# TopCrowd –

## Efficient Crowd-enabled Top-k Retrieval on Incomplete Data

Christian Nieke[1], Ulrich Güntzer[2], Wolf-Tilo Balke[1]

[1] IFIS, TU Braunschweig, Braunschweig, Germany
{ nieke, balke}@ifis.cs.tu-bs.de
[2] Inst. f. Informatik, Universität Tübingen,Tübingen, Germany
ulrich.guentzer@informatik.uni-tuebingen.de

**Abstract.** Building databases and information systems over data extracted from heterogeneous sources like the Web poses a severe challenge: most data is incomplete and thus difficult to process in structured queries. This is especially true for sophisticated query techniques like Top-k querying where rankings are aggregated over several sources. The intelligent combination of efficient data processing algorithms with crowdsourced database operators promises to alleviate the situation. Yet the scalability of such combined processing is doubtful. We present TopCrowd, a novel crowd-enabled Top-k query processing algorithm that works effectively on incomplete data, while tightly controlling query processing costs in terms of response time and money spent for crowdsourcing. TopCrowd features probabilistic pruning rules for drastically reduced numbers of crowd accesses (up to 95%), while effectively balancing querying costs and result correctness. Extensive experiments show the benefit of our technique.

**Keywords:** Query processing, top-k queries, crowdsourcing, incomplete data.

## 1 Introduction

Bringing together peoples' cognitive abilities and efficient information processing algorithms is one of the hottest topics in database and information systems research [1–3]. The basic idea is simple, yet intriguing: let both, machines and people, do what they do best and combine their efforts! Machines and algorithms are perfect for fast and structured processing tasks over huge amounts of data, whereas humans are hard to beat in cognitive tasks or whenever flexible and intelligent decisions are needed. This combination is especially efficient where the classical structured databases are complemented by information derived from large collections of (semi-structured) data extracted in a 'schema later'-fashion from the Web. In particular, today's Web information management ties real world entities like consumer products, persons, or news items/media to additional information, thus boosting user experience.

Indeed, the usefulness of crowdsourced operators for query processing in databases has already been shown for tasks like the on-demand completion of missing values (CNULL values [2]), human-powered sorts and joins [4], human-guided similarity

ER, p. 1, 2014.

search [5], or entity linking and reconciliation [6]. However, the central question of query expressivity vs. processing costs remains. In the database world data accesses have to be minimized and the classical storage hierarchy is exploited for scalability. Now, crowdsourcing human intelligence tasks (HITs) adds another very expensive layer. Deploying HITs costs money and their completion via platforms like Amazon's Mechanical Turk (https://www.mturk.com), CrowdFlower (http://crowdflower.com/), or Samasource (http://samasource.org/) needs considerable time. Therefore, a careful balancing of a human intervention's usefulness with respective costs is necessary for all crowd-enabled query processing. *This paper focuses on exactly this problem: we introduce a new algorithm for crowd-enabled Top-k retrieval on incomplete data*.

Top-k retrieval, i.e. the aggregated ranking of database items according to user-provided preference functions, features a variety of efficient algorithms tailored to different scenarios. For example, a real world application of Top-k retrieval is support for purchases like buying a car or laptops for a company (which we will use as a running example in this paper). However, classical approaches to Top-k retrieval assume complete knowledge of all attributes for all objects in the database, which is not true in case of datasets aggregated from different sources like Open Linked Data or Web portals presenting offers from different vendors. For example, the average battery runtime of a laptop might be an important factor in decision making, but this piece of information might not be included in every vendor's description. In such scenarios, easily and reliably getting the best offers from an incomplete data set is well worth waiting a bit for the result and even investing a few dollars for crowdsourcing.

Looking at the challenges it is clear that a crowd-enabled Top-k algorithm should derive a correct and complete ranking at minimum cost in terms of response time and money spent. However, there is also some space for trade-offs: since crowd accesses are really expensive, can some of the result correctness be traded for better cost effectiveness? This question is valid, since Web sources generally do not claim completeness. They just collect all reasonably trustworthy pieces of information extracted from the Web (cf. DBPedia [7], Freebase [8], or the YAGO knowledge base [9]). Thus, our approach also has practical impact on Web information systemengineering.

We present TopCrowd an innovative algorithm for crowd-enabled Top-k retrieval. In particular we investigate safe Top-k pruning rules, tuple selection heuristics for crowdsourcing, and probabilistic result set correctness. Thus, TopCrowd is able to balance crowdsourcing costs in terms of money and response times, HIT selection and batching, and result correctness. Our main contributions are:

- A sophisticated algorithmic framework for Top-k query processing over databases with missing values utilizing crowdsourcing techniques.
- Safe pruning rules for correct crowd-enabled Top-k retrieval enhanced by probabilistic rules for drastically reduced numbers of crowd accesses, and thus costs.
- Serious performance gains by an order of magnitude depending on the retrieval scenario
- Good prediction of either hard upper bound for costs (given a probability of result quality) or hard lower bound for probability of result quality (given a cost limit).
- Very good batch size optimization balancing financial cost and runtime.

