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

For reviewing purposes only. Please do not distribute!

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
Sapienza University
Rome, Italy
mele@dis.uniroma1.it

Matteo Riondato
Brown University
Providence, RI, USA
matteo@cs.brown.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 successfully used to address problems that would be hard or impossible to solve for machines. On the other hand, as we also 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. In this work, we address this limitation, 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 theoreticalwork. Our computational model allows evaluating the performance of crowdsourcing algorithms along different axes, including monetary cost and time. 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 very close (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 almost optimally. Finally, we evaluate our algorithm on real and artificial datasets using the CrowdFlower crowdsourcing platform, showing that our approach is also effective in practice.

1. INTRODUCTION

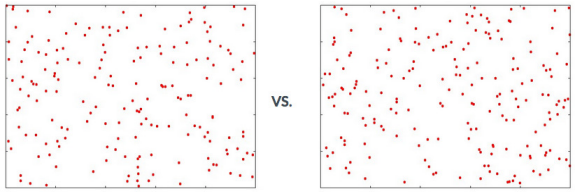
Crowdsourcing is a computational paradigm that enables outsourcing pieces of the computation to humans who perform them under monetary compensation. The main rationale for the involvement of humans is the existence of tasks that are easy to perform for a person but very difficult or impossible to accomplish for a machine. It has applications in machine learning, visualization, recommendation systems, computational photography, and data analytics. Examples include identifying the best picture representing a certain object or location, editing a block of text to improve clarity or find style errors, or suggesting the best value or price for a product. As these examples highlight, one of the main reasons for bringing humans into the computational loop is that the task is under-specified 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.

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 [10, 25]. There are two main sources of error. The first is *incompleteness of information*: tasks may be underspecified in many respects, so that individual factors come into play when humans perform them. For example, when asked to choose the picture *best* representing the Colosseum, people will not in general agree on the definition of *best*. Similarly, people will not converge on a single answer when requested to value items (e.g., real estates or cars) on the basis of limited information, such as one or more pictures and a summary of the main characteristics of the items they are requested to value. 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*). As an example of the first two, a human worker may provide an answer that is not the one she actually intended to give, or erroneously select the minimum of two elements instead of the maximum. *Spammers* instead are workers with disruptive behavior: they either try to perform as many tasks as possible by selecting answers randomly to maximize their personal utilities, or they select the wrong answer on purpose.

To motivate the topic of this work, assume first 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 this question by using abilities that are either hard-wired or naturally learnt in the course of time. Even though one may err, chances are that she will respond correctly. Thus, if we ask several individ-

uals the same question, we expect that the majority will answer correctly, and the higher the number of people we ask, the higher our confindence that the answer is correct. In cases like this we can successfully apply the paradigm of the *wisdom of crowds* [19]. Indeed, it is for tasks like this 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



(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 the questions that we asked.

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 are three characteristics that expert workers possess: (1) they obtain training or pass tests that 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. We call nonexpert workers *naïve*.

The design and analysis of algorithms that employ human workers on crowdsourcing platforms require computational and cost models that are flexible 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 that cannot be attained by naïve workers but that come at a higher cost.

1.1 Contributions and Roadmap

In this paper we attempt to capture important aspects of crowdsourcing by presenting computational models and designing algorithms on top of them. We introduce the concept of an expert, who can add qualitatively more power to computations that we are able to perform. We make our study concrete by addressing the problem of finding the maximum among a set of elements. This is a problem often encountered in practice in crowdsourcing scenarios (e.g., ranking of search results, web page relevance evaluation, selection of the best labeling for a picture). 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 [22, 23] (without considering experts).

To summarize our contributions:

- By performing experiments on the CrowdFlower crowdsourcing platform, we identify key characteristics of workers' performance in diverse scenarios (Sect. 3.1).
- Building on the findings we present models that can capture the behavior in the diverse settings (Sect. 3.2).
- We introduce the concept of crowdsourcing experts and we incorporate them in our model (Sect. 3.3).
- Based 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, which we analyze theoretically and we show that it is optimal (Sect. 4).
- We perform a series of simulations for evaluating the efficiency of our algorithm in practice complementing our theoretical anlysis, and we perform a set of experiments on CrowdFlower to show its effectiveness in finding the maximum (Sect. 5).

