Crowdsourcing for Top-K Query Processing over Uncertain Data

Eleonora Ciceri, Piero Fraternali, Davide Martinenghi and Marco Tagliasacchi

Abstract—Querying uncertain data has become a prominent application due to the proliferation of user-generated content from social media and of data streams from sensors. When data ambiguity cannot be reduced algorithmically, crowdsourcing proves a viable approach, which consists in posting tasks to humans and harnessing their judgment for improving the confidence about data values or relationships. This paper tackles the problem of processing top-K queries over uncertain data with the help of crowdsourcing for quickly converging to the real ordering of relevant results. Several offline and online approaches for addressing questions to a crowd are defined and contrasted on both synthetic and real data sets, with the aim of minimizing the crowd interactions necessary to find the real ordering of the result set.

Index Terms—User/Machine Systems, Query processing.

1 Introduction

Both social media and sensing infrastructures are producing an unprecedented mass of data that are at the base of numerous applications in such fields as information retrieval, data integration, location-based services, monitoring and surveillance, predictive modeling of natural and economic phenomena, public health, and more. The common characteristic of both sensor data and user-generated content is their uncertain nature, due to either the noise inherent in sensors or the imprecision of human contributions. Therefore query processing over uncertain data has become an active research field [47], where solutions are being sought for coping with the two main uncertainty factors inherent in this class of applications: the approximate nature of users' information needs and the uncertainty residing in the queried data.

In the well-known class of applications commonly referred to as "top-K queries" [26], the objective is to find the best K objects matching the user's information need, formulated as a scoring function over the objects' attribute values. If both the data and the scoring function are deterministic, the best K objects can be univocally determined and totally ordered so as to produce a single ranked result set (as long as ties are broken by some deterministic rule).

However, in application scenarios involving uncertain data and fuzzy information needs, this does not hold. For example, in a large social network the importance of a given user may be computed as a fuzzy mixture of several characteristics, such as her network centrality, level of activity, expertise, and topical affinity. A viral marketing campaign may try to identify the "best" K users and exploit their prominence to spread the popularity of a product [20]. Another instance occurs when sorting videos for recency or popularity in a video sharing site [4]: for example, the video timestamps may be uncertain because the files were annotated at a coarse granularity level (e.g., the day), or perhaps because similar but not identical types of annotations are available (e.g., upload instead of creation time). Sometimes, data processing may also be a source of uncertainty; for example, when tagging images with a visual quality or representativeness index, the score may be algorithmically computed as a probability distribution, with a spread related to the confidence of the algorithm employed to estimate quality [21], [39].

Furthermore, uncertainty may also derive from the user's information need itself; for example, when ranking apartments for sale, their value depends on the weights assigned to price, size, location, etc., which may be uncertain because they were specified only qualitatively by the user or estimated by a learning-to-rank algorithm [50].

When either the attribute values or the scoring function are nondeterministic, there may be no consensus on a single ordering, but rather a space of possible orderings. For example, a query for the top-K most recent videos may return multiple orderings, namely all those compatible with the uncertainty of the timestamps. To determine the correct ordering, one needs to acquire additional information so as to reduce the amount of uncertainty associated with the queried data. Without this reduction, even moderate amounts of uncertainty make top-K answers become useless, since none of the returned orderings would be clearly preferred to the others.

An emerging trend in data processing is *crowd-*

• *Eleonora Ciceri, Piero Fraternali, Davide Martinenghi and Marco Tagliasacchi are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy.*
E-mail: first.last@polimi.it

This work is partly funded by the EC's FP7 CUbRIK and SmartH2O projects, and the FESR project Proactive

sourcing [16], defined as the systematic engagement of humans in the resolution of tasks through online distributed work. This approach combines human and automatic computation in order to solve complex problems, and has been applied to a variety of data and query processing tasks, including multimedia content analysis, data cleaning, semantic data integration, and query answering [51]. When data ambiguity can be resolved by human judgment, crowdsourcing becomes a viable tool for converging towards a unique or at least more determinate query result. For example, in an event detection and sorting scenario, a human could know the relative order of occurrence of two events; with this information, one could discard the incompatible orderings. However, crowdsourcing has problems of its own [1]: the output of humans is uncertain, too, and thus additional knowledge must be properly integrated, notably by aggregating the responses of multiple contributors. Due to this redundancy, significant budget savings may be achieved by avoiding to post even a small amount of tasks. This problem requires an appropriate policy in the formulation of the tasks to submit to the crowd, aimed at reaching the maximum reduction of uncertainty with the smallest number of crowd task executions.

The goal of this paper is to define and compare task selection policies for uncertainty reduction via crowdsourcing, with emphasis on the case of top-K queries. Given a data set with uncertain values, our objective is to pose to a crowd the set of questions that, within an allowed budget, minimizes the expected residual uncertainty of the result, possibly leading to a unique ordering of the top K results.

The main contributions of the paper are as follows:

1) We formalize a framework for uncertain top-K query processing, adapt to it existing techniques for computing the possible orderings, and introduce a procedure for removing unsuitable orderings, given new knowledge on the relative order of the objects (Section 3).
2) We define and contrast several measures of uncertainty, either agnostic (*Entropy*) or dependent on the structure of the orderings (Section 4).
3) We formulate the problem of *Uncertainty Resolution* (UR) in the context of top-K query processing over uncertain data with crowd support (Section 5.1). The UR problem amounts to identifying the shortest sequence of questions that, when submitted to the crowd, ensures the convergence to a unique, or at least more determinate, sorted result set. We show that no deterministic algorithm can find the optimal solution for an arbitrary UR problem.
4) We introduce two families of heuristics for question selection: *offline*, where all questions are selected prior to interacting with the crowd, and *online*, where crowd answers and question selection can intermix. For the offline case we define a relaxed, probabilistic version of optimality, and exhibit an algorithm that attains it as well as sub-optimal but faster algorithms (Section 5.2). We also generalize the algorithms to the case of answers collected from noisy workers (Section 5.3).
5) We propose an algorithm that avoids the materialization of the entire space of possible orderings to achieve even faster results (Section 5.4).
6) We conduct an extensive experimental evaluation of several algorithms on both synthetic and real datasets, and with a real crowd, in order to assess their performance and scalability (Section 6).

2 Background

We consider the problem of answering a top-K query over a relational database table T containing N tuples. The relevance of a tuple to the query is modeled as a score. Let $t_i \in T$ be a tuple in the database, defined over a relation schema $\mathcal{A} = \langle A_1, \ldots, A_M \rangle$, where $A_1, \ldots, A_M$ are attributes. Let $s(t_i)$ denote the score of tuple t_i, computed by applying a *scoring function* over t_i's attribute values. Generally, $s(t_i)$ is computed by using an aggregation function

$$s(t_i) = F(s(t_i[A_1]), \ldots, s(t_i[A_M]); w_1, \ldots, w_M), \quad (1)$$

where $t_i[A_j]$ is the value of the j-th attribute of t_i, $s(t_i[A_j])$ its relevance with respect to the query, and w_j is the weight associated with the j-th attribute, i.e., the importance of A_j with respect to the user needs. It is common to define (1) as a convex sum of weighted attribute scores [41].

When both the attribute values and the corresponding weights are known, the tuples in T can be totally ordered in descending order of $s(t_i)$ by breaking ties deterministically. Instead, if either the attribute values or the weights are uncertain, the score $s(t_i)$ can be modeled as a random variable. The corresponding probability density function (pdf) f_i can be obtained either analytically, from the knowledge of the domain, or by fitting training data [42], [36], [5].

In the following we focus on the case in which f_i represents a continuous random variable, from which the simpler discrete case can be derived. We make very weak assumptions on the class of pdf's: f_i can be any function that can be approximated with a piecewise polynomial function defined over a finite support $[l_i, u_i]$, where l_i and u_i are the lowest and highest values that can be attained by $s(t_i)$. This approximation allows us to handle the most common probability distributions [25]. For ease of presentation, from now on we focus on uniform probability distributions. Let then δ_i denote the spread of the score distribution associated with the tuple t_i, i.e., $\delta_i = u_i - l_i$. Without loss of generality, we assume that the scores are normalized in the $[0, 1]$ interval. Therefore, $l_i \in [0, 1 - \delta_i]$ and $u_i \in [\delta_i, 1]$. Figure 1(a)

illustrates an example with three tuples whose score is represented by means of a uniform pdf.

The uncertain knowledge of the scores $s(t_i)$ induces a partial order over the tuples [42]. Indeed, when the pdf's of two tuples overlap, their relative order is undefined. Therefore, we define the *space of possible orderings* as the set of all the total orderings compatible with the given score probability functions. This space can be represented by means of a *tree of possible orderings* (henceforth: TPO), in which each node (except the root) represents a tuple t_i, and an edge from t_i to t_j indicates that t_i is ranked higher than t_j (denoted $t_i \prec t_j$). Each path $t_{r(1)} \prec t_{r(2)} \prec \ldots \prec t_{r(N)}$, where $r(k)$ is the index of the tuple ranked at position k, is associated with a probability value. Complete paths from the root (excluded) to the leaf $t_{r(N)}$ (included) represent a possible ordering of the underlying set of tuples T. For example, the score distributions in Figure 1(a) induce the TPO in Figure 1(b) , where each ordering is associated with its probability value.

In the next three sections we define and show how to build a model that represents and quantitatively measures the uncertainty in the space of possible orderings. In Section 3, first we summarize how to build the TPO starting from the score pdf's via state-of-the-art techniques, then show how to remove irrelevant orderings from the TPO. In Section 4 we propose different measures that quantify the level of uncertainty in a TPO. Based on this, in Section 5 we introduce a framework that determines the sequence of questions to ask for reducing uncertainty in top-K query scenarios.