## 2 Related Work

Due to the Web as a new information source Top-k retrieval in databases was investigated starting about a decade ago and now has reached a mature state. While the first applications focused on multimedia databases [10] and middleware solutions [11], it soon became clear that Top-k retrieval is important for a wide variety of scenarios like Web databases[12], mobile scenarios[13], or distributed retrieval settings [14].

Basically, for a database instance of N objects the model for Top-k queries on D attributes is a D-dimensional space usually restricted to $[0,1]^D$. Using a user-provided *utility* or *preference function*, numerical as well as categorical attributes can individually be transformed for each attribute into a total order [15]. Moreover, users provide a *monotonic scoring function* $totalscore: [0,1]^D \rightarrow [0,1]$ that allows the aggregation of individual attribute scores into a final score for subsequent ranking. Top-k algorithms usually distinguish two types of access: *sorted* and *random* access. While sorted accesses iterate over the objects in any list for some attribute in descending score order, random accesses directly retrieve some object's score value with respect to any attribute. Because the costs of these access types may strongly differ (usually random accesses are much more expensive than sorted accesses), algorithms that flexibly adapt to the respective usage scenario have been developed (see e.g. [13]).

Incomplete data sets create new problems: they contain CNULL values [2], which usually have a well-defined value (e.g. "5 hours" for some laptop's average battery runtime) that is however yet unknown to the system. In the presence of these CNULL values, conventional Top-k database algorithms will not work, as it is impossible to correctly rank objects with unknown attribute values with either kind of access (cf. failing access heuristics in [11, 13, 16]). We therefore need a way to either reliably estimate missing scores (see [17] and [18] for ranking in incomplete databases) or elicit the exact value using a new kind of access: the *crowd access*. The *crowd access* crowdsources the cognitive task of procuring adequate attribute values by e.g. performing Web searches to find the required information on a laptop in a vendor's specification or by calling the vendor of a used car and asking for additional details.

Of course, this leads to the question of the *quality* of information gathered by the crowd. Here typical safeguards can be used like blending in gold questions with previously established answers to detect malicious or incompetent users or performing majority votes over several users working on the same task (see. e.g. [2, 19]). Still, depending on the task there may be differing quality levels. For instance, for purely factual tasks like looking up movie genres on IMDB.com, experiments in [19] show a correctness of 95% at only 0.03 USD per tuple including gold questions. Similar results have been shown for labeling training data in the IR community. Hence, for simple attribute look-ups in Top-k query processing safeguarding with gold questions and banning malicious or incompetent workers works fine (see section 4.2).

It is obvious that due to the relatively high costs (especially in terms of response times ranging in the area of minutes for a crowdsourced HIT), the number of *crowd accesses* should be limited as much as possible. Especially for larger Web datasets, crowdsourcing every incomplete object up front to perform a classical Top-k retrieval is prohibitive. Thus, the most related work to our approach is [20] where approximate

variants of the basic threshold algorithm to reduce run-time costs are presented. Top-k algorithms are basically considered as linear index scans over the descending score lists for each attribute. Based on probabilistic arguments, a point can be determined where it is safe to drop candidate items and to terminate the index scans. However, these approximation algorithms cannot cope with incomplete data, nor do they balance different access types or perform probabilistic selections of candidates for more complex score elicitation. Still, the probabilistic estimation of the final error in result correctness can to some degree be used in our scenario.

Another related problem is discussed in e.g. [21–23], which deal with Top-k requests over objects that need a crowd operator for comparison (e.g. comparing pictures). Here, the challenge is to find the Top-k objects without having to perform the (naïve) $O(n^2)$ comparisons of each object with each other, while we deal with the problem of reducing the O(n) crowd lookups of missing attributes, whereas the order can be found algorithmically.

# 3 The TopCrowd Algorithm

In the following we will present our algorithm TopCrowd which allows performing Top-k retrieval on incomplete data while optimizing the cost of *crowd accesses* necessary to retrieve missing data.

## 3.1 Formal Definition

Given a D-dimensional dataset containing N objects, we will denote objects $o_i, 1 \leq i \leq N$ as tuples of attributes $o_i[1] \dots o_i[D]$ as $o_i \in ([0,1] \cup \{CNULL\})^D$.

Every attribute is assigned a score in [0,1] or it is considered missing, but could be retrieved via crowdsourcing (CNULL). We further denote the set of all items $A$ and define the set of incomplete items $I$ as: $I := \{ o_i \in A \mid \exists_{1 \leq j \leq D}\, o_i[j] = CNULL \}$ and the set of complete items $C$ as: $C := \{ o_i \in A \mid \nexists_{1 \leq j \leq D}\, o_i[j] = CNULL \}$.

