Skyline Queries in Crowd-Enabled Databases

Christoph Lofi
National Institute of Informatics
Tokyo 101-8430, Japan
lofi@nii.ac.jp

Kinda El Maarry
Institut für Informationssysteme
Technische Universität Braunschweig
Braunschweig, Germany
elmaarry@ifis.cs.tu-bs.de

Wolf-Tilo Balke
Institut für Informationssysteme
Technische Universität Braunschweig
Braunschweig, Germany
balke@ifis.cs.tu-bs.de

ABSTRACT

Skyline queries are a well-established technique for database query personalization and are widely acclaimed for their intuitive query formulation mechanisms. However, when operating on incomplete datasets, skylines queries are severely hampered and often have to resort to highly error-prone heuristics. Unfortunately, incomplete datasets are a frequent phenomenon, especially when datasets are generated automatically using various information extraction or information integration approaches. Here, the recent trend of crowd-enabled databases promises a powerful solution: during query execution, some database operators can be dynamically outsourced to human workers in exchange for monetary compensation, therefore enabling the elicitation of missing values during runtime. Unfortunately, this powerful feature heavily impacts query response times and (monetary) execution costs. In this paper, we present an innovative hybrid approach combining dynamic crowd-sourcing with heuristic techniques in order to overcome current limitations. We will show that by assessing the individual risk a tuple poses with respect to the overall result quality, crowd-sourcing efforts for eliciting missing values can be narrowly focused on only those tuples that may degenerate the expected quality most strongly. This leads to an algorithm for computing skyline sets on incomplete data with maximum result quality, while optimizing crowd-sourcing costs.

Categories and Subject Descriptors

H.2.4 **[Query Processing]**

General Terms

Algorithms, Human Factors

Keywords

Skyline Queries, Crowd-Sourcing, Incomplete Data.

1. INTRODUCTION

Crowd-Enabled DBMS [1] are currently a hot topic in the database community due to their promise of bridging the gap between somewhat inflexible yet efficient relational data management, and somewhat inefficient yet flexible human intelligence in retrieval tasks. Hence, such systems are particularly well-suited to deal with the challenges posed by modern applications. In particular, crowd-enabled DBMS allow for dealing with the widespread problem of *incomplete data* in an effective manner during runtime: queries may request missing values or even complete tuples by dynamically creating crowd-sourcing tasks eliciting the missing information from other sources. These tasks, generally called HITs[1], are then executed by human workers who are usually paid for their efforts.

Clearly this approach is a powerful tool, but each time a HIT has to be executed, the monetary costs and the runtime of a particular query increases, opening up a whole new field of relational query optimization [2]. While this problem is not too pronounced for traditional SQL-style queries (if one wants to retrieve for instance a person's email address, the task simply has to be crowd-sourced whenever it is missing), the problem may get arbitrarily difficult in case of aggregations or expensive predicates. Considering for example similarity operators, statistical aggregations, or ranking queries it is easy to see that the result set's correctness is largely dependent on the number and respective behaviour of missing values.

Focusing on skyline queries, in this paper we show how to manage this new optimization problem in crowd-enabled databases queries under the common issue of incomplete data. Skyline queries indeed form a perfect illustrative example: They are an important class for personalized query processing, but in contrast to simple factoid answers like e.g., email addresses, the skyline is a subset of database tuples formed by connections to other tuples following the notion of Pareto optimality. Of course, the query process can be approached in a naïve fashion: if any skyline algorithm is executed on a crowd-enabled database, the DB will automatically retrieve all missing values via crowd-sourcing during the runtime of the algorithm, retrieving the correct skyline set. However, this also leads to maximum costs in terms of money and runtime, since *all missing values* with respect to *all database tuples* need to be elicited from the crowd. In particular, the naïve retrieval does not take into account whether a tuple will be part of the final result set or not – which forms the optimization challenge: *maximum correctness at minimum costs*.

Therefore, in this paper, we will focus on cost optimizations for skyline queries in crowd-enabled databases. We show how to perform crowd-enabled skyline queries in a cost-efficient manner. In particular, we introduce a heuristic approach which will slightly compromise the expected result quality in order to significantly reduce query costs and runtimes. Our approach shows that not all missing values need to be crowd-sourced (in contrast to naïve

EDBT/ICDT '13, March 18 - 22 2013, Genoa, Italy.

[1] HIT: Human Intelligence Task, the smallest unit of crowd-sourceable work; many similar HITs are organized in HIT groups

querying), but instead we deal with the majority of the database tuples heuristically. This allows a significant reduction in the required number of crowd-sourcing tasks, but also introduces an (potentially significant) error into the computation of the resulting skyline set depending on the quality of the heuristic.

One of the more successful heuristic approaches for skyline computation on incomplete data is missing data prediction. In order to control and minimize the error introduced by this prediction, we introduce the notion of *prediction risk*, i.e. the risk quantified by the probable impact each tuple may have on the skyline correctness whenever a predicted value is used without crowdsourcing. By computing this risk factor, we can restrict the crowd-sourcing efforts to exactly those tuples which will strongly affect the result quality with high probability, and use suitable value predictions for all remaining tuples posing only a limited threat to the overall result quality. By adjusting the number of HITs to be issued, the algorithm can be customized in order to find the best trade-off between costs, performance, and result quality in each given application scenario. In particular, this allows for achieving a good skyline result quality quickly after issuing just a few actual crowd-sourcing tasks.