2. RELATED WORK

Many works in the algorithmic literature dealt with the problem of sorting or computing the maximum of a set of elements using comparators some of which may be faulty [17, 24], even without considering the crowdsourcing settings. Various different models and solutions were proposed. A number of works considered the idea that the comparator errs with some probability at each comparison, independently from other comparisons [4–6, 9, 11, 12, 23]. They presented algorithms to compute the maximum element, studying numerous variants of this purely probabilistic error model.

[1] Prices are from August 2013.

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 in [6] or [5] (in both works, more sophisticated models are also considered). 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 situation where such conclusion is not possible: an expert has more "capabilities" than a naïve worker and her answers can not 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 [7, 8] and more in general to computational tasks involving communication across noisy channels. For a survey of literature on this broader area of research see for example [16]. 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, i.e., that comparators may belong to different classes with different error parameters and different costs, and one can not be used to "simulate" the other.

In the crowdsourcing environment, Marcus et al. [13] 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 [22] and Venetis et al. [23] present algorithms for finding the maximum in crowdsourcing environments based on static and dynamic tournaments. Different error models taken from the psychometrics literature are considered. The optimal parameters for the algorithms are computed using simulated annealing, given a specific error model and a budget of computational resources. There is no discussion 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. [18]: 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.

Karger et al. [11] 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. Venetis and Garcia-Molina [21] present mechanisms to detect badly performing workers and to rule them out. In our settings, the classes of the workers are known in advance, as the experts are hand-picked and assumed to perform much better than the other workers. In a sense, our work can be considered complementary to [11].

Mason and Watts [14] investigated the impact 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 can not 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. [15] proposed algorithms to compute the number of workers whom to ask the same question in order to achieve the best accuracy with a fixed available budget. The workers all belong to the same class and correctly or incorrectly perform a task with a probability depending on the task but not on the workers.

Davidson et al. [5] introduced algorithms inspired by [6] 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, which is a key contribution of our work.

3. MODELING CROWDSOURCING

In this section we formalize crowdsourcing computation for computing the maximum (or best) from 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. We call M_L the *maximum element of* L. The set L will often be clear from 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 Sect. 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 performed in prior works as it simplifies the formalization and it can be generalized) and returns the one that she believes has the maximum value. Following Venetis et al. [23], the algorithms we consider are organized in *logical time steps*. In the sth logical step, they send a batch B_s of pairwise comparisons 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.[3] 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*[4] 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. [23] 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. See the discussion in Sect. 3.4.

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

[3] One can also envision nonadaptive algorithms.

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

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 croudsourcing platform.[5] We describe first the two 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 simulate an expert by having multiple naïve workers answer the same question independently, or is there a* cognitive barrier*, at least for certain types of questions?*

Datasets. We created two different datasets:[6]

- DOTS (inspired by [21]): It consists of a collection of images containing randomly placed dots. The number of dots in each picture goes from 100 to 1500, ranging in steps of 20.
- CARS: This dataset contains a description of cars. We downloaded a set of approx. 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.

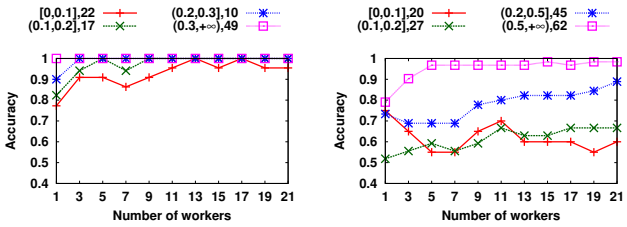
  CARS a very noisy dataset. For instance we found several sets of cars of the same make and model that varied significantly in the price, often in the order of several thousands dollars. Most of the times this difference is due to differences in equipment, but sometimes different dealers also sell the same car at different prices. Showing the entire equipment to workers is impractical and would lead to higher spam 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, multiple channels, and has no restrictions for European users.

For each of dataset we used 50 elements for comparisons, and some additional ones for *gold* comparisons, i.e., 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 Fig. 2. In Fig. 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*. Independently from the difference in the number of dots, the accuracy is quite low when considering a single worker but 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. We also measured the accuracy when looking at the absolute difference rather than the relative one. The results are qualitatively the same as the ones that we report so we omit the corresponding plot.