For each complete object in C we can calculate its total score, representing how well it corresponds to a given query using any *monotonic* function $score : C \rightarrow [0,1]$. To avoid confusion, in the following we will refer to the score attributes simply as 'attributes' and 'score' always refers to the total score.

## 3.2 Basic Algorithm

The main optimization objective of our algorithm is to reduce the number of expensive crowd accesses. To achieve this, our algorithm performs the following 4 basic steps which will be discussed in detail in the following subsections:

1. Classic Top-k retrieval on complete objects
2. Optimal safe pruning of incomplete objects
3. Probabilistic ranking of incomplete objects
4. Crowd access cost control

In the first step we perform classical Top-k retrieval on all those objects that are completely known, resulting in a temporary Top-k result $K_c$ and the remaining incomplete objects $I$ that could not be handled by classical Top-k.

In the next step, we perform optimal safe pruning, i.e. we discard *exactly* those incomplete objects that could never reach the Top-k, independent of the actual value of all their CNULL values, to avoid all strictly unnecessary *crowd accesses*.

We then estimate the probability of all remaining incomplete objects to reach the Top-k after crowdsourcing, and rank them accordingly, so that the most promising ones will be crowdsourced first, which could allow further pruning.

We could now start crowdsourcing the incomplete objects one by one according to their rank and return to the pruning step after receiving any new result until no incomplete objects are left, which would minimize the number of crowd accesses to reach the *correct* result in an optimal way, assuming a correct rank estimation and correct results from the crowd accesses. However, in practice the user might have an interest in balancing financial costs (which depend on the number of crowd accesses) and processing time and might be willing to sacrifice some of the result quality to improve those costs.

We therefore added a fourth step that allows the user to control these costs by either letting her define an amount of money she is willing to spend and presenting her with a hard lower bound probability that the result will be correct (*P_target*), or letting her define the target probability she requires, and giving a hard upper bound for financial costs, while always trying to optimize processing time.

In the following, we will explain the single steps of the algorithm in more detail.

### 3.3 Optimal Safe Pruning

After receiving the temporary result $K_c$ from the initial Top-k retrieval on the complete objects in the first step, or a previous iteration, we define the score of the worst object in this set as the minimal score, *min_topk*, which an incomplete object must at least surpass to replace any object in the temporary Top-k list.

Given an *upper bound* for each incomplete object as the score after replacing missing attributes with 1, we can then prune all incomplete objects whose upper bound is below or equal *min_topk*, as they will never replace any item in $K_c$. This makes the algorithm *correct*, as it prunes *only* items that cannot improve the Top-k result, but also *optimal*, as it prunes *all* items that, given our current knowledge, will never improve the Top-k. Objects with an upper bound of exactly *min_topk* will be pruned by our algorithm, as they might change, but not *improve* the result. □

### 3.4 Ranking of Incomplete Objects

When crowdsourcing batches of incomplete objects, one should obviously start with the candidates that are most likely to actually be part of the Top-k, especially when following a probabilistic approach. In the following, we will first present two approaches used as lower and upper baseline during our evaluation, followed by a more sophisticated approach to estimate the ranking.

**Lower Baseline: Naïve UpperLowerRatio.** For our lower baseline, we developed the following, very intuitive way to predict if an incomplete object will be in the Top-k, by incorporating only its upper and lower bound score, where the lower bound is defined as the score of an incomplete object, with all missing attributes set to 0.

Assuming no knowledge of the scoring function and the data distribution, we assume a uniform distribution of scores, and define the probability of an incomplete object $I_i \in I$ to get into the current Top-k result set $K_c$ as the probability of the objects score to reach above *min_topk*. Given its upper and lower bound, the conditional probability can therefore be defined as:

$$P(score(I_i) > min\_topk) = \frac{upperBound(I_i) - min\_topk}{upperBound(I_i) - lowerBound(I_i)}$$

Given these probabilities for each incomplete object, we can now select the most probable objects for crowdsourcing. In the following, we will refer to this ranking strategy as *UpperLowerRatio*. While this approach is very fast and intuitive, the underlying assumption of a uniform distribution of scores is of course very likely to be wrong, a problem that is addressed by our more sophisticated prediction approach presented later.

**Upper Baseline: Perfect Ranking**. For evaluation purposes, we define the upper baseline of what a prediction could possibly achieve as *perfect ranking*, i.e. a strategy that selects the objects ordered descending by their actual score, which is of course only known beforehand in our experimental setting, and not for real applications.

**Probabilistic Prediction and Projection: KDEScorePrediction.** This approach tries to predict the score probability distribution of an incomplete object, using the actual distribution of the complete objects.