Our contribution in this paper can be summarized as follows:

- We design a *cost-aware workflow* for crowd-enabled skyline query execution over incomplete data
- We illustrate the *design space* within that workflow, and show different heuristics and implementation decisions
- We develop a *heuristic* for assessing the *risk* of a given predicted database tuple with respect to skyline quality

We provide an exhaustive *evaluation* of our heuristics and show that our approach provides high-quality results saving costs at least one order of magnitude less compared to naïve crowd-sourcing.

The remainder is structured as follows: the next chapter discusses related topics and how our system differs from previous approaches. Then, we present our workflow for crowd-enabled skyline queries with missing values, focusing on value prediction, prediction quality assessment, tuple risk computation, and crowd-sourcing of potentially harmful tuples. Finally, we provide extensive evaluations of our approach and conclude with a summary and outlook.

2. TOWARDS CROWD-ENABLED SKYLINE QUERIES

In the following section, we provide an overview of crowd-enabled databases and the state of the art for completing missing information in datasets. Furthermore, we briefly summarize skyline queries and current heuristic approaches for dealing with skyline computation on incomplete data.

2.1 Crowd-Enabled DBs and Missing Data

Crowd-enabled DBMS [1], [3], [4] fuse traditional relational technology with the cognitive power of people at query time, enabled by crowd-sourcing services like Amazon's Mechanical Turk, CrowdFlower, or SamaSource. The crowd can be used to complete data missing in tables in a *query-driven fashion*, i.e. queries can be executed despite incomplete data by either filling empty fields or inserting completely new tuples at runtime. Moreover, even complex cognitive tasks like reference reconciliation can be performed, and the crowd can be leveraged to "implement" database operators requiring cognitive abilities like scoring tuples with respect to perceived criteria (e.g., scoring images by visual appeal) or performing perceptual comparisons (e.g., tagging images with emotions or moods). While such cognitive tasks are a highly interesting feature with respect to personalization and preferences modeling, in the following we focus on the aspect of missing data.

In previous studies on crowd sourcing it has been shown under certain constraints that missing values in database tuples can be elicited with surprising efficiency and quality as long as the information is generally available [5]. Especially for factual data that can be looked-up on the Web without requiring expert knowledge (e.g., product specifications, telephone numbers, addresses, etc.), the expected data quality is quite high with only a moderate amount of quality assurance (e.g., majority votes). For example, [5] reports that crowd-sourced manual look-ups of movie genres in IMDB.com are correct in ~95% of all cases with costs of $0.03 per tuple (including quality assurance).

In a brief pre-study, we examined two typical aggregated datasets for e-commerce. Both contain technical specifications of cell phones and have been crawled in summer 2011 from meta-shopping directory sites, namely PhoneArena.com[2] and Heise.de[3]. Such datasets are very representative for our approach: they have been aggregated over some time integrating multiple sources, being subsequently employed to help users decide on a particular purchase. Furthermore, all data missing in these datasets is still available on the Web somewhere (e.g., the manufacturers' sites) and can thus be elicited via crowd-sourcing. However, obtaining this data automatically is very challenging. In particular, we observed that in the Heise dataset 10% of all values are missing (for 1,131 different phones with a total of 48 attributes), and for the PhoneArena dataset 26% of all values are missing (399 phones with 37 attributes).

2.2 Skyline Queries and Missing Information

Skyline Queries [6] are a popular personalization technique for databases, successfully bridging set-based SQL queries and top-k style ranking queries [7]. They implement the economic concept of *Pareto optimality* and thus allow for very intuitive and simple personalization: for each relevant attribute users simply provide a preference order under ceteris paribus semantics (e.g., "lower prices are preferred to higher prices assuming that all other attributes are equal"). Then, for any two tuples, where one tuple is preferred regarding one or more attribute(s) but equal with respect to the remaining attribute(s), rational users will always prefer the first object over the second one (the first object *Pareto dominates* the second one). Thus, the skyline set is computed by retrieving only tuples which are not dominated by any other tuple, i.e. all Pareto optimal tuples. Computing skyline sets is considered to be an expensive operation. Therefore, a variety of algorithms have been designed to significantly push the computation performance [8], e.g., by presorting [9], partitioning [10], or parallelization [11].

But regardless of the chosen algorithm, to compute skylines the value of each relevant attribute has to be accessed at least once for each tuple. Here, two major solution strategies can be chosen to deal with missing information: a) eliciting all missing information during / before computing the skyline in order to work on a complete dataset or b) dealing with missing information heuristically, which results in skylines afflicted with some error depending on the quality of the heuristic.

[2] http://www.phonearena.com/phones

[3] http://www.heise.de/mobil/handygalerie/

Heuristic approaches dealing with missing and/or uncertain data have been proposed in the past. Some approaches rely on using some default procedure to deal with missing information (or NULL values), e.g., by treating them as being incomparable or assuming the best/worst possible values [12]. One of the first works explicitly focusing on computing skylines over incomplete data is [13]. Here the actual Pareto semantics of skylines is changed: a tuple dominates other tuples, if it is better regarding at least one attributes and at least equal *or showing NULL values* in all other attributes. However, this implies non-transitive dominance relationships and may lead to cyclic dominance behaviour, which has to be resolved. However, the problem of missing information can also be treated differently: instead of changing semantics, algorithms on incomplete data should aim at computing approximate skyline sets as similar as possible to what would have been retrieved had all information been available. Therefore, in our work, the criteria for judging the quality of a skyline heuristic is the *error* induced by tuples with missing values as compared to the real skyline computed from a complete dataset.