(a) DOTS dataset (b) CARS dataset

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

This experiment 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. Second, if we want to achieve a higher accuracy we often need multiple classes of workers. This observation led us to the introduction and modelling of *experts*, skilled workers that are able to correctly rank elements with close values, but that require a higher monetary compensation. This is the topic of Sect. 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. We present a threshold-based model, which also captures the behavior in the CARS experiment.

Probabilistic Error Model. A common approach is to assume that 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

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

[6] The datasets will become available at time of publication.

with the residual probability, independently of any other comparison she or other workers might perform [4–6, 9, 11, 12, 23]. 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 error 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 Fig. 2(a). Of course, one can create instances for the DOTS on which we can not 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.

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 [20]). 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. 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 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 the threshold δ allows us to take into account user input errors, malicious workers, or prior bias that the workers may have (we will show an example of this bias in the experiments).

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 take the value 0, so the threshold model is a generalization of the probabilistic model.

3.3 Threshold Model with Experts

The experimental findings of Sect. 3.1 demonstrate that there are types of questions that typically require knowledge that is not 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 the experts, as we mention in the introduction.

The extension of the threshold model to capture experts is straightforward. 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 $\delta_\mathrm{n} \gg \delta_\mathrm{e}$ and $\varepsilon_\mathrm{e} \leq \varepsilon_\mathrm{n}$ (possibly $\varepsilon_\mathrm{e} = 0$). Given that there are now two thresholds, we talk of *naïve-indistinguishable* elements for elements u, v with $\mathsf{d}(u, v) < \delta_\mathrm{n}$, and of *expert-indistinguishable* if $\mathsf{d}(u, v) < \delta_\mathrm{e}$. Indeed, under our model, two elements that are expert-indistinguishable are also naïve-indistinguishable.

In the rest of this paper we set the residual error probability ε_e of the experts to zero for simplicity of the analysis (all our results carry over to the more general setting with high probability by repeating questions). Assuming that experts' residual error probability is vanishingly small is reasonable in practice, because usually experts are not susceptible to the residual errors that naïve users are. In particular, we expect them not to be spammers, they have somewhat complete information (e.g., they will hardly confuse a base car model with a full-optionals one), they do not make input mistakes or have a wrong bias about the value of an object (e.g., they would not be surprised if a specific Mercedes–Benz model costs less than a Mazda). Still, it may very difficult even for them to pick correctly between two elements with very close values.

3.4 Cost Models

We analyze our algorithms using different cost models for different resources. For each of them, we express the cost of the algorithm as a function of the size of the input $n = |L|$.

Monetary Cost. 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 ($c_\mathrm{e} \gg c_\mathrm{n}$). Therefore, if an algorithm performs $x_\mathrm{e}(n)$ expert comparisons and $x_\mathrm{n}(n)$ naïve comparisons, the total *monetary cost* of the algorithm is

$$\mathsf{C}(n) = x_\mathrm{e}(n) \cdot c_\mathrm{e} + x_\mathrm{n}(n) \cdot c_\mathrm{n}.$$

Time. Another important cost function is the *time* $\mathsf{T}(n, |W|)$, that is, the number of *physical* time steps to compute the element $e \in L$ whose value approximates M. The number of time steps depends

on the number of available workers $|W|$: if there are not enough workers, a single *logical* step of the algorithm will require multiple physical time steps. The minimum number of workers needed to perform each logical step as a single physical step (resulting in the minimum[7] $\mathsf{T}(n, |W|)$) is

$$\mathsf{W}(n) = \max_t |W_t| \ .$$

Note that the number of logical time steps of an algorithm can be an unsatisfactory proxy for $\mathsf{T}(n, |W|)$ when the number of active workers can considerably vary along consecutive time steps. When instead this number is large enough or is substantially constant, as remarked by Venetis et al. [23], the number of logical time steps is a reasonable measure of the time complexity. We decouple logical and physical steps in order to be as general as possible.