A common approach for predicting the distribution of objects in a multidimensional space, given a sample of the distribution, is multivariate kernel density estimation (KDE), see e.g.,[24]. The KDE estimates a density function φ as:

$$\varphi_H(x) = \frac{1}{n}\sum_{i=1}^{n} K_H(x - x_i)$$

Here the $x_i$ are the n sample vectors in the space, and $K_H$ is a kernel function with a given bandwidth H for smoothing. In a nutshell, the idea of a KDE is to create a normalized histogram of the space and smooth it using the kernel.

For each incomplete item o with *m* missing attributes, an *m*-dimensional flat of that space $F_o^m$ can be defined as:

$$F_o^m := \{ (f[1], \ldots, f[D]) \mid \forall_{1 \le i \le D}: \begin{cases} f[i] = o[i], o[i] \neq CNULL \\ f[i] \in [0,1], \quad else \end{cases} \}$$

$F_o^m$ represents the sub area of the whole D-dimensional space, in which the real coordinates of *o* will be after crowdsourcing. This allows to calculate the probability

of an incomplete object to be above *min_topk* by calculating the m-dimensional integral over the density in the area of $F_o^m$ that would yield a score above *min_topk*, divided by the integral over the density in all of $F_o^m$.

Unfortunately, the complexity of calculating the discrete integral in a high dimensional space is exponential in terms of the number of dimensions. Assuming we used 100 steps per dimension to sample the space, we would have to calculate the density for $100^m$ points, which becomes computationally infeasible even for moderate *m*. Even though there are some computationally more efficient approaches of sampling, using Monte Carlo techniques or sparse grids (see e.g., [25]and [26]), they are still far from performing in real-time and our assumption of considering local computation as negligible would no longer hold. To avoid this problem (related to the *curse of dimensionality*) we propose the following approach, using a *localized sample* and the *projection* of multidimensional objects to a single score axis.

To understand our prediction method, it is essential to understand, that we are not actually interested in the *exact multidimensional coordinates* of the data, but only in their resulting *aggregated scores*. Thus, rather than predicting the distribution of coordinates in $F_o^m$, we will project all complete items in $F_o^m$ to a one dimensional score axis reflecting only the objects' aggregated scores, and use their distribution to predict the score of the respective incomplete object.

To do this, we define a set of *support points* of the incomplete o, $SP_o$ as the set of all complete items in $F_o^m$ as: $SP_o := C \cap F_o^m$. As $F_o^m$ will most likely be sparsely populated in higher dimensions, we will increase its size by a parameter +/-Δ along all known attributes of o, which we will call $F_{o,\Delta}^m$. This allows us to define the (larger or equal) set $SP_{o,\Delta}$ as: $SP_{o,\Delta} := C \cap F_{o,\Delta}^m$.

We can now use the scoring function to project the *support points* onto the one dimensional score axis, to receive a set of *support scores* $SS_{o,\Delta}$ as: $SS_{o,\Delta} := \{ss \in [0,1] \mid \exists sp \in SP_{o,\Delta}: ss = score(sp)\}$.

Using these support scores as samples for a univariate KDE, we are able to create a score density function of o, $\varphi_o(score)$ which we finally use to predict the conditional probability of the incomplete object o to be above *min_topk* given its upper bound and lower bound as:

$$P(score(o) > min\_topk) = \frac{\int_{x=min\_topk}^{upperBound(o)} \varphi_o(x)}{\int_{x=lowerBound(o)}^{upperBound(o)} \varphi_o(x)}$$

Using this approach, rather than sampling a high dimensional space, we only need to perform a ranged query to retrieve the support points and then sample the one dimensional score density to calculate the discrete integral. During our experiments, we found that this calculation could be performed for all incomplete objects within a manner of seconds, and is thus negligible compared to crowdsourcing.

Note, that the choice of Δ can of course affect the performance of this approach. If chosen to small, there will not be enough *support points* for a good prediction, but if chosen to large, the prediction will lose its specificity for the incomplete object. This intuition was supported in a set of experiments, but for reasons of brevity we will simply set it to 0.02, which worked well in our experiments. We will refer to this estimation strategy as *KDEScorePrediction*.

### 3.5 Controlling Crowdsourcing Costs

In the following we will show how to balance the three major concerns of a user, namely result quality, processing time and financial cost, by giving the user full control over a tradeoff between result quality and financial costs and optimizing processing time in exchange for a potential, minor financial overhead.

As safe pruning must unfortunately be based on the upper bound score of incomplete objects, this often leads to crowdsourcing a long tail of objects which were estimated to be very unlikely to reach the Top-k but cannot be pruned *safely*. However, a user might be willing to trade a certain amount of result quality for lower costs, by accepting a probabilistic approach that ignores some of the unlikely candidates for crowdsourcing, as long as the result is *likely enough*, i.e. it is correct with a user defined probability *P_target*.

