mathsf{u}_\mathrm{e}(n) = 5$ (f) $\mathsf{u}_\mathrm{n}(n) = 50, \mathsf{u}_\mathrm{e}(n) = 10$

Figure 7: Average cost as a function of n with $c_\mathrm{n} = 1$ and different values of c_e, $\mathsf{u}_\mathrm{n}(n)$ and $\mathsf{u}_\mathrm{e}(n)$. We have $c_\mathrm{e} = 10$ (row 1), 20 (row 2), 50 (row 3). We compare the cost of our 2-phase algorithm for various values of the estimation factor.

used suggested choosing a value $\mathsf{u}_\mathrm{n}(n) = 5$ for Algorithm 2. We used the CrowdFlower workers to perform the naïve comparisons.

CrowdFlower does not provide a service of experts for the problem that we address, so we leveraged the wisdom-of-crowd effect [28], simulating each expert query by 7 naïve queries and selecting the answer that received most votes, just as we did in Section 3.1. As we saw there, this approach is effective in the DOTS experiment but not in the CARS one. This points to the dependency of the crowd's performance on the type of expertise required for the task at hand. We report the results for this latter experiment (CARS) as well, as they provide further evidence for the findings of Section 3.1.

Experiments on DOTS. This experiment was inspired from previous work [31]. We used images of random dots extracted from DOTS. In particular, we used 80 images of random dots (50 images are used for building the dataset and 30 for the golden set, used for gold comparisons). The golden set associated to this dataset is formed by images with a number of dots from 200 to 800 with step 20. The two sets have no elements in common. We performed two experiments asking the users to select the image with the minimum number of random dots.

The final results were almost perfect. In both experiments, nine elements passed to the second phase, and in both they were the real top-9 elements.

From the discussion in Section 3.1 about the findings presented in Figure 2(a), for the DOTS experiment we can simulate expert workers by using several naïve ones. Indeed, that is what we did for the second phase of our algorithm. Since CrowdFlower does not provide a service of experts for the problem that we address, we

leveraged the wisdom-of-crowd effect [28], simulating each expert query by 7 naïve queries and selecting the answer that received most votes, just as we did in Section 3.1. As we saw there, this approach is effective in the DOTS experiment.

In Table 1 we can see a summary of the results of the second phase, which confirm the conclusions of Section 3.1: The second-phase (simulated) experts were able to always find the minimum, and, further, the output correctly orders the top-9 elements, except in one case for one of the experiments, in which the top-6 and top-7 elements are swapped. This verifies that this is an example of a question where using expert users is clearly not recommended.

# dots	Exp. 1	Exp. 2
100	1	1
120	2	2
140	3	3
160	4	4
180	5	5
200	7	6
220	6	7
240	8	8
260	9	9

Table 1: The ranking of the last round of the two DOTS experiments.

Indeed, performing the two-level approach for this problem is an overkill. The results of the 2-MaxFind algorithm with CrowdFlower workers are almost equally as good. We repeated the algorithm 14 times in which in all but one case the correct instance was returned.

Experiments on CARS. In this experiment we wanted to study the quality of the results when the data are much more fuzzy. The goal is to perform queries to find the car that is most highly priced.

Again we conducted two experiments. The results here are also very good. Again nine cars passed to the second round and in both cases the most expensive car passed to the second round even though in this case some of the top-ranked cars did not pass to the second round. As with the DOTS experiment, we attempted to simulate an expert by performing 7 naïve queries. The conclusions of Section 3.1 about Figure 2(b) suggest that in this case this approach will not work, and here we want to check this hypothesis.

We can see the ranking of the last round in Table 2 in Appendix C. Note that, as we mentioned, the top car always reaches the last round. However, in contrast with DOTS, the simulated experts are not able to identify it, indicating the need for real experts for this task. Furthermore, note that because of the fuzziness of the data some cars far from the top-10 arrive at the second phase.