3 TREE OF POSSIBLE ORDERINGS (TPO)

3.1 Building the TPO

A method for constructing a TPO $\mathcal{T}$ was proposed in [42]. Let T be a table containing the tuples with uncertain score $\{t_1, \ldots, t_N\}$. In order to build the tree, a dummy root node is created. Then, the sources (i.e., tuples $t_i \in T$ such that there does not exist any $t_j \in T$ such that $l_j > u_i$) are extracted from T and attached as children of the root. Next, each extracted source is used as a root for computing the next level of the tree. The asymptotic time complexity of building the tree up to level K is $O(KN^2)$. Finally, the probability $\Pr(\omega)$ of any ordering ω in the tree can be computed, e.g., with the *generating functions* technique [25] with asymptotic time complexity $O(N^2)$, or via Monte Carlo sampling.

Figure 1(b) shows the TPO obtained from the score distributions in Figure 1(a), along with the probability of each ordering, indicated next to each leaf. Each internal node n at depth d is associated with a probability $\Pr(n)$, obtained by summing up the probabilities of the children of n; such a value denotes the probability of the prefix of length d formed by the nodes along the path from the root to node n.

3.2 Limiting the TPO to depth K

We observe that processing a top-K query over uncertain data only requires computing the orderings of *the first* K tuples compatible with the pdf's of the tuple scores. In other words, when a top-K query is posed, only the sub-tree $\mathcal{T}_K$ of possible orderings up to depth K is relevant to answer the query. Building the complete tree $\mathcal{T}$ of depth N is thus unneeded, as the probabilities $\Pr(\omega_K)$ for each $\omega_K \in \mathcal{T}_K$ can be computed without knowing $\mathcal{T}$ and its probabilities, and thus much more efficiently. Indeed, as discussed in Section 6.2, while $|\mathcal{T}|$ increases exponentially with N and δ, $|\mathcal{T}_K|$ is typically slightly larger than K. Figure 2(a) shows an example TPO with 4 tuples; Figure 2(b) shows the same TPO when only the first $K = 2$ levels are considered.

3.3 Pruning the TPO

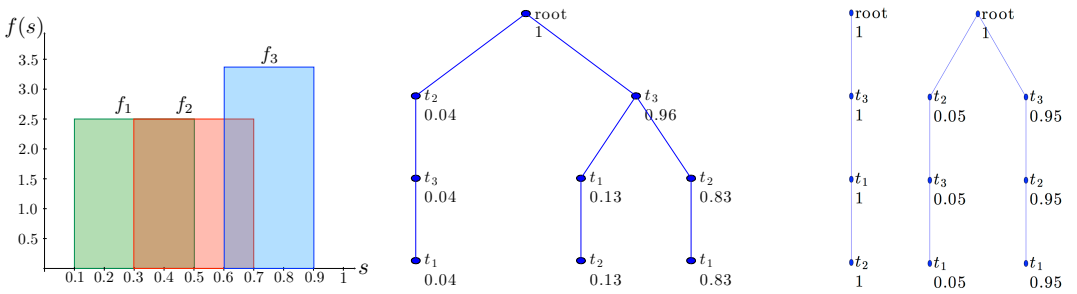
If the relative order of two tuples in a TPO is known, e.g., as determined by a crowd answer assumed to be correct, we can prune all the paths incompatible with such an order.

In particular, when considering two tuples t_i and t_j in the full TPO $\mathcal{T}$ (i.e., when $K = N$), each path in $\mathcal{T}$ agrees either with $t_i \prec t_j$ or with $t_i \not\prec t_j$. Thus, $\mathcal{T}$ can be partitioned into two sub-trees: *i)* $\mathcal{T}^{t_i \prec t_j}$, which contains all the paths (from the root to a leaf) in $\mathcal{T}$ in which t_i is ranked higher than t_j, and *ii)* $\mathcal{T}^{t_i \not\prec t_j}$, which contains all the remaining paths. Figure 1(c) shows two sub-trees derived from the tree in Figure 1(b) when either $t_1 \prec t_2$ or $t_1 \not\prec t_2$. Note that the leaf probabilities are always normalized so that they sum up to 1, i.e., each probability $\Pr(\omega)$ in a sub-tree $\mathcal{T}' \in \{\mathcal{T}^{t_i \prec t_j}, \mathcal{T}^{t_i \not\prec t_j}\}$ is recomputed as $\frac{\Pr(\omega)}{\sum_{\omega' \in \mathcal{T}'} \Pr(\omega')}$. The sub-tree that agrees with the known relative order of t_i and t_j becomes the new TPO.

Instead, when $K < N$, it may happen that some orderings in $\mathcal{T}_K$ are not affected by the knowledge of the relative order of some tuples. For instance, consider the TPO $\mathcal{T}$ in Figure 2(a) and its restriction $\mathcal{T}_2$ to $K = 2$, shown in Figure 2(b). When considering the relative order of t_3 and t_4, the paths $\langle t_1, t_2 \rangle$ and $\langle t_2, t_1 \rangle$ in $\mathcal{T}_2$ belong to both $\mathcal{T}_2^{t_3 \prec t_4}$ and $\mathcal{T}_2^{t_3 \succ t_4}$, although with different probabilities, as shown in Figure 2(c). In such cases, each path ω_K in which the relative order of t_i and t_j is ambiguous must be inserted in both the sub-trees $\mathcal{T}_K^{t_i \prec t_j}$ and $\mathcal{T}_K^{t_i \succ t_j}$, and the probability of each path must be computed accordingly. In Section 5.3 we show how to generalize the definition of $\mathcal{T}_K^{t_i \prec t_j}$ and $\mathcal{T}_K^{t_i \not\prec t_j}$ when crowd answers might be noisy and the relative order of two tuples uncertain.

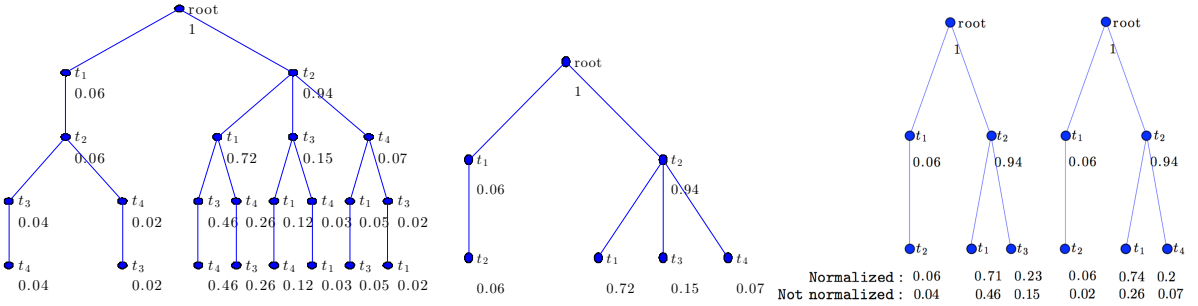
4 MEASURING UNCERTAINTY

Reducing uncertainty via crowdsourcing requires acquiring additional knowledge from the crowd. Thus



(a) Score pdf's (b) Tree of possible orderings $\mathcal{T}$ (c) Left: $\mathcal{T}^{t_1 \prec t_2}$. Right: $\mathcal{T}^{t_1 \not\prec t_2}$

Fig. 1: (a) Score pdf's for tuples t_1, t_2 and t_3; (b) their TPO $\mathcal{T}$; (c) sub-trees corresponding to the possible relative orders of t_1 and t_2. Each node is labeled with the probability of the corresponding prefix



(a) A complete TPO $\mathcal{T}$ with $N = 4$ (b) TPO $\mathcal{T}_2$ cut at $K = 2$ (c) $\mathcal{T}_2^{t_3 \prec t_4}$ (left); $\mathcal{T}_2^{t_3 \not\prec t_4}$ (right)

Fig. 2: (a) Tree of possible orderings $\mathcal{T}$; (b) its cut at depth $K = 2$; (c) possible relative orders of t_3 and t_4.

it becomes important to quantify the uncertainty reduction that can be expected by the execution of a crowd task. Given a TPO $\mathcal{T}_K$, we propose four measures to quantify its level of uncertainty. For convenience, we treat $\mathcal{T}_K$ as a set and write $\omega \in \mathcal{T}_K$ to indicate that ω is one of the orderings in $\mathcal{T}_K$, and $|\mathcal{T}_K|$ to denote the number of orderings in $\mathcal{T}_K$.

Entropy. The first measure relies on Shannon's entropy, which quantifies the average information conveyed by a source that emits symbols from a finite alphabet. In our context, the alphabet is represented by the orderings in $\mathcal{T}_K$. Each ordering $\omega \in \mathcal{T}_K$ is mapped into a symbol having probability $\Pr(\omega)$. Then, $U_H(\mathcal{T}_K)$ measures the uncertainty of $\mathcal{T}_K$, based on the probabilities of its leaves:

$$U_H(\mathcal{T}_K) = -\sum_{\omega \in \mathcal{T}_K} \Pr(\omega) \log_2 \Pr(\omega). \tag{2}$$

The procedure used for computing $U_H(\mathcal{T}_K)$ has a complexity of $O(|\mathcal{T}_K|)$.

Weighted entropy. The measure $U_H(\mathcal{T}_K)$ only considers the probabilities of the leaves of $\mathcal{T}_K$. However, in a top-K context, better ranked tuples are more important. Thus, we define weighted entropy as a weighted combination of entropy values at the first K levels of the TPO:

$$U_{H_w}(\mathcal{T}_K) = \sum_{k=1}^{K} \eta(k) U_H(\mathcal{T}_k) \tag{3}$$

where $\eta(k)$ weighs the relevance of level k, i.e., the smaller k, the larger $\eta(k)$. Computing $U_{H_w}(\mathcal{T}_K)$ has a complexity of $O(|\texttt{nodes}(\mathcal{T}_K)|)$.