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); (2) indeed many past works on crowdsourcing algorithms studied the problem of finding the maximum [13, 22, 23]; (3) it is well-specified and amendable to (albeit nontrivial) theoretical analysis. Furthermore, we hope 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. Let $M \in L$ be an element with maximum value among those in L. The algorithm finds an element $e \in L$ such that $\mathsf{d}(M, e) \leq 3\delta_{\mathrm{e}}$. Its time and monetary costs depend on the value $\mathsf{u}_{\mathrm{n}}(n)$ which represents the number of elements in L that are naïve-indistinguishable from M: $\mathsf{u}_{\mathrm{n}}(n) = |\{e : \mathsf{d}(M, e) \leq \delta_{\mathrm{n}}\}|$. Similarly, we define $\mathsf{u}_{\mathrm{e}}(n) = |\{e : \mathsf{d}(M, e) \leq \delta_{\mathrm{e}}\}|$. We assume that $\mathsf{u}_{\mathrm{n}}(n) = o(n)$, which also implies that $\mathsf{u}_{\mathrm{e}}(n) = o(n)$.

Remark. For the sake of presentation we assume that the residual errors ε_{n} and ε_{e} (see Sect. 3) are equal to 0. Our results can be extended to any values less than $1/2$.

4.1 An Expert-Aware Max-Finding Algorithm

The algorithm consists of two phases, summed up in Alg. 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 $O(\mathsf{u}_{\mathrm{n}}(n))$ that contains M, using only naïve workers to perform comparisons.*[8]

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

[7] For the given algorithm of course.

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

Algorithm 1: Find an element close to the maximum M

input : A set L of n elements, a function $\mathsf{u}_{\mathrm{n}}(n) = o(n)$.
output: An approximation of the maximum element M in L.
1 Obtain a set $S \subset L$ using Alg. 2 with naïve workers
2 Return the output of Alg. 3 on input S with experts

PROBLEM 2. *Given an input set S of size $O(\mathsf{u}_{\mathrm{n}}(n))$ containing M, return an element e, such that $\mathsf{d}(M, e) = O(\delta_{\mathrm{e}})$, using only experts and performing as few comparisons as possible.*[9]

One could solve this problem with $\Theta(|S|^2)$ experts comparisons by performing a simple round-robin tournament[10] among the elements in S. In order to reduce the number of comparisons performed by experts, we instead use one of the algorithms proposed in [2, Sect. 3] and return the element found by this algorithm. More details about the algorithm of choice for the second phase are given in Sect. 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 Alg. 2. It relies on some combinatorial properties of *round-robin tournaments*, which we prove below. In particular, Lemma 1 shows a key property of element M in round-robin tournaments.

LEMMA 1. *In a round-robin tournament among the elements of L, the maximum element M wins at least $n - \mathsf{u}_{\mathrm{n}}(n)$ comparisons.*

PROOF. By definition of $\mathsf{u}_{\mathrm{n}}(n)$, there are at most $\mathsf{u}_{\mathrm{n}}(n)$ 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 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 a round-robin 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 a round-robin tournament among the elements of A, there are at most $2r - 1$ elements that win at least $|A| - r$ comparisons each .*

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 in order 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 against among the elements in $A \setminus S$ and y the number of elements that it wins against among the elements in S. We will now show that $y \geq |S| - r$. Since $e \in S$, 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

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

[10] In a round-robin tournament, each element is compared against all others.

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

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

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 a round-robin tournament, otherwise we may miss M. We could easily find S by performing a round-robin 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 a round-robin tournament within each subset. We discard elements that lose at least $\mathsf{u_n}(n)$ comparisons in their subset round-robin tournament 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 round-robin 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 Alg. 2. In Lemma 3 we prove its correctness and show that it performs only $O(n\mathsf{u_n}(n))$ comparisons. Later, in Sect. 4.3 we prove that this bound is optimal, within constant factors.

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.