We obtained the results for 14 executions of 2-MaxFind for this problem as well, but the algorithm performed poorly. In none of the executions was the real minimum returned. Instead most of the times the fourth-ranked car was given as a solution, and sometimes even cars not in the top-10. Clearly a truly informed expert opinion is required in this case.

Evaluation of Search Results. In the paragraphs above, we considered two rather artificial application examples that admit objectively correct answers, allowing us to test accurately the model and our algorithm with real workers. One though would expect that if we are interested in applying our approach to real life, we should assume that in many cases an objective answer might not exist (otherwise we would probably not resort to humans), yet it might still be possible to apply the same technique.

In the third set of experiments we considered a more realistic scenario, addressed often by search-engine companies, namely, comparing query results. Typically this is performed by human judges, who are experts in identifying the most relevant result for a given query and in a given context (e.g., user profile)—but even using this approach the inter-agreement among the judges is not perfect.

The objective evaluation of such a problem is far from trivial, as results depend on the particular user, context, and history. To alleviate this problem we considered two specific queries from the area of approximation algorithms: "asymmetric tsp best approximation" and "steiner tree best approximation." The reason for selecting such queries is twofold: (1) We believe that there is a clear best result for the majority of the searches—the paper or a link that contains the current (recently published) best result; (2) We can find real human experts (researchers in the area of algorithms) who can evaluate the results.

For each of the queries we obtained 50 results from Google, distributed uniformly among the top-100 results; this allowed us to have results that are relevant to the queries in different extents. We applied our two-phase algorithm considering workers from CrowdFlower as naïve and researchers in the area of algorithms as experts, who informed us about the best (and recent—within 5 year) approximation algorithms for the two problems. We experimented with values of $\mathsf{u}_{\mathrm{n}}(50) = 6, 8, 10$. In both queries and for all these values of $\mathsf{u}_{\mathrm{n}}(50)$ the maximum was promoted to the second round (and the experts identified it, of course).

Finally, we applied the 2-MaxFind algorithm, using only naïve users. In detail, for each query we executed two runs of 2-MaxFind on CrowdFlower, for a total of four independent runs. Results show that the naïve-only approach was not satisfying: naïve users were able to identify the best result only in one of the four cases (one run over the asymmetric TSP instance). This suggests that a naïve-only approach may not be sufficiently reliable for this type problem.

6. CONCLUSIONS

In this paper we defined computational and cost models for crowdsourcing, formalizing the idea of using workers with different expertise levels. Our main conclusion is that there are two types of tasks, tasks in which the wisdom-of-crowds paradigm holds, in which case we can simulate an expert by the use of several non-expert users, and tasks in which it does not hold and in which there is a necessity for real expertise. To model these two types of applications we considered different error models. A novelty of our work is the definition of the *threshold error model with experts* and it is applicable precisely when we need workers with different expertise levels. We used these models to develop and analyze an algorithm for approximate max-finding in these settings. The algorithm uses naïve workers to filter out the majority of the elements, and then asks the expert workers to focus on a restricted set. We also provide lower bounds to the number of comparisons required to perform the task. Our experimental evaluation of the models and the algorithm on synthetic and real-world problems using the CrowdFlower platform shows that the former are realistic and the latter performs well in practice. In particular, it shows that for some applications simple crowdsourcing approaches can lead to erroneous results, and in these cases the use of real experts is of paramount importance.

7. ACKNOWLEDGEMENTS

We would like to thank CrowdFlower for granting us an academic license, allowing us to perform experiments with a reduced cost. We also want to thank Evgeniy Gabrilovich for useful feedback on the experiments. Finally, the anonymous reviewers have provided us with a lot of constructive comments, which have improved the paper with respect to both content and presentation.

8. REFERENCES

[1] M. Aigner. Finding the maximum and minimum. *Discrete Applied Mathematics*, 74(1):1 – 12, 1997. ISSN 0166-218X. doi: 10.1016/S0166-218X(96)00012-1. URL http://www.sciencedirect.com/science/article/pii/S0166218X96000121.