ORA. The third measure is based on the idea of comparing all the orderings in $\mathcal{T}_K$ with an ordering that is *representative* in some sense. To this end, we adopt the Optimal Rank Aggregation (ORA) as the "average" ordering [41]: the ORA is the ordering with minimum average expected distance from all other orderings in $\mathcal{T}_K$. That is,

$$\omega^{\text{ORA}} = \arg\min_{\omega \in \mathcal{T}_K} \frac{1}{|\mathcal{T}_K|} \sum_{\omega_i \in \mathcal{T}_K} d(\omega_i, \omega) \Pr(\omega_i), \tag{4}$$

where $d(\cdot,\cdot)$ measures the distance between two orderings. Here we use the weighted Kendall-Tau distance [23] and adapt it to the top-K context, as follows. Let ω_1 and ω_2 be two orderings in the tree $\mathcal{T}_K$. Let us indicate with $\sigma_i(t)$ the position of tuple t in ω_i, and with $\mathcal{I}(\omega_1, \omega_2)$ the set of pairs of tuples whose order is inverted in ω_1 and ω_2, i.e.

$$\mathcal{I}(\omega_1,\omega_2) = \{(t,t') | (\sigma_1(t) - \sigma_1(t'))(\sigma_2(t) - \sigma_2(t')) < 0, \; t \in \omega_1, t' \in \omega_2\}. \tag{5}$$

We define:

$$d(\omega_1,\omega_2) = \sum_{(t,t') \in \mathcal{I}(\omega_1,\omega_2)} \pi(\sigma_1(t), \sigma_2(t)) \cdot \pi(\sigma_1(t'), \sigma_2(t')) \tag{6}$$

where $\pi(\cdot,\cdot)$ is a weight that decreases as its arguments increase. In this way, inversions involving a tuple near the head of the orderings weigh more than

near the tail. Possible choices of $\pi(\cdot,\cdot)$ are shown in Section 6.3 in Equations (17).

The average expected distance from ω^{ORA} induces an uncertainty measure:

$$U_{\text{ORA}}(\mathcal{T}_K) = \frac{1}{|\mathcal{T}_K|} \sum_{\omega_i \in \mathcal{T}_K} d(\omega_i, \omega^{\text{ORA}}) \Pr(\omega_i) \quad (7)$$

Computing $U_{\text{ORA}}(\mathcal{T}_K)$ has a complexity of $O(|\mathcal{T}_K|^2)$.

MPO. The last measure is similar to U_{ORA} but refers to another representative ordering, i.e., the Most Probable Ordering (MPO) [41]:

$$\omega^{\text{MPO}} = \arg\max_{\omega \in \mathcal{T}_K} \Pr(\omega). \quad (8)$$

In turn, this induces an uncertainty measure:

$$U_{\text{MPO}}(\mathcal{T}_K) = \frac{1}{|\mathcal{T}_K|} \sum_{\omega_i \in \mathcal{T}_K} d(\omega_i, \omega^{\text{MPO}}) \Pr(\omega_i) \quad (9)$$

Computing $U_{\text{MPO}}(\mathcal{T}_K)$ has a complexity of $O(|\mathcal{T}_K|)$.

In the following, we will drop the subscript and simply write $U(\mathcal{T}_K)$ to denote the uncertainty of a TPO, because the implementation of our algorithms for uncertainty reduction, discussed in Section 5, is orthogonal to the specific way of measuring uncertainty. However, the impact of the adopted measure is studied experimentally in Section 6.

5 Humans fighting Uncertainty

The number of possible orderings in a TPO (i.e., paths from the root to a leaf in $\mathcal{T}$) depends on the number of tuples N and on the overlaps of their pdf's f_i, $i = 1,\ldots,N$, and can be very large even for small values of N, as we will show in Section 6. We have two main ways of reducing uncertainty in $\mathcal{T}$ to quickly converge to the correct ordering: *i)* building only the first K levels of the TPO, as was shown in Section 3.2, thereby focusing on $\mathcal{T}_K$ instead of $\mathcal{T}$, and *ii)* defining crowd tasks for disambiguating the relative order of tuples in order to prune the TPO as shown in Section 3.3.

Therefore we consider crowd tasks expressed as questions of the form $q = t_i \precsim t_j$, which compare t_i and t_j to determine which one ranks higher. We initially assume that a crowd worker's answer $ans(q)$, which is either $t_i \prec t_j$ or $t_i \not\prec t_j$,[1] always reflects the real ordering of the tuples (we shall relax this assumption in Section 5.3). With that, we can prune from $\mathcal{T}_K$ all the paths disagreeing with the answer.

5.1 Problem definition

We now focus on the problem of Uncertainty Resolution (UR) and use $\mathcal{T}_K$ as a starting point of our investigation, knowing that $\mathcal{T}_K$ can be built with the techniques described in the previous sections. For convenience, let $ans_\omega(q)$ indicate the answer to question q compatible with an ordering $\omega \in \mathcal{T}$. Also, for a sequence of answers $\mathbf{ans} = \langle ans_1,\ldots,ans_n\rangle$, let $\mathcal{T}_K^{\mathbf{ans}}$ indicate the tree $(\ldots(\mathcal{T}_K^{ans_1})\ldots)^{ans_n}$ obtained by pruning the TPO $\mathcal{T}_K$ accordingly. Formally, an underlying real ordering ω is part of the problem definition, although ω does not need to be known in practice.

Definition 5.1: A UR *problem* is a pair $\langle \mathcal{T}_K, \omega\rangle$, where $\mathcal{T}_K$ is a TPO and ω is an ordering of all tuples in $\mathcal{T}_K$, called *real ordering*, such that an ordering $\omega_K \in \mathcal{T}_K$ is a prefix of ω. A *solution* to $\langle \mathcal{T}_K, \omega\rangle$ is a sequence of questions $\langle q_1,\ldots,q_n\rangle$ such that $\mathcal{T}_K^{\langle ans_\omega(q_1),\ldots,ans_\omega(q_n)\rangle}$ only contains ω_K. A solution is *minimal* if no shorter sequence is also a solution.

We consider two classes of algorithms: *i)* offline algorithms, which determine the solution a priori, before obtaining any answer from the crowd, and *ii)* online algorithms, whereby the solution is determined incrementally as the answers to previous questions arrive. These classes reflect two common situations in crowdsourcing markets: one where the interaction is limited to the publication of a batch of tasks, which is evaluated for acceptance as a whole; and one where the employer can inspect the outcome of crowd work as it becomes available and incrementally publish further tasks.

The difference between these two classes lies in the ability of each online algorithm A to probe the real ordering so as to choose at each step a new question, via a function ϕ_A, according to the answers to previously posed questions.

Definition 5.2: A UR algorithm A takes as input a pair $\langle \mathcal{T}_K, B\rangle$, where $\mathcal{T}_K$ is a TPO and B (the *budget*) is a nonnegative integer, and outputs a sequence $\mathcal{Q} = \langle q_1,\ldots,q_B\rangle$ of questions on the tuples in $\mathcal{T}_K$. A is *offline* if $\mathcal{Q}$ is a function of $\langle \mathcal{T}_K, B\rangle$. A is *online* if there is a function ϕ_A such that, for any real ordering ω and for $1 \le i \le B$, we have $q_i = \phi_A(\langle \mathcal{T}_K, \langle q_1,\ldots,q_{i-1}\rangle, \langle ans_\omega(q_1),\ldots,ans_\omega(q_{i-1})\rangle\rangle)$.

An algorithm is optimal if it always finds a minimal solution.

Definition 5.3: A UR algorithm A is *optimal* if, for every UR problem $\mathcal{P} = \langle \mathcal{T}_K, \omega\rangle$ and minimal solution $\mathcal{Q}$ of $\mathcal{P}$, A's output on $\langle \mathcal{T}_K, |\mathcal{Q}|\rangle$ is a solution of $\mathcal{P}$.

Theorem 5.4: No deterministic algorithm is optimal.

Proof: Let $T = \{t_1, t_2, t_3\}$ be such that t_1 dominates t_3, while t_2's pdf overlaps with both t_1's and t_3's. Only three orderings are possible: $\omega_1 = \langle t_1, t_2, t_3\rangle$, $\omega_2 = \langle t_1, t_3, t_2\rangle$, and $\omega_3 = \langle t_2, t_1, t_3\rangle$. The only two questions on overlapping tuples are: $q_1 = t_1 \precsim t_2$ and $q_2 = t_2 \precsim t_3$. Each deterministic algorithm must commit to a choice of either q_1 or q_2 as the first question, with no prior knowledge on the real ordering. However, if the real ordering is ω_2, each deterministic algorithm choosing q_1 as the first question fails to identify the correct ordering with just one question, which could have been done by choosing q_2. Analogously, if the real ordering is ω_3, each deterministic

1. We assume a deterministic rule known to the crowd worker for breaking ties and thus write equivalently $t_i \not\prec t_j$ and $t_j \prec t_i$.

algorithm choosing q_2 fails in a similar way (q_1 suffices to identify the real ordering). Therefore, for every deterministic algorithm there is a UR problem whose minimal solution consists of one question while the algorithm's solution includes two questions. □

5.2 Selecting the best questions

With the result of Theorem 5.4 we cannot hope to find an optimal algorithm. Therefore we now turn our attention to an attainable form of optimality that is of a probabilistic nature, in that it refers to the *expected* amount of uncertainty that remains in the TPO after posing the questions selected by an algorithm. For this reason, we introduce the notion of residual uncertainty as a means of finding the best sequence of questions for a given TPO $\mathcal{T}_K$. Clearly, the only relevant questions (set $\mathcal{Q}_K$) are those that compare tuples with an uncertain relative ordering, i.e., whose pdf's overlap:

$$\mathcal{Q}_K = \{t_i \,?\!\prec\, t_j \mid t_i, t_j \in \mathcal{T}_K \wedge \texttt{overlap}(f_i, f_j) \wedge i < j\} \tag{10}$$

The average *residual uncertainty* $\mathcal{R}_{\mathcal{Q}}(\mathcal{T}_K)$ that can be expected after a crowd worker has answered the questions $\mathcal{Q}$ can be estimated as follows:

$$\mathcal{R}_{\mathcal{Q}}(\mathcal{T}_K) = \sum_{\mathbf{ans}} \Pr(\mathbf{ans}) U(\mathcal{T}_K^{\mathbf{ans}}) \tag{11}$$

where $\mathbf{ans} = \langle ans_1, \ldots, ans_{|\mathcal{Q}|} \rangle$ ranges over all possible sequences of answers for the questions in $\mathcal{Q}$, $\mathcal{T}_K^{\mathbf{ans}}$ is the pruned tree obtained from $\mathcal{T}_K$ according to the answers $\mathbf{ans}$, and $\Pr(\mathbf{ans}) = \Pr(\bigwedge_{j=1}^{|\mathcal{Q}|} ans(q_j) = ans_j)$ is the probability that each answer ans_j in $\mathbf{ans}$ is the actual worker's answer to question $q_j \in \mathcal{Q}$. For a single question $\mathcal{Q} = \{t_i \,?\!\prec\, t_j\}$, we have $\mathcal{R}_{\mathcal{Q}}(\mathcal{T}_K) = \Pr(t_i \prec t_j) U(\mathcal{T}_K^{t_i \prec t_j}) + \Pr(t_i \not\prec t_j) U(\mathcal{T}_K^{t_i \not\prec t_j})$.