Our goal is also loosely related to the area of *probabilistic databases*, since incompleteness can be seen as a source of uncertainty. Uncertainty is inherent in many emerging applications, such as sensor networks, data cleaning, moving objects, etc. Consequently, research in probabilistic databases has sparked much interest, and even skyline queries with uncertainty have already been considered. Most approaches follow the semantics of ‘possible worlds’: from a database perspective, uncertainty occurs either at attribute level [14] or at tuple level [15], [16]. There are two models to describe uncertain objects: continuous or discrete uncertainty models [17], [18]. For continuous uncertainty models, each uncertain object is assigned some interval that comprises all its probable values associated with a probability density function [19]. In contrast, discrete uncertainty models are utilized for applications lacking explicit density functions. Here, each uncertain object is represented by a set of instances. Each instance has a skyline probability that reflects the probability of its occurrence whilst not being dominated by any occurring instance of another object. Finally, the skyline probability of the object is the sum of all of its instances’ skyline probabilities [20], [21].

Since we focus on the case of *crowd-enabled databases*, we cannot rely on concurrently existing possible worlds. Uncertainty is considered on attribute level due to incompleteness and missing values. Thus, there are only two choices for each incomplete tuple during query processing: blindly accept some predicted value or safely crowd-source the task. But, since all predicted values introduce some uncertainty, suitable error bounds associated with the prediction algorithm have to be assigned. We claim that for determining these bounds outliers can be ignored and a heuristic quantification of the error in terms of false positives/negatives leads to cost-efficient and high-quality skyline query processing.

3. WORKFLOW OF CROWD-ENABLED SKYLINE QUERIES

In the following, we present the workflow of our heuristic system. For effectively optimizing crowd-sourcing costs the system should be *self-tuning*: for each database instance the best fitting parameters to reduce crowd-sourcing have to be decided heuristically while keeping the expected skyline correctness high. This design is desirable since the prediction algorithms’ quality varies greatly with the properties of the data. In this sense, our workflow in Figure 1 offers a toolbox allowing different implementation strategies for the basic components. Generally, predictions for all tuples with missing information are made before skylines can be computed. But, then for some high impact/risk tuples, the predicted values are superseded by crowd-sourcing judgements. Eventually, the skyline can be computed with high accuracy. We now outline the respective design space for all components and provide reference implementations as a basis for later evaluation:

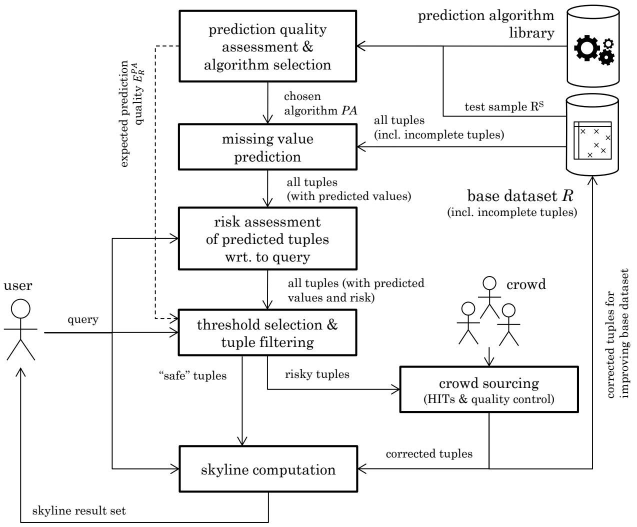
prediction quality assessment & algorithm selection
prediction algorithm library
expected prediction quality E_R^{PA}
chosen algorithm PA
test sample R^S
all tuples (incl. incomplete tuples)
missing value prediction
all tuples (with predicted values)
base dataset R (incl. incomplete tuples)
risk assessment of predicted tuples wrt. to query
user
all tuples (with predicted values and risk)
crowd
query
threshold selection & tuple filtering
corrected tuples for improving base dataset
“safe” tuples
risky tuples
crowd sourcing (HITs & quality control)
corrected tuples
skyline computation
skyline result set

Figure 1 System Workflow

Assume a relational *base dataset* R including incomplete tuples (judging from practice, at most 30% of tuples will be incomplete). Furthermore, we have a *library* of *prediction algorithms* and heuristics for dealing with incomplete data in skyline computation.

Definition 1 (Dataset): Formally, a base dataset R is an instance of a database relation $R \subseteq D_1^\square \times \ldots \times D_n^\square$ on n attributes $A_1, \ldots, A_n$ with $D_i^\square$ as domain of attribute A_i using $\square$ to denote a missing value, i.e. $D_i^\square = D_i \cup \{\square\}$. Each tuple t is denoted by $t := (t_1, \ldots, t_n)$. For simplicity and to allow for easy comparisons, in the rest of the paper we only consider *score values* with respect to preferences, i.e. numerical domains and linearized categorical preferences normalized to [0,1].

Moreover, the subset of all complete tuples R^C is given by $R^C := \{t \in R \mid \forall\, i \in \{1, \ldots, n\}\colon t_i \neq \square\}$ and the subset of all incomplete tuples is then denoted as $R^I := R \backslash R^C$.

Now assume a user is stating a *skyline query*. Such a query is given by any set of preferences over the attributes of the base dataset. A query may also contain additional meta-data for describing the required result quality, maximal query budget, response time requirements, etc. (see section 3.5).

Definition 2 (Numerical Preferences and Pareto Dominance): A numerical preference P_i over attribute A_i with a numerical domain D_i is a *total order* over D_i. If attribute value $a \in D_i$ is preferred over value $b \in D_i$, then $(a, b) \in P_i$, also written as $a >_i b$ (“a dominates b wrt. to P_i”). Analogously, we define $a \gtrsim_i b$ for $a >_i b$ or $a =_i b$. Without loss of generality, we consider only *maximum score* preferences (i.e. all score values are in [0,1], with 0 as worst and 1 as most preferred score).