[2] M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson. Sorting and selection with imprecise comparisons. In *Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 37–48, 2009.

[3] S. Assaf and E. Upfal. Fault tolerant sorting networks. *SIAM Journal on Discrete Mathematics*, 4(4):472–480, 1991. doi: 10.1137/0404042. URL http://epubs.siam.org/doi/abs/10.1137/0404042.

[4] A. Bozzon, M. Brambilla, S. Ceri, M. Silvestri, and G. Vesci. Choosing the right crowd: Expert finding in social networks. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 637–648, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1597-5. doi: 10.1145/2452376.2452451. URL http://doi.acm.org/10.1145/2452376.2452451.

[5] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask?: Jury selection for decision making tasks on micro-blog services. *Proc. VLDB Endow.*, 5(11):1495–1506, July 2012. ISSN 2150-8097. doi: 10.14778/2350229.2350264. URL http://dx.doi.org/10.14778/2350229.2350264.

[6] CSE 2010. *Proceedings of the 1st SIGIR Workshop on Crowdsourcing for Information Retrieval (CSE 2010)*, 2010.

[7] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy. Crowd-powered find algorithms. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 964–975, March 2014. doi: 10.1109/ICDE.2014.6816715.

[8] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 225–236, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1598-2. doi: 10.1145/2448496.2448524. URL http://doi.acm.org/10.1145/2448496.2448524.

[9] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux. Pick-a-crowd: Tell me what you like, and i'll tell you what to do. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 367–374, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2035-1. URL http://dl.acm.org/citation.cfm?id=2488388.2488421.

[10] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. doi: 10.1137/S0097539791195877. URL http://epubs.siam.org/doi/abs/10.1137/S0097539791195877.

[11] I. Finocchi and G. F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 101–110. ACM, 2004.

[12] I. Finocchi, F. Grandoni, and G. F. Italiano. Designing reliable algorithms in unreliable memories. *Computer Science Review*, 1(2):77 – 87, 2007. ISSN 1574-0137. doi: 10.1016/j.cosrev.2007.10.001. URL http://www.sciencedirect.com/science/article/pii/S1574013707000202.

[13] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: Answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4. doi: 10.1145/1989323.1989331. URL http://doi.acm.org/10.1145/1989323.1989331.

[14] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 385–396, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1247-9. doi: 10.1145/2213836.2213880. URL http://doi.acm.org/10.1145/2213836.2213880.

[15] P.-Y. Hsueh, P. Melville, and V. Sindhwani. Data quality from crowdsourcing: a study of annotation selection criteria. In *Proceedings of the NAACL HLT 2009 workshop on active learning for natural language processing*, pages 27–35. Association for Computational Linguistics, 2009.

[16] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors, *NIPS*, pages 1953–1961, 2011.

[17] D. R. Karger, S. Oh, and D. Shah. Budget-optimal crowdsourcing using low-rank matrix approximations. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 284–291. IEEE, 2011.

[18] H. Li, B. Zhao, and A. Fuxman. The wisdom of minority: Discovering and targeting the right group of workers for crowdsourcing. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 165–176, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2744-2. doi: 10.1145/2566486.2568033. URL http://dx.doi.org/10.1145/2566486.2568033.

[19] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *Proc. VLDB Endow.*, 5(1):13–24, Sept. 2011. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=2047485.2047487.

[20] W. Mason and D. J. Watts. Financial incentives and the "performance of crowds". *SIGKDD Explor. Newsl.*, 11(2):100–108, May 2010. ISSN 1931-0145. doi: 10.1145/1809400.1809422. URL http://doi.acm.org/10.1145/1809400.1809422.

[21] L. Mo, R. Cheng, X. S. Yang, C. Ren, S. Lei, E. Lo, B. Kao, and D. W. Cheung. Optimizing task assignment for crowdsourcing environments. Technical Report HKU CS TR-2013-01, University of Hong Kong, 2013.