In crowdsourcing applications, restrictions in the budget used for rewarding workers or in the available time allotted for collecting answers usually limit the number of questions that can be posted to the crowd. In practical scenarios, such restrictions are yet more significant if the crowd is noisy and thus the same question needs to be posed to several workers to increase confidence. Let B denote the maximum number of questions (budget) that can be asked to the crowd workers. Our goal is then to select the best sequence of questions $\mathcal{Q}^* = \langle q_1^*, \ldots, q_B^* \rangle \subseteq \mathcal{Q}_K$ that causes the lowest amount of expected residual uncertainty. Based on this, we introduce a relaxed version of optimality for offline algorithms, and show how to attain it. Then we discuss sub-optimal, but more efficient algorithms.

5.2.1 Offline question selection strategies

We first observe that an offline algorithm determines all the questions to pose before obtaining any answer from the crowd. Therefore, permuting the order in which the questions selected by such an algorithm are asked always leads to the same uncertainty reduction on the tree $\mathcal{T}_K$. The order of questions is thus immaterial, and we shall then consider that the output of an offline algorithm is simply a set (instead of a sequence) of questions.

Definition 5.5: An offline UR algorithm is *offline-optimal* if it outputs the set of B questions $\mathcal{Q}^*$ that minimizes the expected residual uncertainty, i.e.:

$$\mathcal{Q}^* = \arg \min_{\mathcal{Q} \,:\, |\mathcal{Q}| = B \wedge \mathcal{Q} \subseteq \mathcal{Q}_K} \mathcal{R}_{\mathcal{Q}}(\mathcal{T}_K). \tag{12}$$

Best-first search offline algorithm ($\texttt{A}^*-\texttt{off}$). An implementation of an offline-optimal algorithm can be obtained by adapting the well-known A* best-first search algorithm [12] to UR. $\texttt{A}^*-\texttt{off}$ explores the solution space with the help of a *solution tree* defined as follows: *i)* each node n is associated with a gain function $f(n)$, which determines the priority of the node in the search process for the optimal solution; *ii)* each edge outgoing from n represents the choice of a question $q \in \mathcal{Q}_K$; *iii)* the maximum tree depth is B; *iv)* for each pair of paths P and R of B questions, $|P \cap R| < B$ (so as to avoid considering the same set of questions multiple times). Let n be a node in the solution tree at depth d and let $\mathcal{Q}_n$ denote the sequence of questions on the path between the root and n. The gain function $f(n)$, defined as knowledge plus heuristic, is $f(n) = g(n) + h(n)$. The *knowledge* $g(n)$ is given by the uncertainty reduction expected by asking the questions in $\mathcal{Q}_n$:

$$g(n) = U(\mathcal{T}_K) - \mathcal{R}_{\mathcal{Q}_n}(\mathcal{T}_K). \tag{13}$$

Since the contribution to uncertainty reduction of a set of dependent questions is at most the sum of the single contributions (and at most the residual uncertainty), the following *heuristic* $h(n)$ is an upper bound on the uncertainty reduction expected by asking $(B - d)$ more questions:

$$h(n) = \min\{\mathcal{R}_{\mathcal{Q}_n}(\mathcal{T}_K), \sum_{q \in \mathcal{Q}_{\text{best}}} \mathcal{R}_{\mathcal{Q}_n}(\mathcal{T}_K) - \mathcal{R}_{\mathcal{Q}_n \circ \langle q \rangle}(\mathcal{T}_K)\} \tag{14}$$

where $\mathcal{Q}_{\text{best}}$ is the set of $(B - d)$ questions (obtained by enumeration) that, if asked after $\mathcal{Q}_n$, guarantee the highest uncertainty reduction. A* traverses the solution tree keeping a sorted priority queue of nodes. The higher the expected gain $f(n)$ achieved by traversing a node, the higher the node priority. Thus, high priority nodes will be explored before others. When the algorithm traverses a path of length B, the questions on that path become the selected set $\mathcal{Q}^*$.

Theorem 5.6: $\texttt{A}^*-\texttt{off}$ is offline-optimal.

Offline-optimality follows immediately from the fact that A* is *complete* [12], and thus considers all candidate solutions, while retaining only the optimal one. Yet, $\texttt{A}^*-\texttt{off}$ is computationally very expensive, so we shall also consider two sub-optimal, but more efficient algorithms.

Algorithm 1: Top-1 online algorithm (T1−on)

Input: *TPO* $\mathcal{T}_K$, *Budget* B
Output: *Optimal sequence of questions* $\mathcal{Q}^*$
Environment: *Underlying real ordering* ω
1) $\mathcal{Q}^* := \emptyset$;
2) **for** $i := 1$ to B
3) **if** $|\mathcal{T}_K| = 1$ **then break**;
4) $q_i^* := \arg\min_{q \in \mathcal{Q}_K \setminus \mathcal{Q}^*} \mathcal{R}_{\langle q \rangle}(\mathcal{T}_K)$; // *see Equation* (11)
5) $\mathcal{Q}^* := \mathcal{Q}^* \circ \langle q_i^* \rangle$; // *appending the selected question*
6) Ask q_i^* to the crowd and collect the answer $ans_\omega(q_i^*)$
7) $\mathcal{T}_K := \mathcal{T}_K^{ans_\omega(q_i^*)}$; // *updating the TPO*
8) **return** $\mathcal{Q}^*$;

Top-B offline algorithm (TB−off). This simpler method computes for each question $q \in \mathcal{Q}_K$ the expected residual uncertainty $\mathcal{R}_q(\mathcal{T}_K)$. Then, we sort the questions in ascending order of $\mathcal{R}_q(\mathcal{T}_K)$ and define $\mathcal{Q}^*$ as the set of top B questions.

Conditional offline algorithm (C−off). This method iteratively selects one question at a time based on the previous selections. Let $\{q_1^*, \ldots, q_i^*\}$ be the first i selected questions ($\emptyset$ when $i = 0$). The $(i+1)$-th question q_{i+1}^* is selected by C−off from $\mathcal{Q}_K \setminus \{q_1^*, \ldots, q_i^*\}$ so as to minimize $\mathcal{R}_{\langle q_1^*, \ldots, q_i^*, q_{i+1}^* \rangle}(\mathcal{T}_K)$, i.e., the residual uncertainty conditioned by the choice of the previously selected questions $q_1^*, \ldots, q_i^*$. The final output is thus $\mathcal{Q}^* = \{q_1^*, \ldots, q_B^*\}$.

5.2.2 Online question selection strategies

An online algorithm has the ability to determine the i-th question based on the answers collected for all the previously asked $i-1$ questions. Differently from the offline case, the output of an online algorithm is treated as a sequence and not as a set, since each received answer may influence the choice of the next question, and thus the order matters.

Best-first search online algorithm (A^*−on). An online UR algorithm can be obtained by iteratively applying A^*−off B times. At the i-th step, $1 \leq i \leq B$, A^*−on identifies the i-th question q_i^* in its output and asks it to the worker, thus obtaining the answer $ans_\omega(q_i^*)$. Question q_i^* is simply the first element of the sequence $\mathcal{Q}_i^*$ returned by A^*−off for the TPO $\mathcal{T}_K^{\langle ans_\omega(q_1^*), \ldots, ans_\omega(q_{i-1}^*) \rangle}$ with budget $(B-i+1)$, where ω is the real ordering and $q_1^*, \ldots, q_{i-1}^*$ are the previously selected questions (initially, A^*−off is applied on $\mathcal{T}_K$ with budget B and the first question in its output is chosen as q_1^*). Intuitively, each step chooses the most promising question q_i^* within the horizon of the remaining $B - i + 1$ questions to be asked based on the current knowledge of ω. Note that, as new answers arrive, the next most promising questions might no longer coincide with the rest of the previously planned sequence $\mathcal{Q}_i^*$.

Being based on A^*−off, A^*−on is costly. Thus, we also consider a simpler but more efficient online algorithm.

Top-1 online algorithm (T1−on). Algorithm 1 illustrates the T1−on algorithm, which builds the sequence of questions $\mathcal{Q}^*$ iteratively until the budget B is exhausted (line 2). At each iteration, the algorithm selects the best (Top-1) unasked question, i.e., the one that minimizes the expected residual uncertainty with budget $B = 1$ (line 4). The selected question q_i^* is then appended to $\mathcal{Q}^*$ and asked to the crowd. Depending on the answer, the TPO $\mathcal{T}_K$ is updated to the subtree that agrees with the answer to q_i^* (line 7). Early termination may occur if all uncertainty is removed, i.e., the tree is left with a single path (line 3).