We define the concept of *Pareto dominance* $t_1 >_P t_2$ between tuples $t_1, t_2 \in D_1 \times ... \times D_n$ by $t_1 \gtrsim_i t_2$ with respect to *all* attributes, and t_1 dominates $t_2$with respect to at least one attribute:
$t_1 >_P t_2 \Leftrightarrow \forall\, i \in \{1, ..., n\}: \mathrm{t}_1 \gtrsim_i \mathrm{t}_2 \ \wedge\ \exists i \in \{1, ... n\}: \mathrm{t}_1 >_i \mathrm{t}_2$

A *skyline query* is given by a set of preferences $P = \{P_1, ..., P_n\}$, with one preference for each attribute.

3.1 Missing Value Prediction

In the following, we provide a brief overview of value prediction algorithms. Numerous studies have already examined different approaches with varying complexities for dealing with missing data values. Some approaches simply discard incomplete instances by feature or object marginalization [22], acquire the missing data at a cost [23] (e.g., by manually eliciting them), or employ reduced or hybrid models [23]. However, the most common way is to estimate and replace missing values (in this paper called *value prediction*, in literature also often referred to as *imputation*).

There are two major families of imputation algorithms [24]: predictive value imputation and distribution based imputation. Whilst value imputation directly estimates missing values, distribution based imputation estimates the conditional distribution of missing values upon which the predictions will be based, e.g., in the C4.5 decision tree generation algorithm [25]. Most popular methods for predictive value imputation are mean imputation, median imputation, mode imputation, nearest neighbour imputation, regression imputation [26], and imputation using decision tree algorithms like CART [27]. More complex imputations seek to also reflect the correlation structure of data, and induce relationships between the available attribute values and missing features [28], e.g., surrogate splits for classification trees [29]. The correct choice of an imputation method depends on multiple factors:

- The *data's nature and the percentage of its missing values*. For example, imputation may be quite misleading when applied to astronomical datasets, where the missing values are often physically meaningful [30] and do not necessarily follow simple patterns.
- The *time of imputation*, i.e. at induction time (on historical training data) or at prediction time (in the test cases). Particularly, it has been shown that imputations are quite successful at induction time [31], [32].
- The *missing data model*: missing at random (MAR), missing completely at random (MCAR) or not missing at random (NMAT). while imputation can be a reasonable approach for both MAR and MCAR, there is no way to impute values reliably with NMAR, because the values were never observed [32].

Please note that in this paper we assume that values are at least *approximately missing at random*, as is often the case when aggregating or extracting datasets automatically.

In particular this means that for each tuple most attribute values are available, while just some are randomly missing. The same holds for attributes: for each attribute most tuples will have a respective value. In cases where for any attribute or tuple no or only very few values are given, the prediction algorithms may not return reliable results, and overall prediction quality will be low. Still, our system will be able to detect and deal with such cases by replacing unreliably predicted values with accurate crowd-sourced values in the workflow.

In our later experiments, we will mainly rely on *k-nearest neighbour imputation* (KNN) as it has been shown to be quite robust when the percentage of missing values increases [27]. Furthermore, KNN can predict both qualitative and quantitative attributes, doesn't require the creation of a predictive model for each attribute with missing data, and can work with instances having multiple missing values.

Additionally, aiming to also test our work using prediction algorithms with lower complexity and resulting precision, we chose the simpler *median imputation* technique. Unlike mean imputation, which is affected by the presence of outliers, utilizing the median assures robustness [27].

3.2 Assessing Prediction Quality

Before queries can be processed, the system has to assess the effectiveness of prediction algorithms with respect to the current dataset. Unfortunately, prediction effectiveness is hard to assess analytically. Therefore, we sample the current prediction quality by periodically measuring performance in a small test scenario. For this purpose, a suitable *test sample* $R^S \subseteq R^C$ is prepared by obfuscating complete tuples: values are removed in a similar pattern in which information is missing in the base dataset. In our reference implementation, we randomly remove values from the test sample such that the percentage of missing values per attribute matches the base dataset. We then apply each algorithm in the prediction algorithm library on this test sample to obtain predictions for the values which had just been removed. Thus, we can assess the expected prediction quality of each algorithm on the current dataset by measuring the error between the predicted values and the respective original values, e.g., by the overall *mean squared error* $\bar{e}^2$ of all predicted tuples and attributes.

Moreover, more detailed statistics about the behaviour of each algorithm is stored to be used in later stages of our workflow. In particular, for each attribute a_i we elicit the mean error $\bar{e}_i$ between the predictions and the real values, and the standard deviation σ_i^e of those errors. In order to increase the accuracy of the quality analysis at the cost of additional effort, the results of multiple sampling runs may be aggregated. Finally, the algorithm PA showing the smallest overall mean squared error is selected for later predicting all missing values.

Definition 3 (Error Vector): For prediction algorithm PA and base dataset R, an error analysis vector E_R^{PA} with $\bar{e}^2$ as mean squared error of all attributes, $\bar{e}_i$ as mean error of attribute a_i, and σ_i^e as respective standard deviation is given by: $E_R^{PA} = (\bar{e}^2, \bar{e}_1, ..., \bar{e}_n, \sigma_1^e, ..., \sigma_n^e)$

3.3 Quantifying Potential Skyline Errors

After the selected prediction algorithm has been used to predict all missing values for the current base dataset, the system has to decide for which tuples the predicted values are used during skyline processing - and for which tuples they are replaced by accurate values obtained from crowd-sourcing judgements. This step has the highest impact on the overall system performance, and has two extreme cases:

- *no tuple is crowd-sourced*, and only predicted values are used. Here, the costs are minimal (no crowd-sourcing necessary), and the result quality strictly depends on the accuracy of the used prediction algorithm. While in general, the quality achieved by this approach will be very low, for some highly predictable datasets (like the NBA dataset shown in chapter 4), it will nonetheless provide usable results at very low costs.