[22] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. Technical report, Stanford University, 2014. URL http://ilpubs.stanford.edu:8090/1078/.

[23] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 361–372, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1247-9. doi: 10.1145/2213836.2213878. URL http://doi.acm.org/10.1145/2213836.2213878.

[24] A. Pelc. Searching games with errors – fifty years of coping with liars. *Theoretical Computer Science*, 270(1):71–109, 2002.

[25] A. Ramesh, A. Parameswaran, H. Garcia-Molina, and N. Polyzotis. Identifying reliable workers swiftly. Technical report, Stanford University, 2012. URL http://ilpubs.stanford.edu:8090/1043/.

[26] B. Ravikumar, K. Ganesan, and K. B. Lakshmanan. On selecting the largest element in spite of erroneous information. In *STACS 87*, volume 247 of *Lecture Notes in Computer Science*, pages 88–99, 1987. doi: 10.1007/BFb0039597.

[27] Y.-A. Sun, S. Roy, and G. Little. Beyond independent agreement: A tournament selection approach for quality assurance of human computation tasks. In *Human Computation*, volume WS-11-11 of *AAAI Workshops*. AAAI, 2011.

[28] J. Surowiecki. *The wisdom of crowds*. Random House LLC, 2005.

[29] L. L. Thurstone. A law of comparative judgment. *Psychological review*, 34(4):273, 1927.

[30] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 673–684, 2013. ISSN 1063-6382. doi: 10.1109/ICDE.2013.6544865.

[31] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. In *CrowdKDD 2012*. Stanford InfoLab, August 2012. URL http://ilpubs.stanford.edu:8090/1044/.

[32] P. Venetis and H. Garcia-Molina. Dynamic max algorithms in crowdsourcing environments. Technical report, Stanford University, August 2012. URL http://ilpubs.stanford.edu:8090/1050/.

[33] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 989–998, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1229-5. doi: 10.1145/2187836.2187969. URL http://doi.acm.org/10.1145/2187836.2187969.

[34] A. Yao and F. Yao. On fault-tolerant networks for sorting. *SIAM Journal on Computing*, 14(1):120–128, 1985. doi: 10.1137/0214009. URL http://epubs.siam.org/doi/abs/10.1137/0214009.

[35] L. Zhao, G. Sukthankar, and R. Sukthankar. Robust active learning using crowdsourced annotations for activity recognition. In *Human Computation*, 2011.

APPENDIX

A. DISCUSSION ON THE MODEL AND ALGORITHM

Algorithm optimizations. It is possible to optimize the implementation of the algorithm to reduce the time and monetary costs by reducing the number of comparisons it performs in practice. Firstly, we can avoid repeating the comparison of two elements multiple times by the same type of workers. This can happen both in the first phase (naïve workers) or in the second phase (experts). In particular, in the first phase we may have that two elements belongs to the same group at different iterations of the loop on line 4: there is no need to compare them again after the first time. The algorithm will keep an $n \times n$ table containing in cell (i, j) the result of the first comparison between element e_i and element e_j. A second optimization allows to filter out more elements at the end of each iteration of the loop. This would have the net result of having a smaller L_w at the end of iteration w, and therefore terminating earlier. To understand this optimization one should realize that what the algorithm is doing in the first phase is trying to identify all the elements that would win at least $n - \mathsf{u}_\mathrm{n}(n)$ comparisons in an all-play-all tournament among all the n elements. Now, an element may never lose more than $\mathsf{u}_\mathrm{n}(n)$ comparisons in a single iteration of the loop, but it may lose more than that (against different elements) in multiple iterations. This would imply that in a global all-play-all tournament, the element would lose more than $\mathsf{u}_\mathrm{n}(n)$ comparisons so, according to Lemma 1, it cannot be the maximum. We can therefore keep, for each element, a counter of the number of losses against different elements across all the iterations. At the end of each iteration we check these counters for all the elements and remove from L_{i+1} the elements for which the counter is greater than $\mathsf{u}_\mathrm{n}(n)$.