5.3 Handling noisy workers

In a crowdsourcing scenario, the collected answers might be noisy. Let p denote a crowd worker's accuracy (i.e., the probability that his/her answer is correct). Handling this case requires a simple redefinition of the trees $\mathcal{T}_K^{t_i \prec t_j}, \mathcal{T}_K^{t_i \not\prec t_j}$. When $p < 1$, both trees will represent the whole set of possible orderings included in $\mathcal{T}_K$, but the probabilities of the orderings need to be adjusted. Let $\Pr(\omega)$ and $\Pr(\omega | ans_q = t_i \prec t_j)$ denote, respectively, the probability of the same ordering ω in $\mathcal{T}_K$ and $\mathcal{T}_K^{t_i \prec t_j}$. Then, by Bayes' theorem [52]

$$\begin{aligned} \Pr(\omega | ans_q = t_i \prec t_j) &= \frac{\Pr(ans_q = t_i \prec t_j | \omega) \Pr(\omega)}{\Pr(ans_q = t_i \prec t_j)} \\ &= \frac{\Pr(ans_q = t_i \prec t_j | \omega) \Pr(\omega)}{p \Pr(t_i \prec t_j) + (1-p) \Pr(t_i \not\prec t_j)}, \end{aligned} \tag{15}$$

where $\Pr(ans_q = t_i \prec t_j | \omega) = p$, if $t_i \prec t_j$ in ω; otherwise, $1 - p$; similarly for $\mathcal{T}^{t_i \not\prec t_j}$.

5.4 Incremental algorithm

The number of orderings in a TPO can be large if there are many overlaps in the tuple score distributions, thereby affecting the execution time of our algorithms. Hence, we propose the incr algorithm, shown in Algorithm 2, that does not receive as input a TPO $\mathcal{T}_K$. Instead, it builds the TPO incrementally, one level at a time, by alternating tree construction with a round of n questions and tree pruning. The number n of questions posed at each round is between 1 and B, therefore incr can be considered a hybrid between an online and an offline algorithm.

Each TPO $\mathcal{T}_k$, $1 \leq k \leq K$, is built by adding one level to the previous TPO $\mathcal{T}_{k-1}$ (line 10), i.e., by attaching to each ordering ω_{k-1} in $\mathcal{T}_{k-1}$ the unused sources as leaves. We only build new levels if there are not enough questions to ask (line 5), i.e., n questions for all the rounds but the last one, where $B \mod n$ questions are asked (line 3). Then, we select the best questions, pose them to the crowd, collect the answers and apply the pruning accordingly (lines 6–9), until either the budget B is exhausted or the TPO is entirely built (line 2). We thus keep the TPO as pruned as

Algorithm 2: Incremental algorithm (`incr`)

Input: *Tuple set T, Budget B, Depth K, Questions per round n*
Output: *Optimal sequence of questions $\mathcal{Q}^*$*
Environment: *Underlying real ordering ω*

```
1) Q* := ∅; ans := ∅; k := 1; T_1 := TPO with only the first level;
2) while |ans| < B and k < K
3)     n' := min(n, B − |ans|); // # of questions at next round
4)     Q_k := relevant questions for T_k; // as in Equation (10)
5)     if |Q_k \ Q*| ≥ n' // there are enough relevant questions to ask
6)         ⟨q_1, ..., q_n'⟩ := arg min_{Q⊆Q_k\Q*,|Q|=n'} R_Q(T_k); // as in (11)
7)         Q* := Q* ∘ ⟨q_1, ..., q_n'⟩; // appending the selected questions
8)         ans := ans ∘ ⟨ans_ω(q_1), ..., ans_ω(q_n')⟩ // appending answers
9)         T_k := T_k^ans; // updating the TPO
10)    else k := k + 1; T_k := T_{k−1} extended by one level;
11) return Q*;
```

possible, and only proceed to computing the next level when the uncertainty in the previous levels is so low that it does not require n questions to ask.

6 Experimental evaluation

In this section we evaluate the proposed uncertainty reduction methods on several synthetic and two real datasets, and collect answers through a real crowdsourcing platform.

First, we exploit the synthetic datasets to investigate the impact of uncertainty; the study shows that even small sizes of the dataset ($N < 100$) might lead to an extremely large number of possible orderings. This justifies the need for considering top-K query results, which dramatically reduce the number of orderings by restricting the analysis to the tuples occurring in the first K levels of the tree (Figure 3).

Then, we compare the online, offline and incremental methods described in Section 5 on uncertain datasets characterized by thousands of possible orderings of top-K tuples (Figure 5). For completeness, we include in our analysis the comparison with two simple algorithms used as baselines: `Random` and `Naive`.

The `Random` algorithm returns a sequence of B different questions chosen completely at random among all possible tuple comparisons in $\mathcal{T}_K$.

The `Naive` algorithm avoids irrelevant questions by returning a sequence of B questions chosen randomly from $\mathcal{Q}_K$ (see (10)), i.e., from all the possible comparisons between tuples in $\mathcal{T}_K$ *that have overlapping pdf's*.

6.1 Datasets and evaluation measures

Synthetic datasets. The synthetic datasets consist of collections of tuples with uncertain scores. The score of each tuple $s(t_i)$ is uniformly distributed within the interval $[l_i, u_i]$. We used $\delta = u_i - l_i$ as a parameter to tune the level of uncertainty. For each tuple, the value l_i is sampled at random in the interval $[0, 1-\delta]$ from a uniform or a Zipfian distribution, such that $0 \leq l_i < u_i \leq 1$. For each configuration of the parameters, we generate 10 instances so as to compute the average performance. The real ordering ω_r of the tuples is simulated by sampling, for each tuple, a score value from the corresponding pdf, and sorting tuples in decreasing order of score value. Such an ordering corresponds to a path of the TPO $\mathcal{T}$.

YouTube dataset. Real datasets are often characterized by tuples whose score uncertainty cannot be represented with a uniform distribution. As a concrete example, we considered the case of videos downloaded from YouTube. Their upload timestamps are an uncertain indication of the occurrence time of the event captured by each video. Therefore, we estimated the distribution $f_{\texttt{offset}}$ describing the temporal offset between the upload time and the event time from a training dataset. The dataset was obtained by downloading 3000 videos in response to keyword queries referring to events happened in October-November 2012, either unexpected (e.g., earthquakes) or announced (e.g., presidential elections). We processed this dataset so as to: *a)* manually remove the irrelevant videos returned in response to the keyword queries, and *b)* shift the timestamps of the videos by setting the origin ($t = 0$) to the actual occurrence time of the event. After preprocessing, the training dataset contained 939 videos, whose timestamps were used to estimate $f_{\texttt{offset}}$. In order to assess the quality of workers on a real crowdsourcing platform, we also downloaded 180 videos related to 36 events of 2014, and 210 videos related to 42 events of 2013 (5 per event). We then extracted a set of relevant questions about the videos, asking the correct temporal ordering of two events. Finally, the collected instances were used as a test dataset, which contains 5 sets of $N = 78$ videos. For each set of videos a TPO is built ($K = 10$) by assigning to each video an uncertain occurrence time, modeled with a non-uniform distribution $f_{\texttt{offset}}$ centered in the upload time.

Image quality dataset. The second real dataset consists of a collection of images affected by different types of distortion (e.g., blur, Gaussian noise, JPEG compression, etc.) extracted from the IVC dataset [24]. Each image comes with a Mean Opinion Score (MOS) value (a number in the range $[0, 100]$), which can be used to determine the real ordering ω_r of the images based on their quality. MOS values are difficult to obtain, since they are the result of the aggregation of scores provided by individuals that take part in time-consuming subjective evaluation campaigns. For this reason, objective image quality metrics have been proposed in the literature to automatically assign quality scores. For example, the SSIM metrics [48] evaluates the difference between an image I_d and its original (distortion-free) version I_o, and produces as output an objective quality metrics for I_d. Due to the different kinds of distortion affecting digital images and the complex task of modeling visual perception, objective metrics (including SSIM) provide an approximate (uncertain) indication of image quality. In [48], a mapping

Full name	Parameter	Tested values
Size of dataset	N	100, 500, **1000**, 10000, 100000, 1000000
Number of results	K	1, 3, 5, 7, **10**
Question budget	B	5, **10**, 20, 30, 40, 50
Score probability distribution spread	δ	1e-6, 1e-5, 1e-4, 1e-3, **3e-3**
Measure of uncertainty	U	entropy (U_H), weighted entropy (U_{H_w}), ORA (U_{ORA}), **MPO** (U_{MPO})
Accuracy of annotators	p	0.8, 0.9, **1**
Number of questions per round for `incr`	n	1, 3, **5**, 7, 10, 15, 20, 25

TABLE 1: Operating parameters for synthetic datasets (defaults used in all figures are in bold)

between the objective/subjective scores is provided, showing that a single SSIM value may correspond to a range of MOS values, comprised on average within an interval of 10 MOS units. This finding allows modeling uncertainty in image quality scores with a uniform probability distribution f_q centered on the SSIM value, with a spread δ of 10 MOS units. We considered 10 different sets of $N = 15$ images each, with the goal of determining the top-K ($K = 3$) images in each set based on their quality. The proposed methods were used to determine which questions to ask, i.e., which pairs of images need to be compared to sort images according to their visual quality. This dataset is characterized by a very high level of uncertainty due to difficulties in devising objective quality metrics mimicking the human visual system. Increasing N would thus lead to compare pairs of images with too similar a quality, thus being adversely affected by the subjective nature of the task.

Evaluation measures. In the experiments, we assess performance of the various algorithms by comparing the average distance from the real ordering in the obtained TPO. Note that, if uncertainty is measured as a distance from a representative ordering (such as ORA or MPO), which in turn is a probabilistic proxy for the real ordering, then minimizing the residual uncertainty indeed amounts to minimizing an expectation of the distance from the real ordering.

The average distance $D(\omega_r, \mathcal{T}_K)$ between the real ordering ω_r and the orderings in $\mathcal{T}_K$ is given by:

$$D(\omega_r, \mathcal{T}_K) = \sum_{\omega \in \mathcal{T}_K} \Pr(\omega) d(\omega_r, \omega), \tag{16}$$

where $d(\omega_r, \omega)$ is a distance function. We use Kendall-Tau distance for the Youtube dataset and weighted Kendall-Tau distance for all other datasets.