1 $g \leftarrow 4\mathsf{u_n}(n)$
2 $i \leftarrow 0$
3 $L_i \leftarrow L$
4 **while** $|L_i| > 2\mathsf{u_n}(n) - 1$ **do**
5 $L_{i+1} \leftarrow \emptyset$
6 Partition L_i into subsets $G_1, \ldots, G_\ell$ of size g (the last one may be smaller)
7 **forall the** $G_j, 1 \leq j \leq \ell - 1$ **do**
8 Perform a round robin tournament among the elements of G_j
9 Let W_j be the set of the elements of G_j that win at least $g - \mathsf{u_n}(n)$ comparisons in the round-robin tournament.
10 $L_{i+1} \leftarrow L_{i+1} \cup W_j$
11 **end**
12 **if** $|G_\ell| \leq \mathsf{u_n}(n)$ **then**
13 $L_{i+1} \leftarrow L_{i+1} \cup G_\ell$
14 **end**
15 **else**
16 Let W_ℓ be the set of the elements of G_j that win at least $|G_\ell| - \mathsf{u_n}(n)$
17 $L_{i+1} \leftarrow L_{i+1} \cup W_\ell$
18 **end**
19 $i \leftarrow i + 1$
20 **end**
21 **return** L_i

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.*

PROOF. The fact that $|S| \leq 2\mathsf{u_n}(n) - 1$ follows trivially from the condition of the **while** block on line 4 of Alg. 2. After each round-robin tournament, we discard only elements that lose more than $\mathsf{u_n}(n)$ comparisons so all elements that never lose more than $\mathsf{u_n}(n)$ comparisons are never discarded and therefore are in S. This means that in particular S contains all elements that, in a round-robin tournament among all elements of L, would lose at most $\mathsf{u_n}(n)$ comparisons or, equivalently, that would win at least $n - \mathsf{u_n}(n)$ 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. After the round-robin tournament among the elements of G_j, it follows from Lemma 2 applied to G_j that there cannot be more than $2\mathsf{u_n}(n) - 1$ elements with at least $g - \mathsf{u_n}(n)$ wins. Then at most $2\mathsf{u_n}(n) - 1$ elements from each set G_j belong to L_{i+1}. This means that (ignoring ceilings, for clarity of the presentation) as long as $|L_i| \geq 4\mathsf{u_n}(n)$, we have

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

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

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

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 round-robin 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

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

□

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 a round-robin tournament on the set S and pick the element e wi*th 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 presented in [2, Sect. 3.1]; it performs $O(\mathsf{u_n}(n)^{3/2})$ expert comparisons to return an element e s.t. $\mathsf{d}(M, e) \leq 2\delta_\mathrm{e}$.
3. Use the randomized algorithm of [2, Sect. 3.2]; this performs $\Theta(\mathsf{u_n}(n))$ expert comparisons returns an element e with the guarantee that $\mathsf{d}(M, e) \leq 3\delta_\mathrm{e}$ with high probability.

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. 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 returns an element that is closer to the maximum (only $2\delta_e$ far, the best possible for the model [2]). For this reason we use the 2-MaxFind algorithm for the simulations in Sect. 5.

For the sake of completeness, we now outline the algorithm and present its pseudocode in Alg. 3 (see also [2, Sect. 3.1]). Consider the candidate set S returned by Alg. 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 a round-robin tournament between 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 round-robin tournament among these elements determines the winner.

Algorithm 3: Find (approximation of) maximum element M

input : A set S of s elements.
output: An estimate of the maximum element M in S.
1 Label all items as candidates
2 **while** *More than* $\lceil\sqrt{s}\rceil$ *candidates* **do**
3 Pick an arbitrary set of $\lceil\sqrt{s}\rceil$ candidate elements and play them in a round-robin tournament. Let x have the most number of wins
4 Compare x against all candidate elements and eliminate all elements that lose to x
5 **end**
6 Play a final round-robin tournament among the at most $\lceil\sqrt{s}\rceil$ survivors and return the element with the most wins

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 described in [2, Sect. 3.2].

Correctness. The following lemma shows the correctness of our algorithm, assuming the randomized algorithm from [2, Sect. 3.2] is used[11] (i.e., option 3 from the previous discussion).

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* $\mathsf{d}(M,e) \leq 3\delta_e$.

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

Cost analysis. The following lemmas quantify the various costs of our algorithm assuming, as we mentioned, that in the second phase we apply the randomized algorithm in [2, Sect. 3.2].