Model vs. real life. Clearly, the model we considered (as is the case with several ones proposed in previous work) makes some simplifying but unrealistic assumptions, such as the existence of a partial order and of a unique error threshold common to all naïve users, and that responses are independent. This is part of the price we pay for a framework that can capture settings in which experts are beneficial or even necessary to achieve satisfactory results and that at the same time allows the design and rigorous analysis of algorithms.

Although the algorithms we propose apply directly to real settings as long as we can define suitable values for the parameters they require, generalizations of our model could reflect scenarios of practical interest more faithfully. For instance, for a more realistic model, we should assume that even if the difference between the values of two elements is above δ_n a worker may err, albeit with a smaller probability than p_err, that the error probability depends on the distance and the worker, and so on.

Extensive experimental work along the lines of the previous paragraph is of great interest but it may require considerable economic resources. Although it is in the scope of the present paper, our attention in this work is more shifted towards the definition of a model that captures naturally the concept of experts and is amenable to analysis. We leave the extensions mentioned above for future work; yet in Section 5 we apply our model in some controlled, albeit rather small, examples for which we possess a ground truth, showing its applicability in real-life scenarios.

B. MISSING PROOFS

In this section we include the proofs missing from the main body of the paper.

B.1 Proof of Lemma 2

PROOF. Let $S \subset A$ be the set of elements with at least $|A| - r$ wins. If $|S| \leq r$ the lemma follows immediately. So, assume that $|S| > r$. Each element in S can win against at most $|A| - |S|$ elements from $A \setminus S$. Then each element from S must also win against at least $|S| - r$ other elements from S so as to have at least $|A| - r$ wins. Indeed, for a given element $e \in S$, let x be the number of elements that e wins among the elements in $A \setminus S$ and y the number of elements that it wins among the elements in S. We will now show that $y \geq |S| - r$. Because $e \in S$, we have that $x + y \geq |A| - r$. In addition, $x \leq |A \setminus S| = |A| - |S|$. Combining these two equations we obtain $y \geq |A| - r - (|A| - |S|) = |S| - r$. Each element in S wins against at least $|S| - r$ other elements in S, so there must be at least $|S|\,(|S| - r)$ wins of elements in S

against other elements in S. Within S one can play at most $\binom{|S|}{2}$ comparisons, so we have that

$$|S|\,(|S|-r) \leq \binom{|S|}{2},$$

which is true if and only if $|S| \leq 2r-1$. □

B.2 Proof of Lemma 3

PROOF. The fact that $|S| \leq 2\mathsf{u}_{\mathrm{n}}(n)-1$ follows trivially from the condition of the **while** block on line 4 of Algorithm 2. After each all-play-all tournament, we discard only elements that lose more than $\mathsf{u}_{\mathrm{n}}(n)$ comparisons so all elements that never lose more than $\mathsf{u}_{\mathrm{n}}(n)$ comparisons are never discarded and therefore are in S. This means that in particular S contains all elements that, in an all-play-all tournament among all elements of L, would lose at most $\mathsf{u}_{\mathrm{n}}(n)$ comparisons or, equivalently, that would win at least $n-\mathsf{u}_{\mathrm{n}}(n)$ comparisons. From Lemma 1 we know that the maximum $M \in L$ is in this latter set, so $M \in S$.