6.2 Uncertainty and problem size

We start by examining the impact of uncertainty on full datasets containing all the N tuples. Note that the amount of uncertainty depends on the combination of N and δ (i.e., the spread of the score distribution) : given a fixed value of δ, the number of overlaps among the pdf's increases with N, since the score distributions are more densely spaced in $[0, 1]$. The size of the problem, measured in terms of the number of orderings in $\mathcal{T}$, grows exponentially in both N and δ. Thus, even with relatively small values of N, the number of orderings quickly becomes intractable when increasing δ. For example, when $N = 100$ and $\delta = 0.001$, there are 12 overlapping pairs of tuples, which generate as many as $|\mathcal{T}| = 4000$ possible orderings , while increasing N to just 110 tuples makes the number of orderings grow by an order of magnitude.

However, users are mostly interested in the top-K results only. Indeed, regardless of N, we observed that the number of tuples in the first K levels of $\mathcal{T}$ is only slightly larger than K, growing slowly when increasing δ. Figure 3 shows the impact of uncertainty when one focuses on $\mathcal{T}_K$, for $K = 10$ and several different values of N and δ. Both the number of overlaps of tuples and the number of orderings grow with N, although in a much more tractable manner: for $N = 100$ and $\delta = 0.001$ the average number of orderings is $|\mathcal{T}_K| = 1.2$, while for $N = 1000$ and $\delta = 0.001$ the average number of overlaps is 10, leading to $|\mathcal{T}_K| = 273$ ($\mathcal{T}_K$ built in $13s$). A similar problem size is obtained, e.g., when $N = 10^6$ and $\delta = 10^{-6}$ ($|\mathcal{T}_K| = 203$). To test our algorithms, we chose a default scenario with high uncertainty: N=1000, δ=0.003 (32 overlaps on average, $|\mathcal{T}_K| = 39,330$, and $\mathcal{T}_K$ built in more than 5 hours).

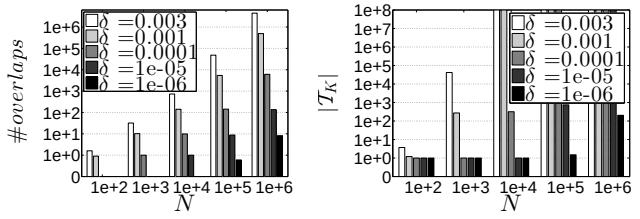
Although there are combinations of the parameters N and δ for which the number of orderings is exceedingly large, they correspond to cases of little practical interest. Indeed, in such cases, the pdf's of the tuples in the first K levels are nearly all overlapping with each other, indicating an extremely large amount of uncertainty in the data. In those cases, very little is known about the ordering among tuples, and other means to reduce uncertainty (e.g., aggregating/fusing data from additional sources) should be applied [53].

6.3 Comparing the methods on synthetic data

In this section we compare the effectiveness of the proposed methods in reducing the number of possible orderings, using the `Random` and `Naive` algorithms as baselines. We adopt the default values of the parameters indicated in Table 1, unless stated otherwise.

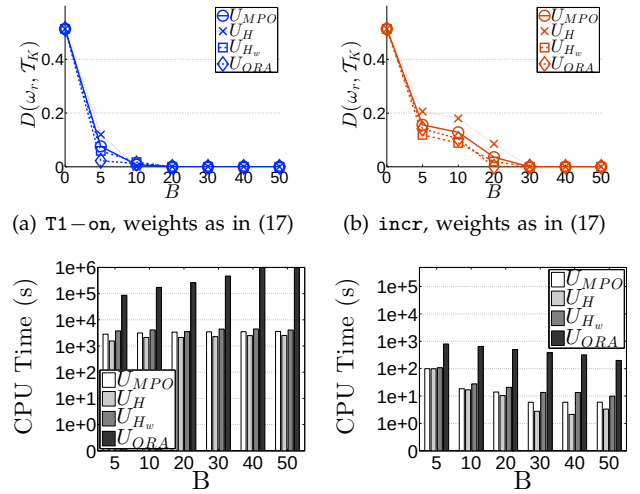
Uncertainty measure - Figure 4 shows the performance of `T1-on` and `incr` with the four different uncertainty measures (U_H, U_{H_w}, U_{MPO}, and U_{ORA}). The weights $\pi(\cdot, \cdot)$ used for the distance (6) vary logarithmically (adapted from [23] to the top-K context):

$$\pi(i,j) = \begin{cases} \frac{1}{\log(\min(i,j)+1)} & (i > K \vee \\ & j > K) \\ \frac{1}{\log_2(\min(i,j)+1)} - \frac{1}{\log_2(\max(i,j)+2)} & otherwise \end{cases} \tag{17}$$



(a) # overlaps at depth K (b) # orderings at depth K

Fig. 3: Number of overlaps and orderings in $\mathcal{T}_K$ (K=10) as dataset size N and pdf's spread δ vary



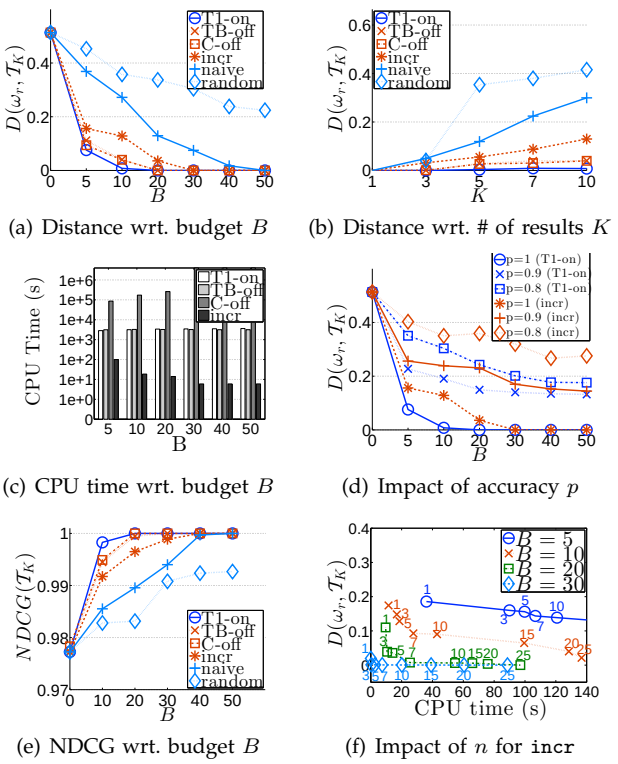
(a) `T1−on`, weights as in (17) (b) `incr`, weights as in (17)

(c) `T1−on`, CPU time (d) `incr`, CPU time

Fig. 4: Impact of uncertainty measures as B varies.

The results obtained with entropy (U_H) are consistently worse than with all the other measures, which make the algorithms converge to the real ordering much more quickly. Indeed, U_H only takes into account the probabilities of the orderings, thus neglecting the structure of the TPO and inadequately quantifying its uncertainty. For instance, with U_H, a TPO with branching only at level $K-1$ may easily be considered more uncertain than a TPO with branching close to the root. The other measures achieve similar performance (Figures 4(a), 4(b)), but U_{MPO} requires the lowest CPU time (Figures 4(c) and 4(d)) and, thus, we adopt U_{MPO} as the default. Note that Figure 4(c) omits the time to compute the TPO (more than 5 hours in the default case), which is paid by `T1−on` only once and is the same for all the measures. No such cost is incurred by `incr`.

Number of questions B - Increasing the number of questions B to ask reduces the uncertainty while converging to the underlying real ordering, as shown in Figure 5(a). Indeed, the collected answers allow discarding incompatible orderings from the TPO. The `T1−on` algorithm achieves the best performance (Figure 5(a)), converging to the real ordering with $B = 20$ questions, when considering the default



(a) Distance wrt. budget B (b) Distance wrt. # of results K

(c) CPU time wrt. budget B (d) Impact of accuracy p

(e) NDCG wrt. budget B (f) Impact of n for `incr`

Fig. 5: Performance as B, K, p and n vary.

parameters. The offline algorithms, namely `TB−off` and `C−off`, obtain the same result as `T1−on` when $B = 1$ (since in this case the strategies coincide) and eliminate all uncertainty when $B = 20$. However, since none of the answers are available when choosing the questions, `TB−off` and `C−off` are outperformed in terms of quality by `T1−on`, which has knowledge of previous answers. `C−off` achieves slightly better quality than `TB−off`, because at each step a new question is selected by taking into account the past choices. The `incr` algorithm eliminates all uncertainty when $B = 30$, thus attaining lower quality than `T1−on`, `TB−off` and `C−off`, which all require the full materialization of the TPO. Yet, `incr` is much faster, thus representing a good tradeoff between computational time and quality. The `Random` baseline yields very poor results. With respect to the `Naive` heuristics, `T1−on` attains the same performance with 50% to 80% fewer questions. Figure 5(a) reports results for uniform data distributions, but the same trends and relative strengths are observable under a Zipfian distribution (results omitted due to space constraints).

Figure 5(a) does not include the two algorithms based on A^*, namely `A*−on` and `A*−off`. This is due to their high computational complexity, which makes them impractical for a problem size using the default parameters. Hence, we repeated the same experiment with $N = 500$, $K = 7$ and $\delta = 0.001$. In this case, `T1−on` achieved nearly the same performance as `A*−on`, and `C−off` did the same with `A*−off` (in

both cases less than 1% difference in the required budget). Therefore, T1−on and C−off are preferable, in practice, to A*−on and A*−off, respectively.

Number of results K - Increasing the number of results K increases the uncertainty, as a larger number of possible orderings can be generated. Indeed, Figure 5(b) shows that, for a fixed budget of $B = 10$ questions, the distance from the real ordering increases when increasing K.

CPU time - Figure 5(c) shows the required CPU time for selecting the questions when B varies. The results show that the CPU time increases with B. Apart from incr, all algorithms require a fixed amount of time, not shown in the bars, to build the TPO; although overall very high (more than 5 hours for our default case), this is a cost paid only once before asking any question to the workers. A*−off and C−off are computationally intractable even for a small B. Conversely, TB−off exhibits the lowest overhead, because it analyzes the questions in $\mathcal{Q}_K$ only once, on the first iteration. The time required by T1−on increases with B (since multiple iterations of the algorithms are required). Instead, incr runs much faster, since the TPO is only partially materialized, and thus question selection steps are quicker. The indicated times may even decrease as B grows, since, once B questions are posed, incr may still need to build the TPO up to level K, which is more expensive for more uncertain TPO's (i.e., for lower values of B).