- *all tuples are crowd-sourced*, and all predictions are discarded and replaced. This extreme case will result in very high costs due to the large number of required crowd-sourcing HITs (which cost time and money). However, with adequate quality control, we can assume that all values obtained via crowd-sourcing are highly accurate. Therefore, this case shows very high result quality, especially when encountering systematically biased and therefore hard to-predict datasets.

In our system, we exploit the synergy between the two extreme cases by crowd-sourcing only the most *risky* tuples (i.e. tuples having a low prediction quality and having a possibly large impact on the skyline computation). For all tuples considered *safe* (e.g., tuples that have only a small effect on the skyline computation or can be predicted accurately), the predicted values are simply used. Therefore, the next crucial step is to assess the risk of each predicted tuple, and then tuples can be classified as being "risky" or "safe".

For illustrating the general problem space of this challenge, consider Figure 2: here, without loss of generality we focus on tuples in just two dimensions. Most tuples are complete (in accordance with our assumption that only up to ~30% of all values are missing at random). When ignoring all incomplete tuples the resulting skyline is given by $sky_c = \{s1, \ldots, s6\}$.

Definition 4 (Skyline of the Complete Subset): On the subset of complete tuples R^C and a set of preferences P, sky_c is defined as:
$$sky_c := skyline(R^C, P) = \{t_1 \in R^C \mid \nexists\, t_2 \in R^C : t_2 >_P t_1\}$$

The tuples with missing values in the base dataset are denoted by $p1, \ldots, p8$. In Figure 2, these tuples are shown in their *approximate* position when applying the chosen prediction algorithm. Obviously, in a 2-dimensional example at most one attribute value per tuple can be missing: let us assume $\{p1, p2, p4, p6\}$ are missing values in dimension 2 and $\{p3, p5, p7, p8\}$ in dimension 1. Since the predicted value may not necessarily be correct, in reality the respective tuples could be located anywhere along the dashed lines. It is easy to see that the quality of the skyline set computed by using predicted values can vary strongly depending on how correct the predictions are. For illustration again consider Figure 2: the skyline set using predicted values is $\{p1, s1, s2, p2, s3, p3\}$. However, now assume the value of $p3$ had been predicted vastly too high and its real value would allow $s3$ to dominate $p3$ (i.e. $p3$ lies left of the skyline frontier). Then the skyline quality would have several problems. First, $p3$ would not be part of the correct skyline: it is a *false positive*. Second, this prediction error also affects $s4$, $s5$, and $s6$, which are actually not dominated by $p3$ and therefore should have been part of the skyline; the three tuples are *false negatives*. All remaining tuples are not affected by the prediction error of $p3$ (*true positives* or *true negatives*). In summary, using $p3$'s predicted value may result in three false negatives and one false positive. Based on these observations, we developed a heuristic to rank all tuples needing prediction with respect to their potential to induce errors (in terms of false negatives or false positives). We will consider each predicted tuple individually.

It is easy to see that the only tuples with a chance to be false positives or false negatives are those with either predicted values, or complete tuples which are not dominated by any other complete tuple (i.e. all tuples in sky_c). Complete tuples dominated by any other complete tuple are always true negatives, and can therefore be ignored. Furthermore, the potential of introducing errors into the skyline computation heavily depends on the prediction accuracy of each single tuple. While in theory, the predicted tuples in Figure 2 could be located anywhere along the dashed lines, some locations are more likely than others, depending on the current tuple, the base dataset, and the chosen prediction algorithms. Unfortunately, correctly assessing the exact prediction accuracy per tuple is very complex. Therefore we use the following heuristic model to capture each tuple's potential to induce errors: we assume the prediction accuracy for each predicted tuple to be within *normal bounds*, i.e. we entirely ignore outliers having extremely irregular values which cannot be captured by the chosen prediction algorithm. For quantifying the "normal" accuracy behaviour of a given predicted tuple t^p, we rely on the algorithm accuracy statistics E_R^{PA} elicited during the algorithm selection step in chapter 3.2. With these statistics, we quantify the *prediction interval* of t^p assuming that the algorithm estimated any missing value including the respective systematic prediction error $\bar{e}_i$ and additionally overestimated or underestimated each value by the standard deviation, i.e. $\pm\sigma_i^e$. Then, the two interesting cases are the *upper bound tuple* t^+ and the *lower bound tuple* t^-, because these two tuples dominate the largest / lowest number of other tuples when finally computing the skyline under normal error assumption.

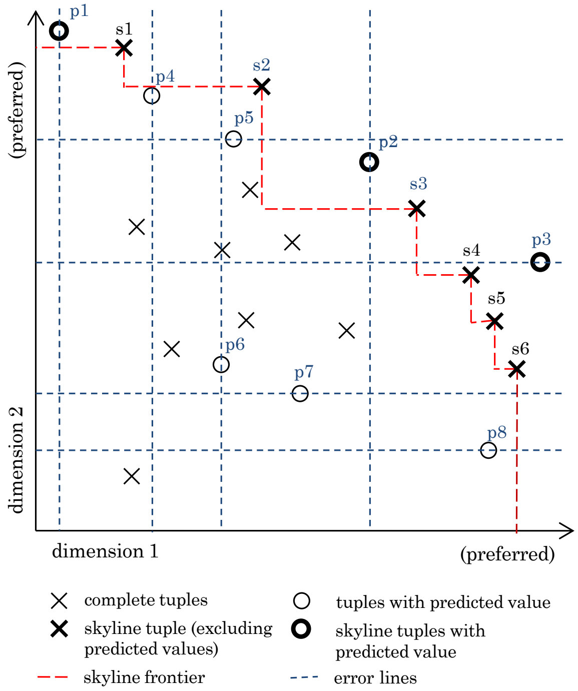

Figure 2: Predicted Tuples and the Skyline

Definition 5 (Upper/Lower Bound Tuple): Let $t \in R^I$ be a tuple with incomplete values, and t^p be the predicted tuple using some prediction algorithm. Then the upper/lower bound tuples t^+ and t^- are defined attribute-wise as follows:

$$t_i^+ = \begin{cases} if\ (t_i \neq \square):\ t_i \\ if\ \ (t_i = \square):\ (t_i^p + \bar{e}_i) + \sigma_i^e \end{cases}$$

$$t_i^- = \begin{cases} if\ (t_i \neq \square):\ t_i \\ if\ \ (t_i = \square):\ (t_i^p + \bar{e}_i) - \sigma_i^e \end{cases}$$

In the following, we focus on the expected errors for each predicted tuple t^p when computing the skyline of $R^C \cup \{t^p\}$, assuming that the real values for t^p are bounded by t^+ and t^-. For illustrating the possible effects on the skyline, we present four scenarios in Figure 5 which are discussed in the following. Later, we generalize our observations designing an algorithm which computes all possible false positives and false negatives for any given tuple: (Please note that we sum up the different variations t^+, t^p, and t^- as t when discussing the skyline sets):

- *Scenario a)* In this scenario, t^p is a skyline tuple (no tuple of sky_c dominates t^p) Therefore, when computing the skyline t^p dominates s_2. The resulting predicted skyline is $sky_p = \{s1, t, s3\}$. Now, if t is predicted too low within the normal bounds of error, then in the worst case (indicated by t^+) it could additionally dominate $s1$, and the correct skyline should be $\{t, s3\}$. From this perspective, $s1$ is a possible false positive in sky_p. On the other hand, if t is predicted to high, then in the worst case (t^-) it would still be a skyline object, but would not dominate $s2$ anymore. This results in the skyline set $\{s1, s2, t, s3\}$. Therefore, in this case, $s2$ could be a false negative with respect to the predicted skyline sky_p. Hence, assuming normal prediction accuracy t^p could induce up to one false positive, and one false negative.
- *Scenario b)* Again t^p a predicted skyline tuple, resulting in the skyline $sky_p = \{s1, s2, t, s3\}$. Similar to scenario a), $s2$ can be a false positive, if t is predicted too low ($s2$ is dominated by t^+). But additionally, if t is predicted too high, and in reality is t^-, then the resulting skyline would be $\{s1, s2, s3\}$, and t itself would be a false positive. Therefore, in this scenario, two different false positives can result from normal error behaviour.
- *Scenario c)* In this scenario, t^p is no skyline tuple, resulting in $sky_p = \{s1, s2, s3\}$. But if t was predicted too low (its real value is t^+), then it becomes a skyline objects, and therefore t itself is a false negative in sky_p. In this case it would dominate $s1$, which in turn results in a false positive.
- *Scenario d)* Finally, there may be tuples which do not result in any false positives or negatives under normal error conditions. This is shown in this last scenario.

Based on the previous example scenarios, we will now formalize the set of false positives and false negatives for a given predicted tuple t^p in several easy-to-implement rules.

False positives can be computed using the rule out of the following four having a matching precondition (see Figure 4):

a) If t^p is a skyline tuple, and no tuple dominates its least preferable tuple t^-, then all tuples which are dominated by t^p, but not by t^- are potential false positives.

b) Analogously to the previous case, but t^- is dominated by a skyline tuple. Therefore, tuple t^+ can be a false positive.

c) If t^p is not a skyline tuple, then it can be responsible for false positives only if t^+ dominates some skyline tuples (which then would not be skyline objects anymore).

d) If both t^p and t^+ are dominated, then no false positives can occur within normal error bounds.

Formally, these rules lead to the following definition:

Definition 6 (Set of False Positives): Let $t^p = PA(t)$ be a predicted tuple with its upper/lower bound tuples t^+ and t^-. Also, sky_c is the skyline of all tuples in R^C with respect to the preferences P. Then, the set of possible false positives $fp(t^p)$ can be computed by the one of these four rules:

a) **If** $\left(\nexists\, s \in sky_c\colon (s >_P t^p)\right) \land \left(\nexists\, s \in sky_c\colon (s >_P t^-)\right)$ **then**
$$fp(t^p) = \left\{s \in sky_c \middle| (t^+ >_P s) \land \left(t^p \not>_p s\right)\right\}$$

b) **If** $\left(\nexists\, s \in sky_c\colon (s >_P t^p)\right) \land \left(\exists\, s \in sky_c\colon (s >_P t^-)\right)$ **then**
$$fp(t^p) = \left\{s \in sky_c \middle| (t^+ >_P s) \land \left(t^p \not>_p s\right)\right\} \cup \{t^p\}$$

c) **If** $\left(\exists\, s \in sky_c\colon (s >_P t^p)\right) \land \left(\exists\, s \in sky_c\colon (t^+ >_P s)\right)$ **then**
$$fp(t^p) = \left\{s \in sky_c \middle| (t^+ >_P s)\right\}$$

d) **If** $\left(\exists\, s \in sky_c\colon (s >_P t^p)\right) \land \left(\nexists\, s \in sky_c\colon (t^+ >_P s)\right)$ **then**
$$fp(t^p) = \emptyset$$