LEMMA 5. *Given* W_n *naïve workers and* W_e *experts, the time cost* $\mathsf{T}(n)$ *of our algorithm is*

$$\mathsf{T}(n) = O\left(\log n \frac{ng}{W_n} + \frac{(\mathsf{u}_n(n))^{1.7}}{W_e} + \frac{(\mathsf{u}_n(n))^{0.6}\log^2 \mathsf{u}_n(n)}{W_e}\right).$$

PROOF. The first phase requires

$$i^* \leq \log_2 n$$

logical steps to compute the set S. This is the number of iterations of the loop on line 4 in Alg. 2 (see proof of Lemma 3). In each of them we perform $O(ng)$ comparisons, which take $O(ng/W_n)$ time steps to be performed by W_n naïve workers.

In the second phase we follow the algorithm in [2], which works in two sub-phases. In the first sub-phase, it requires $(\mathsf{u}_n(n))^{0.7}$ rounds, in each of which $O(\mathsf{u}_n(n))$ comparisons are performed, so each round require $O(\mathsf{u}_n(n)/W_e)$ time steps, for a total of

$$O\left(\frac{\mathsf{u}_n(n)^{1.7}}{W_e}\right)$$

time steps. In the second sub-phase, it performs a round-robin tournament among the elements of a set of size $O((\mathsf{u}_n(n))^{0.3}\log \mathsf{u}_n(n))$. This requires $O((\mathsf{u}_n(n))^{0.6}\log^2 \mathsf{u}_n(n)/W_e)$ time steps. □

LEMMA 6. *The optimum number of workers that minimize the time cost of our algorithm is* $\mathsf{W}_n(n) = O(ng) = O(n\mathsf{u}_n(n))$ *naïve workers and* $\mathsf{W}_e(n) = O(\mathsf{u}_n(n) + (\mathsf{u}_n(n))^{0.6}\log^2 \mathsf{u}_n(n))$ *experts.*

PROOF. The thesis follows easily from the proof of Lemma 5. □

LEMMA 7. *Our algorithm performs* $O(n\mathsf{u}_n(n))$ *naïve and* $O((\mathsf{u}_n(n))^{1.7} + (\mathsf{u}_n(n))^{0.6}\log^2 \mathsf{u}_n(n))$ *expert comparisons. Accordingly, its monetary cost* $\mathsf{C}(n)$ *is*

$$\mathsf{C}(n) = O(c_n n\mathsf{u}_n(n) + c_e((\mathsf{u}_n(n))^{1.7} + (\mathsf{u}_n(n))^{0.6}\log^2 \mathsf{u}_n(n))) .$$

PROOF. In the first phase of the algorithm we perform $O(n\mathsf{u}_n(n))$ comparisons, each at a cost c_n. In the second phase we perform $O((\mathsf{u}_n(n))^{1.7} + (\mathsf{u}_n(n))^{0.6}\log^2 \mathsf{u}_n(n))$, each at cost c_e. □

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 is an algorithm that computes an element* e *such that* $\mathsf{d}(e,M) \leq 2\delta_e$ *and that performs at most* $4n\mathsf{u}_n(n)$ *naïve comparisons and* $2\mathsf{u}_n(n)^{3/2}$ *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 $\mathsf{u}_n(n)$ elements are naïve indistinguishable from the maximum. This implies that the number of expert comparisons required, are in the worst case $\Omega(\mathsf{u}_n(n))$. In fact, for the threshold error model we consider in this paper, it is possible to prove stronger results. The following theorem follows from [2].

LEMMA 8. *Any deterministic algorithm that computes an element* e *such that* $\mathsf{d}(e,M) \leq 2\delta_e$ *must perform at least* $\Omega(\mathsf{u}_n(n)^{4/3})$ *expert comparisons.*

Next, we turn our attention to Problem 1. Here, we prove in Corollary 1 below that the number of comparisons performed by Alg. 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 $n-\mathsf{u}_n(n)$ comparisons in a round-robin tournament among the elements of L, because any of them could be the maximum.

[11] The result holds deterministically if we use the 2-MaxFind algorithm.

LEMMA 9. *Let C be any set of comparisons by naïve workers. If there exists an element e that takes part in fewer than $\mathsf{u}_\mathrm{n}(n)$ 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 Fig. 3). 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 then 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. □

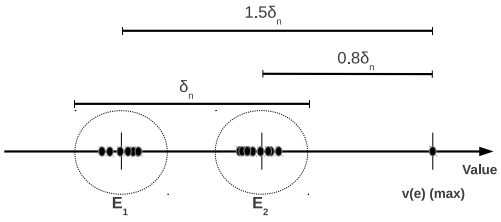


Figure 3: Instance considered in the proof of Lemma 9.