Accuracy of the workers - Figure 5(d) shows the impact of accuracy p for T1−on and incr. When $p = 1$ the worker always answers correctly, while when $p = 0.5$ answers are random. The proposed approach, shown in Section 5.3, never prunes the TPO. The distance from the real ordering reduces as B increases if p is reasonably high ($p \geq 0.8$). Note that majority voting or more sophisticated approaches can be conveniently employed to aggregate the answers of noisy workers, so as to attain a higher overall accuracy. Aggregating answers from multiple workers requires scaling up the budget by a non-negligible factor [40], thus making the budget savings obtained with our algorithms yet more significant.

NDCG measure. Figure 5(e) shows the performance when Normalized Discounted Cumulative Gain [17] is used to measure the quality of the output. Since NDCG measures the quality of a ranking, the performance increases as B increases, i.e., as uncertainty decreases. The relative strengths of the various algorithms are the same as in Figure 5(a).

Number of questions per round n**.** Figure 5(f) shows the performance of the incr algorithm and the required CPU time when the number of per-round questions n varies. Larger values of n entail higher quality (as larger parts of $\mathcal{T}_K$ are built and thus better questions can be chosen), and higher CPU times to build the TPO. In our tests, $n = 5$ (our default) proved a good trade-off between time and quality.

6.4 Comparing the methods on real data

YouTube dataset - Figures 6(a) and 6(b) show the results achieved by T1−on and incr when questions are asked to either an internal crowd of experts (whose accuracy is close to 100%, "expert crowd" label) or a real crowd ("real crowd" label). In the former case, the algorithms rapidly converge to the real ordering, eliminating all the uncertainty with $B = 7$ and $B = 15$, respectively. However, while the time spent by incr is limited, T1−on requires the full materialization of the tree, thus with high CPU times. The collection of answers from the real crowd is described in Section 6.5.

Image quality dataset - Figure 6(d) shows the uncertainty reduction when asking workers to compare pairs of images in terms of their visual quality. The results confirm the findings obtained on the synthetic datasets: T1−on and incr always dominate Naive. However, here Naive achieves almost the same results as Random, since the number of overlaps between the score pdf's is very high. Thus, it is very likely that a question picked at random is relevant.

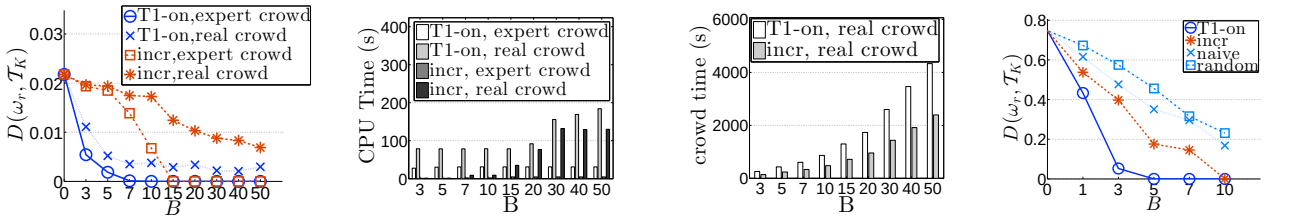
6.5 Tests on a real crowdsourcing platform

In this Section we assess T1−on and incr when questions are asked to noisy workers on a real crowd reached via the Crowdflower[2] platform.

We built a set of questions of the form *"Does A precede B?"* out of the YouTube dataset, where A and B are events with overlapping PDFs. The platform allows the inclusion of test questions to filter out spammers and low quality workers: workers answering more than 70% of the test questions incorrectly are discarded. We paid each answer 0.01$ and tried two scenarios: *i)* a limited set of videos and mild quality tests: 279 questions out of 180 videos for 2014; each question was replicated 4 times; 5 test questions; 158 workers with an overall accuracy of 69.5%; overall cost 47.77$. *ii)* a larger set of videos and stricter quality tests: 419 questions out of 390 videos for 2013 and 2014; each question was replicated 5 times; 10 test questions; 142 workers with an overall accuracy of 91.2%; overall cost 28.90$. In both cases, nearly all answers were collected within 25 minutes, and no one answered after 35 minutes. The second scenario clearly indicates that better quality tests have a strong impact on both cost and quality.

Figures 6(a)-6(c) ("real crowd" label) show the results achieved by T1−on and incr when questions are asked to the Crowdflower crowd in the second scenario. The T1−on algorithm clearly achieves the best quality also in this case, with slightly higher CPU times (Figure 6(b)). However, the overall time incurred by T1−on is extremely high (due to TPO materialization, not included in Figure 6(b), averaging

2. https://crowdflower.com



(a) "YouTube", $N = 78$, $K = 10$ (b) "YouTube", $N = 78$, $K = 10$ (CPU time) (c) "YouTube", $N = 78$, $K = 10$ (crowdsourcing time) (d) "Image quality", $N = 15$, $K = 3$

Fig. 6: Distance to solution achieved on real datasets, when B varies (T1−on and incr)

3 hours and 14 minutes), while incr is acceptably fast also when the budget is large. Note that times are much higher with the real crowd also because we cannot trust their answers, and thus do not prune the TPO (see Section 6.5). Figure 6(c) shows the crowdsourcing time needed to perform tasks on Crowdflower. Note that T1−on requires a larger crowdsourcing time (on average, 86 seconds for each question) with respect to incr (on average, 239 seconds for each batch of 5 questions), since batches containing a single question are more time-consuming for workers.

7 Related Work

Many works in the crowdsourcing area have studied how to exploit a crowd to obtain reliable results in uncertain scenarios. In [32], binary questions are used to label nodes in a directed acyclic graph, showing that an accurate question selection improves upon a random one. Similarly, [33] and [28] aim to reduce the time and budget used for labeling objects in a set by means of an appropriate question selection. Instead, [11] proposes an online question selection approach for finding the next most convenient question so as to identify the highest ranked object in a set. A query language where questions are asked to humans and algorithms is described in [34]; humans are assumed to always answer correctly, and thus each question is asked once. All these works do not apply to a top-K setting and cannot be directly compared to our work.

Uncertainty in top-K queries.

Uncertainty representation. The problem of ranking tuples in the presence of uncertainty has been addressed in several works [42], [26], [25]. As discussed in Section 3, we based our techniques for the construction of a TPO on these works.

Uncertain top-K queries on probabilistic databases. In [30], the quality score for an uncertain top-K query on a probabilistic (i.e., uncertain) database is computed. Moreover, the authors address the problem of cleaning uncertainty to improve the quality of the query answer, by collecting multiple times data from the real world (under budget constraints), so as to confirm or refute what is stated in the database.

Crowdsourcing via tuples comparison. We now discuss recent works on uncertain top-K scenarios where questions comparing tuples in a set are asked to a crowd. In [8], the authors consider a crowd of noisy workers and tuples whose scores are totally uncertain. This approach does not lend itself well to our scenarios, where prior knowledge on the score pdf's is assumed: for instance, when $N = 1000$, $\delta = 0.001$ and workers answer correctly with probability 0.8, their approach would require 999 questions to determine the top-1 tuple, while 2.7 are in average sufficient with our T1−on. The work in [29] proposes a query interface that can be used to post tasks to a crowdsourcing platform such as Amazon MTurk. When addressing a top-K query, their method first disambiguates the order of *all* the tuples by asking questions to the crowd, and then extracts the top-K items. This amounts to asking many questions that are irrelevant for the top-K prefix, since they could involve tuples that are ranked in lower positions. The wasted effort grows exponentially as the dataset cardinality grows. Instead, our work only considers questions that involve tuples comprised in the first K levels of the tree. A more recent work in [35] builds the top-K (top-1 in [43]) list by asking workers to sort small sets of s tuples whose scores are, again, totally uncertain. The top-K tuples are determined via a voting mechanism that refines the set of top-K candidates after each "roundtrip" of tasks, until only K tuples are left. Although when $s = 2$ the tasks are a comparison of two tuples like in our approach, their question selection is completely agnostic of any prior knowledge on the tuples, thus resulting in a much higher overall amount of questions in scenarios like those considered in this paper.

Uncertain top-K sets. In [7] the authors propose procedures for the extraction of k objects that have a specified property. The proposed algorithms extract sets of n objects that are analyzed in parallel by humans. At each round, n tasks are submitted to the crowd, and the objects that are recognized as relevant (i.e., objects having the specified property) are retrieved. Rounds are continuously created, until exactly K objects are retrieved. The work considers both the cases of oracles and of noisy workers. However, this work only takes care of extracting a set of objects, which remain unordered, with no guarantee

to include the top K objects.

Crowdsourcing via other task types. In [2] the authors assume that the ordering of a set of objects is known, and use a crowdsourcing-based algorithm to estimate their score values. In [19] crowdsourcing is used to build a tree where the root represents an initial status, leaves represent a fixed objective and each path represents a sequence of actions to be performed so as to meet the objective. Workers are provided with a subpath and are required to suggest the next action in the sequence to be performed. The goal of the proposed algorithm is to retrieve the best K paths from the tree.

Uncertainty in schema and object matching.

Schema matching. In [52], uncertainty in schema matching is tackled by posing questions to workers. Uncertainty is measured via entropy, and two algorithms (online and offline) are proposed to select the questions reducing uncertainty the most. A similar approach is proposed in [9] for the context of web tables schema matching, although only an online scenario is considered in this case. We have shown that, in top-K contexts, the results obtained by measuring uncertainty via entropy are largely outperformed by the use of other criteria (e.g., U_{MPO}). Noisy workers are used to validate schema matchings also in [14], with emphasis on the design of questions, so as to maximize their informativeness and reduce the noise in validations. Yet, [14] does not present any question selection strategy, which we have shown to be a useful means to obtain good results even with a noisy crowd and simple boolean questions.