Since our ranking strategies UpperLowerRatio and KDEScorePrediction already yield a probability for each item to be a above the current threshold, and assuming these probabilities are independent, we can use these probabilities to calculate the probability of all incomplete items not being above threshold, $P_{result}$, as:

$$P_{result} \coloneqq \prod_{o \in I}(1 - P(score(o) > min\_topk))$$

Using this formula, we can now predict the probability of a correct result and stop the algorithm when a user defined probability *P_target* is reached.

But while allowing a user to save financial costs in exchange for result quality is a good start, she needs a way to evaluate the trade-off between those two factors to make an informed decision. How much money would she save by reducing the probability from 95% to 90% and what would be the probability if she invested 3$?

Fortunately, we can make a prediction about this, using our probabilistic ranking strategies which define the rank *r* on which an item will be crowdsourced and its probability to reach the Top-k. The order of the incompletes is stable, assuming that a few crowdsourced objects do not influence the underlying probability distribution, and further crowdsourcing will only reduce the probability of an item becoming Top-k, by increasing the score needed to replace one of the temporary Top-k items. Using this information, we can a priori calculate a hard, lower bound probability that will be reached after any number *t* of ranked, incomplete objects $o_r$ is crowdsourced as:

$$P_{\text{result\_lowerbound}}(t) \coloneqq \prod_{r=t+1}^{R}\left(1 - P(score(o_r) > min\_topk)\right)$$

Here, $o_r$ is the object at rank *r* and R is the maximal rank of incomplete items. Using this formula, we can now predict the minimal probability that will be reached for a given number of crowd requests *t* (and thus financial costs) or inversely calculate the maximal number of crowd requests (and therefore financial costs) needed to achieve an intended probability.

This leaves us with a well-defined set of objects that we would have to crowdsource if our pessimistic bounds were true and leads us to the problem of runtime optimization. While performing all the remaining *crowd accesses* one by one, could actually reduce the number of crowd accesses below our pessimistic estimate

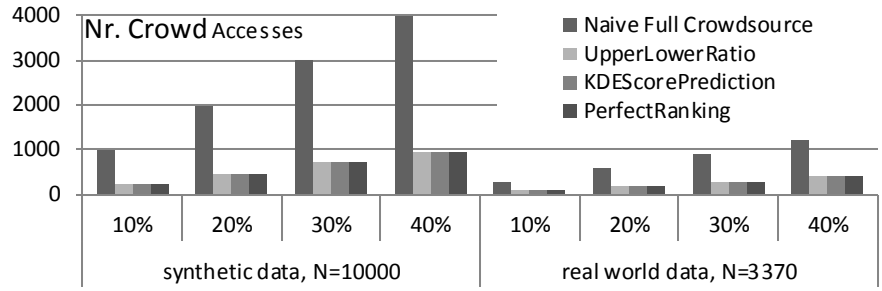


**Figure 1: TopCrowd for *P_target*=100% and different values of *missing_rate***

(by performing additional pruning whenever *min_topk* is increased), this is an extremely inefficient approach in praxis, as crowdsourcing jobs are best performed in large batches. For one, creating a batch with many HITs obviously allows a higher level of parallelization by dividing the tasks amongst several workers. Additionally, quality control usually tries to filter out incompetent or malicious workers by evaluating their agreement with other workers or by testing them using gold questions, which however creates an additional (also financial) overhead for each batch.

When we performed a set of experiments to evaluate the potential of performing small batches (see 4.2), we found that even when performing only one *crowd access* at a time we saved only between 1-2% of *crowd accesses*, and therefore decided to perform all requests in only one batch to optimize runtime. For similar reasons, we also decided to fetch all missing CNULL values of an object in one single *crowd access*, rather than splitting the attributes up into single tasks.

# 4 Evaluation

## 4.1 Datasets and Simulations

To test our algorithms under various conditions, we prepared a number of datasets consisting of both, real world data and synthetic data to allow larger experiments.

The real world data set consists of notebook offers crawled from linked data on the Web. We selected only suitable attributes to be used in Top-k requests. Since we needed a ground truth to assess our algorithms performance, we cleaned the data of all objects with actually missing attributes, and ended up with a set of 3370 notebooks, consisting of data for four dimensions: CPU frequency, hard drive size, memory size and screen size. All attribute values were scaled to an interval [0,1] and used as attribute scores.

For the synthetic data, we created a data set of 20 dimensions by creating pairs of attributes with attribute values in the interval [0,1]. Each pair follows either normal or uniform distribution and corresponds to a correlation factor of -0.8, -0.4, 0, 0.4 or 0.8 respectively. In total, this results in a dataset of over 10.000 objects with 20 dimensions of which *D* are picked randomly to represent either uniform or normal distribution and a certain chance that some of the dimensions will be correlated.