Suppose now that we are at iteration i and consider the elements in a subset G_j of L_i. (To simplify the presentation, we assume that also the last set G_ℓ contains g elements.) After the all-play-all tournament among the elements of G_j, it follows from Lemma 2 applied to G_j that there cannot be more than $2\mathsf{u}_{\mathrm{n}}(n)-1$ elements with at least $g-\mathsf{u}_{\mathrm{n}}(n)$ wins. Then at most $2\mathsf{u}_{\mathrm{n}}(n)-1$ elements from each set G_j belong to L_{i+1}. This means that (ignoring ceilings, for clarity of the presentation) as long as $|L_i| \geq 4\mathsf{u}_{\mathrm{n}}(n)$, we have

$$|L_{i+1}| \leq \frac{|L_i|}{g}(2\mathsf{u}_{\mathrm{n}}(n)-1) = |L_i|\,\frac{2\mathsf{u}_{\mathrm{n}}(n)-1}{4\mathsf{u}_{\mathrm{n}}(n)} \leq \frac{|L_i|}{2}.$$

When $|L_i|$ drops below $4\mathsf{u}_{\mathrm{n}}(n)$, then, by Lemma 2, by the end of the iteration we will obtain that $|L_{i+1}| < 2\mathsf{u}_{\mathrm{n}}(n)$ and in the next iteration the algorithm will terminate. Therefore the algorithm stops after at most i^* iterations, with

$$i^* = \log_2 n - \log_2 4\mathsf{u}_{\mathrm{n}}(n) + 1 \leq \log_2 n.$$

Furthermore, we have that $L_0 = n$, so $\sum_{i=0}^{i^*}|L_i| \leq 2n$.

At each iteration of the loop on line 4, the algorithm performs at most $\binom{g}{2}$ comparisons for the all-play-all tournament of a group G_i. In total, over all iterations and all groups, the number of comparisons that the algorithm performs is at most

$$\sum_{i=0}^{i^*}\frac{|L_i|}{g}\binom{g}{2} = \sum_{i=0}^{i^*}\frac{|L_i|\,(g-1)}{2} \leq \frac{g}{2}\sum_{i=0}^{i^*}|L_i| \leq gn \leq 4n\mathsf{u}_{\mathrm{n}}(n).$$

□

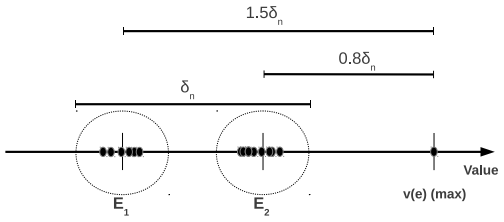


Figure 8: Instance considered in the proof of Lemma 7.

B.3 Randomized Algorithm for the Second phase

For completeness, we present as Algorithm 5 the randomized algorithm for the second phase by Ajtai et al. [2].

Algorithm 5: Randomized Algorithm for finding (approximation of) maximum element M

Input : A set S of s elements.
Output: An estimate of the maximum element M in S, whp.

1 $N_0 \leftarrow S$, $W \leftarrow \emptyset$, $i \leftarrow 0$
2 **while** $|N_i| \geq s^{0.3}$ **do**
3 Sample from W $n^{0.3}$ elements at random and insert them into W
4 Randomly partition the elements in N_i into sets of size $80(c+2)$, for some constant c
5 In each set, perform an all-play-all tournament between the elements of each set to find the minimal element (the element with the fewest wins, with ties broken arbitrarily)
6 Let N_{i+1} contain all of N_i except for the minimal elements found in the previous step
7 $i \leftarrow i+1$;
8 **end**
9 $W \leftarrow W \cup N_i$
10 Play all-play-all tournament between elements of W and return element with highest number of wins, ties broken arbitrarily