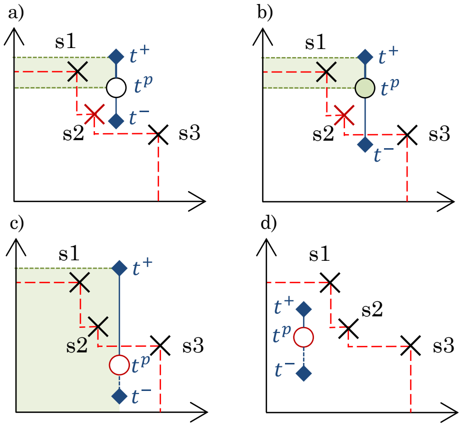

Figure 3: False Positives assuming normal error behaviour

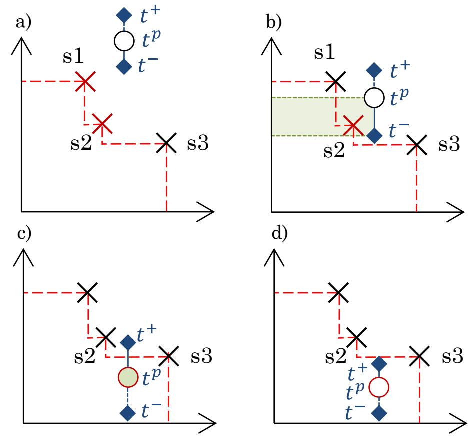

Figure 4: False Negatives assuming normal error behaviour

Analogously, the set of *false negatives* is defined as follows (see Figure 5):

a) If t^p is a skyline tuple, and all tuples dominated by t^p are also dominated by t^-, then there can be no false negatives.

b) If t^p is a skyline tuple, and there exists tuples dominated by t^p, but not by t^-, then all those tuples are potential false negatives.

c) If t^p is not a skyline tuple, but t^+ would be in the skyline, then t^p is a potential false negatives.

d) If both t^p and t^+ are no skyline tuples, then there can be no false negatives.

Definition 7 (Set of False Negatives): Let t^p be a predicted tuple with its upper/lower bound tuples t^+ and t^-. Let sky_c be the skyline of all tuples in R^C with respect to the preferences P. Then, the set of possible false negatives $fn(t^p)$ can be computed by the one of these four rules:

a) **If** $\left(\nexists s \in sky_c: (s >_P t^p)\right) \wedge$
$\left(\forall\, s \in sky_c: (t^p >_P s) \Rightarrow (t^- >_P s)\right)$ **then**
$fn(t^p) = \emptyset$

b) **If** $\left(\nexists s \in sky_c: (s >_P t^p)\right) \wedge$
$\left(\exists\, s \in sky_c: (t^p >_P s) \nRightarrow (t^- >_P s)\right)$ **then**
$fn(t^p) = \left\{s \in sky_c \middle| (t^p >_P s\,) \wedge (t^- \ngtr_p s)\right\}$

c) **If** $\left(\exists s \in sky_c: (s >_P t^p)\right) \wedge \left(\nexists s \in sky_c: (s >_P t^+)\right)$ **then**
$fn(t^p) = \{t^p\}$

d) **If** $\left(\exists s \in sky_c: (s >_P t^p)\right) \wedge \left(\exists s \in sky_c: (s >_P t^+)\right)$ **then**
$fn(t^p) = \emptyset$

The following two lemmas are necessary to show the completeness and non-ambiguousness of the rules in Definition 6 and 7:

Lemma 1 (Definition 6 is complete and non-ambiguous):
Let$C_1 \equiv\ \exists\, s \in sky_c: (s >_P t^p)$, and $C_2 \equiv \exists\, s \in sky_c: (s >_P t^-)$, and $C_3 \equiv \exists\, s \in sky_c: (t^+ >_P\ s)$.
Then the preconditions of the four cases of Definition 6 can be written as follows:
a) $A \equiv \neg C_1 \wedge \neg C_2$ b) $B \equiv \neg C_1 \wedge C_2$
c) $C \equiv C_1 \wedge C_3$ d) $C \equiv C_1 \wedge \neg C_3$
Furthermore, additional semantic constraints resulting from the definition of Pareto dominance follow: C_1 is a stronger version of C_2, therefore $C_1 \Rightarrow C_2$. Also, with respect to false positives, condition C_3 does not matter at all whenever t^p is a skyline object, i.e. $\neg C_1$ holds. Therefore, we can rewrite:
a) $A \equiv \neg C_1 \wedge \neg C_2 \wedge (C_3 \vee \neg C_3)$ b)$B \equiv \neg C_1 \wedge C_2 \wedge (C_3 \vee \neg C_3)$
c) $C \equiv C_1 \wedge C_2 \wedge C_3$ d) $D \equiv C_1 \wedge C_2 \wedge \neg C_3$
It is easy to see that $(A \vee B \vee C \vee D) \equiv true$, showing that the conditions are complete ($C_1 \wedge \neg C_2$ is not possible due to $C_1 \Rightarrow C_2$).
Also,$(A \wedge B) \vee (A \wedge C) \vee (A \wedge D) \vee (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \equiv false$ holds, showing that the conditions are pairwise disjunctive. ■