To test our algorithms in various conditions, we simulated several hundred different user queries per parameter set. A user query is characterized by a random selection of *D* attributes of the dataset combined via a scoring function representing the user's demands. Although our algorithm allows any monotonic scoring function, we used the arithmetic mean of all requested attributes for simplicity. We then randomly selected N objects from the data set and turned a given percentage of them into incomplete objects according to a *missing_rate*. Whenever an object was chosen to be turned into an incomplete object, each attribute had a chance of 50% to be deleted.

### 4.2 TopCrowd Evaluation

We will first evaluate the performance of the TopCrowd algorithm for Top-k query results with guaranteed correctness (*P_target* = 100%) using only safe pruning rules.

Figure 1 shows the results of our first experiment, in which we compared the performance of naïvely crowdsourcing all incomplete items to perform classical Top-k retrieval (Naïve Full Crowdsource) against our TopCrowd algorithm, using each of our three ranking strategies for P_target=100%. The result is shown for different values of *missing_rate*, which of course also defines the number of incomplete objects, and is averaged over at least 3000 samples with values of *D* between 2-4 and $k \in (10,20,40)$. Here, we performed all crowd accesses one by one, followed by an additional pruning after a new Top-k item was found, but as mentioned before, this increased performance by only about 1-2%, so we decided to perform all accesses in one batch in the future. The overall result, however, is promising, and shows a reduction of crowd accesses to about 24% of the original number for the large synthetic data set, and a reduction to 32-34% for the real world data, while the *missing_rate* shows no influence on relative performance. Even more, our TopCrowd algorithm shows no significant difference in performance between all ranking strategies, including the theoretical upper bound *perfect ranking* which delivers optimal results.

To check whether this effect was due to the selection strategies performing similarly well or if there was another effect overshadowing the differences, we compared our proposed selection strategies to *perfect ranking*. To this end, we performed 500 runs for each value of *k* and *missing_rate* in 4 dimensions, performed the initial pruning, and then calculated Spearman's rank correlation coefficient between the predicted ranking for each ranking strategy and the *perfect ranking*. On average, we got a correlation coefficient of 0.31 for *UpperLowerRatio* and 0.43 for *KDEScorePrediction* on synthetic data, and scores about 0.02 lower for the real data. These results show that both algorithms tend towards a correct prediction, and our proposed algorithm *KDEScorePrediction* performs significantly better than our lower baseline *UpperLowerRatio*, but still leaves room for improvement.

As the selection strategies showed clear differences in prediction quality, we further investigated the problem and found that a better prediction strategy does indeed reach the correct Top-k set earlier, but the algorithm still needs to continue crowdsourcing to eliminate all the remaining incomplete items, even though they are very unlikely, to reach guaranteed correctness.