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 9, 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 Discussion

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 the 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 a round-robin 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 round-robin tournament, the element would lose more than $\mathsf{u}_\mathrm{n}(n)$ comparisons so, according to Lemma 1 it can not 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 $\mathsf{u}_\mathrm{n}(n)$.

On the 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 would depend on the application and on the crowdsourcing platform. In practice, this value would have to be estimated. Using the results of some sample comparisons on ground-truth data, one may find at what distance the workers have difficulty distinguishing the elements. Note that this task becomes easier because of the fact that overestimating has effect only on the number of comparisons and does not harm the accuracy of the results.

5. EXPERIMENTS

To evaluate the efficiency of our algorithm we performed a series of simulations, which we present next. Then we show how our algorithm performs on real-life experiments, using the CrowdFlower platform, described in Section 3.1.

5.1 Simulations

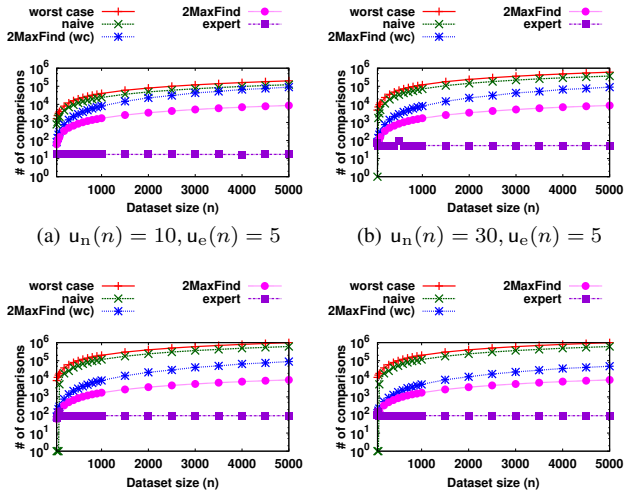
Our algorithm is optimal asymptotically in the sense that minimizes both naïve and expert comparisons (Sect 4). Here we empirically evaluate the efficiency of our algorithm and compare it with 2-MaxFind [2] (see also Sect. 4.1.2) when it uses only experts.

We study the performance of the two 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 parameters: n, δ_n, δ_e; the last two define also 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 the threshold, 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 the 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 loose, in order 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 running time of the algorithm. For our algorithm we considered the upper bound predicted by the theory.

In Fig. 4 we can see the number of comparisons for our algorithm and 2-MaxFind. First note how much smaller is the number of expert comparisons; it only depends on the left-over 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 naiïve comparisons, actually higher than the number of expert comparisons that 2-MaxFind performs. Given though that we expect expert comparisons to have a much higher cost than naïve comparisons (obviously the exact values will depend on the platform and on the application), 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.



(a) $\mathrm{u_n}(n) = 10, \mathrm{u_e}(n) = 5$ (b) $\mathrm{u_n}(n) = 30, \mathrm{u_e}(n) = 5$

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

Figure 4: Number of comparisons as a function of n for different values of $\mathrm{u_n}(n)$ and $\mathrm{u_e}(n)$. Note that the y-axis is on a log scale. Lowest line: # number of expert comparisons of our algorithm on the second phase; 2nd and 3rd line from the bottom: average and worst-case # expert comparisons of 2-MaxFind; top 2 lines: average and worst-case (theoretical upper bound) # expert comparisons of our algorithm.

5.2 Experiments on CrowdFlower

We also conducted a series of experiments on CrowdFlower, with the goal of assessing to what extent our algorithm is able to compute a value close to the maximum.

Settings. We used the datasets DOTS and CARS described in Sect. 3.1. For each dataset we conducted two identical experiments, namely, they have equal configuration and receive the same input data. From each of 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.

We implemented Algorithm 2 with a value of $\mathrm{u_n}(n) = 5$. This implies a value of $g = 20$. Yet, we estimated the number of comparisons required, and for the given number of n and $\mathrm{u_n}(n)$ it turns out that $g = 10$ leads to a lower number of comparisons. We used this value for our experiments. In the second phase we always end up with at most 10 elements, so we just performed a round-robin tournament among experts to deduce the winner.