Object matching. There are several noteworthy works about object matching. In [46], the objective is to identify all pairs of matching objects between two collections of objects. The authors propose a mixed online and offline approach, where the selected sequence of questions is annotated partially by machines and partially by users, and minimizes the number of questions answered by humans. This work was recently extended by the authors of [45], who propose two alternative algorithms for entity resolution. In [49], the B most promising questions are asked to workers so as to enhance Entity Resolution.

Workers' accuracy estimation.

Several works in the state of the art ([10], [7], [8], [22]) use majority voting as a tool for aggregating multiple noisy answers and computing trusted labels. In other cases ([13], [38], [27]) workers are pre-filtered via qualification tests, so that low-quality workers will not access the submitted tasks. Experts may be used to validate uncertain answers [31]. Other works [44], [15], [18], [6], [37] in crowd-related research propose ways to estimate workers' accuracy: it may be computed depending on the number of disagreements with other worker answers (i.e., the larger the number of disagreements, the larger the error probability), or by modeling the behavior of high quality workers versus spammers. In [52], the error probability of the user is supposed to be known, and accordingly the user's answer is considered less relevant as the error probability grows. Finally, [3] uses an approach that mixes test questions to filter out spammers, majority voting to improve the accuracy of single workers and estimation of probability error based on task difficulty.

8 Conclusions and Future Work

In this paper we have introduced Uncertainty Resolution (UR), which is the problem of identifying the minimal set of questions to be submitted to a crowd in order to reduce the uncertainty in the ordering of top-K query results. First of all, we proved that measures of uncertainty that take into account the structure of the tree in addition to ordering probabilities (i.e., U_{MPO}, U_{H_w} and U_{ORA}) achieve better performance than state-of-the-art measures (i.e., U_H). Moreover, since UR does not admit deterministic optimal algorithms, we have introduced two families of heuristics (offline and online, plus a hybrid thereof) capable of reducing the expected residual uncertainty of the result set. The proposed algorithms have been evaluated experimentally on both synthetic and real data sets, against baselines that select questions either randomly or focusing on tuples with an ambiguous order. The experiments show that offline and online best-first search algorithms achieve the best performance, but are computationally impractical. Conversely, the `T1−on` and `C−off` algorithms offer a good tradeoff between costs and performance. With synthetic datasets, both the `T1−on` and `C−off` achieve significant reductions of the number of questions wrt. the `Naive` algorithm. The proposed algorithms have been shown to work also with non-uniform tuple score distributions and with noisy crowds. Much lower CPU times are possible with the `incr` algorithm, with slightly lower quality (which makes `incr` suited for large, highly uncertain datasets). These trends are further validated on the real datasets. Future work will focus on generalizing the UR problem and heuristics to other uncertain data and queries, for example in skill-based expert search, where queries are desired skills and results contain sequences of people sorted based on their topical expertise and skills can be endorsed by community peers.

References

[1] M. Allahbakhsh et al. Quality control in crowdsourcing systems: Issues and directions. *IEEE Internet Comp.*, 17(2):76–81, 2013.

[2] A. Amarilli et al. Uncertainty in crowd data sourcing under structural constraints. In *DASFAA*, pages 351–359, 2014.

[3] A. Anagnostopoulos et al. The importance of being expert: Efficient max-finding in crowdsourcing. In *SIGMOD*, 2015.

[4] M. Cha et al. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Trans. Netw.*, 17(5):1357–1370, 2009.

[5] R. Cheng et al. Efficient join processing over uncertain data. In *15th ACM international conference on Information and knowledge management*, pages 738–747. ACM, 2006.

[6] N. N. Dalvi et al. Aggregating crowdsourced binary ratings. In *WWW*, pages 285–294, 2013.

[7] A. Das Sarma et al. Crowd-powered find algorithms. In *ICDE*, pages 964–975. IEEE, 2014.

[8] S. B. Davidson et al. Top-k and clustering with noisy comparisons. *ACM Trans. Database Syst.*, 39(4):35:1–35:39, 2014.

[9] J. Fan et al. A hybrid machine-crowdsourcing system for matching web tables. *ICDE*, 2014.

[10] C. Gokhale et al. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.

[11] S. Guo et al. So who won?: Dynamic max discovery with the crowd. In *SIGMOD '12*, pages 385–396, 2012.

[12] P. Hart et al. A formal basis for the heuristic determination of minimum cost paths. *IEEE TSSC*, 4(2):100–107, 1968.

[13] F. C. Heilbron and J. C. Niebles. Collecting and annotating human activities in web videos. In *ICMR*, page 377, 2014.

[14] N. Hung et al. On leveraging crowdsourcing techniques for schema matching networks. In *DASFAA*, LNCS 7826, pages 139–154. 2013.

[15] P. G. Ipeirotis et al. Quality management on amazon mechanical turk. In *SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.

[16] P. G. Ipeirotis and E. Gabrilovich. Quizz: Targeted crowdsourcing with a billion (potential) users. WWW '14, pages 143–154.

[17] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *TOIS*, 20(4):422–446, 2002.

[18] M. Joglekar et al. Comprehensive and reliable crowd assessment algorithms. *ICDE*, 2015.

[19] H. Kaplan, I. Lotosh, T. Milo, and S. Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, 2013.

[20] D. Kempe et al. Maximizing the spread of influence through a social network. KDD '03, pages 137–146. ACM.

[21] L. S. Kennedy and M. Naaman. Generating diverse and representative image search results for landmarks. In *WWW*, pages 297–306. ACM, 2008.

[22] S. K. Kondreddi et al. Combining information extraction and human computing for crowdsourced knowledge acquisition. In *ICDE*, pages 988–999, 2014.

[23] R. Kumar and S. Vassilvitskii. Generalized distances between rankings. In *WWW '10*, pages 571–580, 2010.

[24] P. Le Callet and F. Autrusseau. Subjective quality assessment IRCCyN/IVC database, 2005. http://www.irccyn.ec-nantes.fr/ivcdb/.

[25] J. Li and A. Deshpande. Ranking continuous probabilistic datasets. *PVLDB*, 3(1-2):638–649, 2010.

[26] J. Li et al. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[27] X. Liu et al. CDAS: A crowdsourcing data analytics system. *PVLDB*, 5(10):1040–1051, 2012.

[28] A. Marcus et al. Crowdsourced databases: Query processing with people. In *CIDR '11*, pages 211–214, 2011.

[29] A. Marcus et al. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, Sept. 2011.

[30] L. Mo et al. Cleaning uncertain data for top-k queries. In *ICDE, 2013*, pages 134–145. IEEE, 2013.

[31] Q. V. H. Nguyen et al. Minimizing Efforts in Validating Crowd Answers. In *The 2015 ACM SIGMOD/PODS Conference*, 2015.

[32] A. Parameswaran et al. Human-assisted graph search: It's okay to ask questions. *PVLDB*, 4(5):267–278, 2011.

[33] A. Parameswaran et al. Crowdscreen: Algorithms for filtering data with humans. In *SIGMOD '12*, pages 361–372, 2012.

[34] A. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR '11*.

[35] V. Polychronopoulos et al. Human-powered top-k lists. In *WebDB*, pages 25–30, 2013.

[36] B. Qin et al. A rule-based classification algorithm for uncertain data. In *ICDE*, pages 1633–1640. IEEE, 2009.

[37] V. C. Raykar and S. Yu. Ranking annotators for crowdsourced labeling tasks. In *NIPS*, pages 1809–1817, 2011.

[38] J. Redi and I. Povoa. Crowdsourcing for rating image aesthetic appeal: Better a paid or a volunteer crowd? In *ACM MM*, CrowdMM '14, pages 25–30. ACM, 2014.

[39] H. R. Sheikh et al. A statistical evaluation of recent full reference image quality assessment algorithms. *Image Processing, IEEE Trans. on*, 15(11):3440–3451, 2006.

[40] V. S. Sheng et al. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD '08*, pages 614–622, 2008.

[41] M. Soliman et al. Ranking with uncertain scoring functions: semantics and sensitivity measures. In *SIGMOD '11*, pages 805–816, 2011.

[42] M. Soliman and I. Ilyas. Ranking with uncertain scores. In *ICDE '09.*, pages 317 –328, 2009.

[43] P. Venetis et al. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

[44] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. In *International Workshop on Crowdsourcing and Data Mining*, pages 15–21. ACM, 2012.

[45] N. Vesdapunt et al. Crowdsourcing algorithms for entity resolution. *VLDB*, 7(12), 2014.

[46] J. Wang et al. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 international conference on Management of data*, pages 229–240. ACM, 2013.

[47] Y. Wang et al. A survey of queries over uncertain data. *Knowledge and information systems*, 37(3):485–530, 2013.

[48] Z. Wang et al. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[49] S. E. Whang et al. Question selection for crowd entity resolution. *VLDB*, 2013.

[50] H. Yu et al. Enabling ad-hoc ranking for data retrieval. *ICDE*, 2005.

[51] M.-C. Yuen et al. A survey of crowdsourcing systems. In *SocialCom*, pages 766–773. IEEE, 2011.

[52] C. J. Zhang et al. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, July 2013.

[53] W. Zhang et al. A trust based framework for secure data aggregation in wireless sensor networks. In *SECON '06*, pages 60–69, 2006.

Eleonora Ciceri , Ph.D., is a post-doc researcher at Politecnico di Milano, Italy. Her research interests mainly include query processing and crowdsourcing.

Piero Fraternali is full professor of Web Technologies at DEIB, Politecnico di Milano, Italy. His main research interests concern software engineering, and methodologies, tools for Web application development, multimedia information retrieval and human computation.

Davide Martinenghi, Ph.D., is an associate professor at Politecnico di Milano, Italy. His areas of expertise include ranking queries, data integrity maintenance, knowledge representation, and query optimization aspects related to web data access and web search.

Marco Tagliasacchi is currently Associate Professor at DEIB, Politecnico di Milano, Italy. He received the MS degree (2002) in Computer Engineering and the Ph.D. in Electrical Engineering and Computer Science (2006), both from Politecnico di Milano. His research interests include multimedia forensics, multimedia communications (visual sensor networks, coding, quality assessment) and information retrieval.