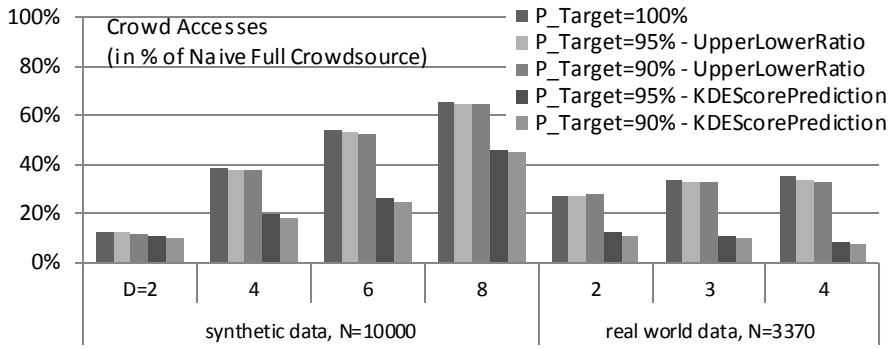


**Figure 2: TopCrowd for different values of *P_target* and *D***

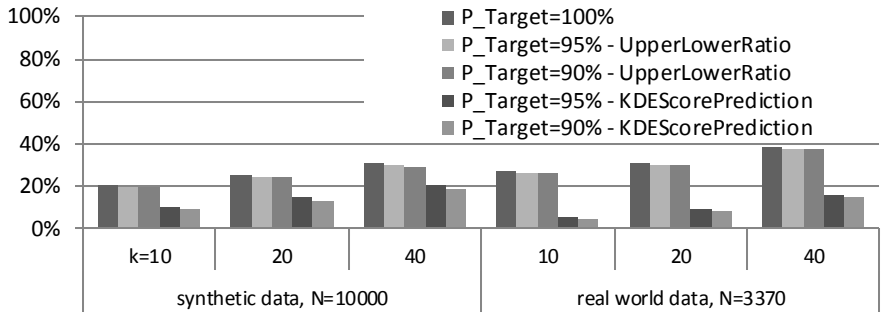


**Figure 3: TopCrowd for different values of *P_target* and *k***

This is exactly the problem which we tried to avoid with our probabilistic approach, so we performed another experiment comparing the ranking strategies for different values of *P_target* and *D*. Figure 2 shows the number of *crowd accesses* needed to achieve the given probability (in percentage of naïve crowdsourcing all incompletes), with each result averaged over at least 2400 samples with $missing\_rate \in (10\%, 20\%, 30\%, 40\%)$ and $k \in (10,20,40)$ while Figure 3 shows the same results for different values of *k* averaged over *D* from 2-4. The result clearly shows that the ranking strategy has a huge impact on our probabilistic approach, and while our lower baseline *UpperLowerRatio* improves performance by only about 1-2%, our strategy *KDEScorePrediction* allows to reduce the number of *crowd accesses* to 9-45% for synthetic data, and even more to 5-15% for our real world data set. While the performance for a higher number of dimensions decreases on the synthetic data set, it actually improves on the real world data set. We believe this effect is caused by the fact, that while in the synthetic data set most dimensions are actually unrelated, and therefore more dimensions simply increase the amount of information missing per object, the dimensions in the real world data set are related, which leads to a better prediction if it is based on more attributes (i.e. it is easier to predict A from B, C and D, rather than just from B).

When one uses a probabilistic approach to terminate the algorithm early, it is of course essential to evaluate if the prediction is indeed correct. To do this, we exam-

ined 16.000 simulations of TopCrowd with *KDEScorePrediction* for each target probability of 90% and 95% and counted the number of cases, where the result was actually correct when our algorithm terminated. We found that our algorithm actually found the correct result in 98.21% of all cases for a target probability of 90% and in 99.21% of all cases for a target probability of 95%, meaning that we are underestimating the probability of the result being correct. While these conservative estimations of our algorithm put us on the safe side when guaranteeing a lower bound probability, it also means that the intended probability was in fact reached earlier, meaning that a better prediction could lead to even better results.

In a final experiment, we performed the actual crowdsourcing operations for 10 runs of TopCrowd with 4 dimensions of the real dataset, a *missing_rate* of 20% and $k$=40. For quality control, we used gold questions and a majority vote of three workers per missing object. Per missing attribute we paid 0.01$ to look up the actual value, which we then transformed into the interval [0,1] the same way as for the original data. This led to a price of about 0.06$ per object including overhead for gold questions and fees to the portal *CrowdFlower*. On average, we paid 4$ for a probabilistic Top-k result with 90% probability and about 17.3$ for a correct result (probability 100%), while crowdsourcing *all* missing attributes would have cost about 40.44$ per experiment run. Our rudimentary safeguard mechanisms including gold questions and the majority vote of three crowd workers led to the correct result in 88% of all cases on average. The correctness could however likely be improved by employing some of the more advanced mechanisms as discussed before. In those cases where the result could not be reliably retrieved, we decided to exclude the object from the Top-k result, as we would rather risk a false negative than a false positive for Top-k scenarios.

## 5 Results and Discussion

Both, Web information management and retrieval, often face the problem of incomplete data, because information about central entities of interest may be distributed over several information sources. Hence, a new kind of query processing (including for instance data extraction, reference reconciliation, or entity ranking) is needed. As argued above for typical scenarios like product searches, individual recommendations, or decision support, this usually requires some intelligence to effectively perform the task. Fortunately the innovative combination of *efficient data processing algorithms* with new *crowdsourced database operators* promises to alleviate the situation.

In this paper we presented TopCrowd, a novel and efficient algorithm that shows the possibility of building sophisticated crowd-enabled query processing operators for Top-k query processing. Our extensive evaluations on synthetic, as well as real world Web data clearly show that the new TopCrowd operator can indeed be practical, although including human intervention. In particular, our *safe pruning rules* always deliver correct query results, while leading to immediate performance gains between 50-75% in terms of necessary crowd accesses, which indeed is valuable for such an expensive type of access. With individual response times and monetary query costs in mind, we then incorporated *probabilistic ranking strategies* allowing to sacrifice a bit

of the result correctness for vast performance improvements in a user-guided fashion. Building on the basic idea of kernel density estimation enhanced by score projection, our approach already allows a tight prediction of either upper bounds for costs (given a desired probability or result quality), or lower bounds for the probability of a correct result, given the costs a user is willing to accept for each query. In fact we showed that our algorithm allows to reduce crowdsourcing costs by up to 95% while at the same time optimizing batch size and therefore runtime. In fact, it can be guaranteed to at least reach some desired probability with just a single batch paying only for slightly more crowd accesses than strictly necessary in a hypothetical optimal algorithm. For future work there is still room for in-detail optimization and a tighter estimation of error bounds. Moreover, our future work will investigate other sophisticated query processing operators and their potential for intelligent crowdsourcing. We believe that it is necessary to harness the benefits of human cognition, assessment, and validation to intelligently steer data management and query processing tasks in a vast variety of applications with growing complexity. Thus, crowdsourced database operators are bound to gain more momentum in future query processing scenarios.