CrowdFlower does not provide a service of experts for the problem that we address, so we simulated an expert query with 7 naïve queries and choose the answer with the majority of votes, votes, just as in Sect. 3.1. As we saw there, this approach is effective in the DOTS experiment but not for CARS. Nevertheless we report the results also for the latter and they confirm the findings of Sect. 3.1.

Experiments on DOTS. This experiment was inspired from previous work [21]. 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 perfectand are summarized in Table **??**. In both experiments, nine elements passed to the second phase, and in both they were the real top-9 elements. The second-phase experts were able to always find the minimum, and, further, the output correctly orders the top-9 elements, except in one case for one experiments, in which the top-6 and top-7 elements are swapped.

# 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.

Experiments on CARS. In this experiment we wanted to study the quality of the results when the data is much more fuzzy. We attempted to find the car that is most highly priced.

We conducted two experiments. We can see the ranking of the last round in Table 1. Note that the top car always reaches the last round, yet the simulated experts are not able to identify it, indicating the need for real experts. Furthermore, note that because of the fuzziness of the data some cars far from the top-10 arrive at the second phase.

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.

6. CONCLUSIONS

In this paper we defined computational and cost models for crowdsourcing, introducing the idea of using workers with different expertise levels. We considered different error models and different costs that arise when optimizing the performances of algorithms for crowdsourcing. The definition of the *threshold error model with experts* is a novelty of our work, and takes into consideration the fact that in many applications it is not possible to simulate the experts using naïve workers, as some of the real-life experiments presented

in Section 5 also suggest. 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 elements, and then asks the expert workers to focus on a restricted set. We provide lower bounds to the number of comparisons required to perform the task. An extensive 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 it these cases the use of real experts is of paramount importance.

7. REFERENCES

[1] M. Aigner. Finding the maximum and minimum. *Discr. Appl. Math.*, 74(1):1 – 12, 1997.

[2] M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson. Sorting and selection with imprecise comparisons. ICALP'09, 2009.

[4] S. Assaf and E. Upfal. Fault tolerant sorting networks. *SIAM J. Discr. Math.*, 4(4):472–480, 1991.

[5] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. ICDT'13, 2013.

[6] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.

[7] I. Finocchi and G. F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). STOC'04, 2004.

[8] I. Finocchi, F. Grandoni, and G. F. Italiano. Designing reliable algorithms in unreliable memories. *Comp. Sci. Rev.*, 1(2):77 – 87, 2007.

[9] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. SIGMOD'12, 2012.

[10] P.-Y. Hsueh, P. Melville, and V. Sindhwani. Data quality from crowdsourcing: a study of annotation selection criteria. ALNLP'09, 2009.

[11] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. NIPS'11, 2011.

[12] 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.

[13] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. VLDB'12, 2012.

[14] W. Mason and D. J. Watts. Financial incentives and the "performance of crowds". *SIGKDD Explor. Newsl.*, 11(2):100–108, May 2010.

[15] 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. TR HKU CS TR-2013-01, U. of Hong Kong, 2013.

[16] A. Pelc. Searching games with errors – fifty years of coping with liars. *Theor. Comp. Sci.*, 270(1):71–109, 2002.

[17] B. Ravikumar, K. Ganesan, and K. B. Lakshmanan. On selecting the largest element in spite of erroneous information. STACS'87, 1987.

[18] Y.-A. Sun, S. Roy, and G. Little. Beyond independent agreement: A tournament selection approach for quality assurance of human computation tasks. AAAI HumComp'11, 2011.

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

[20] L. L. Thurstone. A law of comparative judgment. *Psychol. Rev.*, 34 (4):273, 1927.

[21] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. CrowdKDD'12, 2012.

[22] P. Venetis and H. Garcia-Molina. Dynamic max algorithms in crowdsourcing environments. TR, Stanford University, 2012.

[23] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. WWW'12, 2012.

[24] A. Yao and F. Yao. On fault-tolerant networks for sorting. *SIAM J. Comput.*, 14(1):120–128, 1985.

[25] L. Zhao, G. Sukthankar, and R. Sukthankar. Robust active learning using crowdsourced annotations for activity recognition. AAAI HumComp'11, 2011.