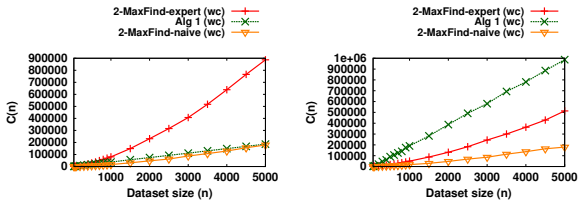
C. MISSING MATERIAL FROM SECT. 5

Model	Price	Exp. 1	Exp. 2
2013 BMW M6 Base -	$123985	2	2
2013 Audi S8 4.0T quattro -	$120375	5	-
2013 Mercedes-Benz ML63 AMG -	$114730	-	-
2013 Mercedes-Benz SL550 -	$114145	2	1
2012 Mercedes-Benz SL550 -	$111675	-	3
2013 Porsche Cayenne GTS -	$97162	7	5
2013 BMW 750 Li xDrive -	$95028	-	-
2012 Audi A8 L 4.2 quattro -	$88991	9	-
2013 Lexus LS 460 Base -	$88110	-	-
2013 Jaguar XJ XJL Portfolio -	$84970	-	-
2013 Chevrolet Corvette 427 -	$83999	1	-
2013 Land Rover Range Rover Sport -	$81151	-	6
2013 Cadillac Escalade Premium -	$75945	-	-
2013 BMW 550 i xDrive -	$72895	5	4
2013 Infiniti QX56 Base -	$71585	-	-
2013 Audi A7 3.0T quattro Premium -	$70020	-	-
2013 Cadillac Escalade EXT Luxury -	$68395	-	-
2013 Porsche Cayenne Diesel -	$67890	7	7
2013 Chevrolet Corvette Grand Sport -	$66510	2	-

Table 2: Ranking at last round of the top-19 cars in the two experiments.

Section 5.1: Plot on worst-case dataset.

In Figure 9 we can see the comparison of the three approaches of Section 5.1 and in Figure 10 the comparison of the cost for different values of the estimation factor (Section 5.2).

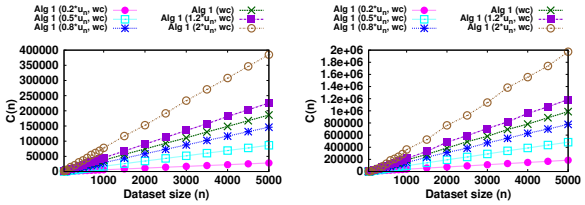


(a) $\mathrm{u_n}(n) = 10, \mathrm{u_e}(n) = 5$ (b) $\mathrm{u_n}(n) = 50, \mathrm{u_e}(n) = 10$

(c) $\mathrm{u_n}(n) = 10, \mathrm{u_e}(n) = 5$ (d) $\mathrm{u_n}(n) = 50, \mathrm{u_e}(n) = 10$

(e) $\mathrm{u_n}(n) = 10, \mathrm{u_e}(n) = 5$ (f) $\mathrm{u_n}(n) = 50, \mathrm{u_e}(n) = 10$

Figure 9: Worst-case cost as a function of n with $c_\mathrm{n} = 1$ and different values of c_e, $\mathrm{u_n}(n)$ and $\mathrm{u_e}(n)$. We have $c_\mathrm{e} = 10$ (row 1), 20 (row 2), 50 (row 3). We compare our approach (Alg 1) with the 2-MaxFind algorithm that uses either only expert or only naïve users.



(a) $\mathrm{u_n}(n) = 10, \mathrm{u_e}(n) = 5$ (b) $\mathrm{u_n}(n) = 50, \mathrm{u_e}(n) = 10$

(c) $\mathrm{u_n}(n) = 10, \mathrm{u_e}(n) = 5$ (d) $\mathrm{u_n}(n) = 50, \mathrm{u_e}(n) = 10$

(e) $\mathrm{u_n}(n) = 10, \mathrm{u_e}(n) = 5$ (f) $\mathrm{u_n}(n) = 50, \mathrm{u_e}(n) = 10$

Figure 10: Worst-case cost as a function of n with $c_\mathrm{n} = 1$ and different values of c_e, $\mathrm{u_n}(n)$ and $\mathrm{u_e}(n)$. We have $c_\mathrm{e} = 10$ (row 1), 20 (row 2), 50 (row 3). We compare the cost of our 2-phase algorithm for various values of the estimation factor.