## 6 References


1. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the World-Wide Web. Communications of the ACM (CACM) (2011).
2. Franklin, M.J., Kossmann, D., Kraska, T., Ramesh, S., Xin, R.: CrowdDB. Proceedings of the 2011 International Conference on Management of Data (SIGMOD). ACM Press, New York, NY, USA (2011).
3. Parameswaran, A.: Answering Queries using Humans , Algorithms and Databases. Syst. Res. 160–166 (2011).
4. Marcus, A., Wu, E., Karger, D., Madden, S., Miller, R.: Human-powered Sorts and Joins. Proc. VLDB Endow. 5, 13–24 (2011).
5. Selke, J., Lofi, C., Balke, W.: Pushing the boundaries of crowd-enabled databases with query-driven schema expansion. Proc. VLDB Endow. (2012).
6. Demartini, G., Difallah, D.E., Cudré-Mauroux, P.: ZenCrowd. Proceedings of the 21st international conference on World Wide Web (WWW). ACM Press, New York, NY, USA (2012).
7. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - A crystallization point for the Web of Data. Web Semant. Sci. Serv. Agents World Wide Web. 7, 154–165 (2009).
8. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase. Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD). ACM Press, New York, NY, USA (2008).
9. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago. Proceedings of the 16th International Conference on World Wide Web (WWW). ACM Press, New York, NY, USA (2007).
10. Güntzer, U., Balke, W., Kießling, W.: Optimizing multi-feature queries for image databases. Proceedings of the 26th International Conference on Very Large Databases (VLDB). , Cairo, Egypt (2000).

11. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. 66, 614–656 (2003).
12. Marian, A., Bruno, N., Gravano, L.: Evaluating top- k queries over web-accessible databases. ACM Trans. Database Syst. 29, 319–362 (2004).
13. Balke, W., Güntzer, U., Kießling, W.: On Real-Time Top k Querying for Mobile Services. Procs. of Int. Conf. on Cooperative Information Systems (CoopIS). pp. 125–143. , Irvine, CA, USA (2002).
14. Balke, W.-T., Nejdl, W., Siberski, W., Thaden, U.: Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. 21st International Conference on Data Engineering (ICDE). pp. 174–185. IEEE, Tokyo, Japan (2005).
15. Fishburn, P.: Preference structures and their numerical representations. Theor. Comput. Sci. 217, 359–383 (1999).
16. Guntzer, J., Balke, W.-T., Kiessling, W.: Towards efficient multi-feature queries in heterogeneous environments. Proceedings International Conference on Information Technology: Coding and Computing. pp. 622–628. IEEE Comput. Soc.
17. Wolf, G., Khatri, H., Chokshi, B., Fan, J., Chen, Y., Kambhampati, S.: Query processing over incomplete autonomous databases. Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB). , Vienna, Austria (2007).
18. Hua, M., Pei, J., Lin, X.: Ranking queries on uncertain data. VLDB J. 20, 129–153 (2010).
19. Lofi, C., Selke, J., Balke, W.-T.: Information Extraction Meets Crowdsourcing: A Promising Couple. Datenbank-Spektrum. 12, 109–120 (2012).
20. Theobald, M., Weikum, G., Schenkel, R.: Top-k query evaluation with probabilistic guarantees. Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB). pp. 648–659. , Toronto, Canada (2004).
21. Alfaro, L. De, Davis, J., Garcia-Molina, H., Polyzotis, N.: Human-Powered Top-k Lists. International Workshop on the Web and Databases (WebDB). , New York, NY, USA (2013).
22. Davidson, S.B., Khanna, S., Milo, T., Roy, S.: Using the crowd for top-k and group-by queries. Proceedings of the 16th International Conference on Database Theory (ICDT). ACM Press, New York, New York, USA (2013).
23. Guo, S., Parameswaran, A., Garcia-Molina, H.: So who won?: dynamic max discovery with the crowd. Proceedings of the 2012 International Conference on Management of Data (SIGMOD). pp. 385–396 (2012).
24. Simonoff, J.S.: Smoothing Methods in Statistics. Springer (1996).
25. M. Jerrum, A.S. ed: The markov chain monte carlo method: an approach to approximate counting and integration. D. Hochbaum (ed.): Approximation Algorithms for NP-hard problems. PWS Publishing Company (1996).
26. Griebel, M., Schneider, M., Zenger, C.: A combination technique for the solution of sparse grid problems. In: De Groen, P. and Beauwens, R. (eds.) Iterative Methods in Linear Algebra. pp. 263–281. IMACS, Elsevier, North Holland (1992).