Lemma 2 (Definition *7* is complete and non-ambiguous):
Let $C_1 \equiv\ \exists s \in sky_c: (s >_P t^p)$, and $C_2 \equiv \exists s \in sky_c: (s >_P t^+)$, and $C_3 \equiv \exists\, s \in sky_c: (t^p >_P s) \nRightarrow (t^- >_P s)$.
Then the preconditions of the four cases of Definition *7* can be written as follows:
a) $A \equiv \neg C_1 \wedge \neg C_3$ b) $B \equiv \neg C_1 \wedge C_3$
c) $C \equiv C_1 \wedge \neg C_2$ d) $C \equiv C_1 \wedge C_2$
Similar to Lemma 1, additional semantic constraints $C_2 \Rightarrow C_1$ and $\neg C_1 \Rightarrow \neg C_2$ ("if t^p is a skyline object, no object can dominate t^+") hold. Also, C_3 is irrelevant for false negatives whenever t^p is not a skyline object (C_1). Therefore, we can rewrite:
a) $A \equiv \neg C_1 \wedge \neg C_2 \wedge \neg C_3$ b) $B \equiv \neg C_1 \wedge \neg C_2\ \wedge C_3$
c) $C \equiv C_1 \wedge \neg C_2 \wedge (C_3 \vee \neg C_3)$
d) $D \equiv C_1 \wedge C_2 \wedge (C_3 \vee \neg C_3)$
Again, $(A \vee B \vee C \vee D) \equiv true$ holds (note that $\neg C_1 \wedge C_2$ is not possible due to $C_2 \Rightarrow C_1$), and also
$(A \wedge B) \vee (A \wedge C) \vee (A \wedge D) \vee (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \equiv false$
■

3.4 Ranking Tuples for Crowd-Sourcing

Using the cardinalities of the sets of false negatives $fn(t^p)$ and false positives $fp(t^p)$ for each predicted tuple t^p, we are able now to finally assign a score for ranking tuples with respect to their priority to be crowd-sourced. This score reflects the potential severity of the error introduced by a predicted tuple when prediction quality is within normal bounds.

Definition 8 (Tuple Score): Given the set of false negatives $fn(t^p)$, false positives $fp(t^p)$, and a weighting factor $\alpha \in [0,1]$, the score of a tuple t^p can be computed as:

$$score(t^p) = \alpha * |fn(t^p)| + (1 - \alpha) * |fp(t^p)|$$

The weighting factor α can be used to adapt the ranking behaviour to reflect different use case requirements with respect to false positives and false negatives. We propose the usage of one of the following three heuristics:

- $\alpha = 0.5$: A default setting which handles false negatives similar to false positives.
- $0.5 < \alpha \leq 1$: This setting emphasizes the weight of false negatives and can be motivated from the user's point of view: when a user examines the skyline result set, false negatives mean that a Pareto-optimal tuple is not visible for the user, limiting his/her choices and maybe preventing an optimal decision. In contrast, although false positives burden the user with additional cognitive overhead for filtering, they still allow for an optimal decision. Therefore, false negatives can be seen as being worse.
- Auto-adjust α: The weighting factor α can also be determined automatically for each dataset. The following simulated crowd-sourcing can be automatically performed for different values of α: using the prepared sample R^S (see 3.2), all missing values are predicted and then tuples are ranked with the current α. Then, the top scored tuples are replaced by correct values, as if crowd-sourced. After computing skylines, we choose the α having the smallest error compared to the real skyline. This results in the value of α which will most probably minimize the resulting skyline error. While this sampling process imposes additional computational effort, this effort is low compared to a later execution of crowd-sourcing HITs.

Please note that the informedness error measure used in our experiments (see 3.6) does not distinguish between false positives and false negatives, hence the effect of α is minor. However, when using different error measures, auto-adjusting and choosing a correct α may become more significant.

3.5 HIT Creation

After all predicted tuples are scored, "safe" and "risky" tuples need to be determined. This is done by ranking all tuples according to their score, and selecting the top-k highest scored tuples for crowd-sourcing (as they have the highest error potential, and are thus unsafe). k should be chosen in accordance with quality requirements or additional meta-data provided by the user. There are many different options to how k can be obtained, allowing for a high degree of customization, e.g.:

- The user directly provides k as part of the query.
- The user provides a maximum tolerable error. Then, the system can again rely on sampling (similar to determining α automatically) for determining k such that it provides the required quality.

- The user does not provide any additional constraints. Then, the system can again rely on sampling to find the best trade-off between the result quality and the number of crowd-sourced tuples. We will briefly discuss this in section 4.2.
- Enhancing previous options, the users could also provide monetary or time constraints, limiting the maximum or minimum number of tuples to be crowd-sourced.

After all unsafe tuples have been determined, HITs are created and issued to a suitable crowd-sourcing platform. The size of the HITs and required quality control procedures (e.g., majority votes, gold questions, etc.) depend on the type of missing information (see [33] for more details on different types of missing information and quality control). After all crowd judgements have been collected, the obtained values can be written to the base dataset as completed tuples, therefore decreasing the degree of missing values over time. Finally, the skyline is computed (using predicted values for safe tuples) and returned to the user.

Definition 9 (Final Skyline): Given the subset of complete tuples R^C and incomplete tuples R^I, a prediction algorithm PA with the error vector E_R^{PA}, and the weighting factor α and ranking limit k, the final predicted skyline can be computed with the following algorithm:

$$R^{predicted} = \bigcup_{t \in R^I}(complete\ t\ using\ PA)$$
$$R^{safe} = \{t^p \in R^{predicted} | Rank_{t^p}\left(score(t^p)\right) > k)\}$$
$$R^{unsafe} = \{t^p \in R^{predicted} | Rank_{t^p}\left(score(t^p)\right) \leq k)$$
$$R^{crowd} = \bigcup_{t^p \in R^{unsafe}}(crowdsource\ t^p)$$
$$R^f = R^C \cup R^{safe} \cup R^{crowd}$$
$$sky_f = skyline\left(R^f, P\right) = \{t_2 \in R^f \mid \nexists\, t_1 \in R^f : t_1 >_P t_2\}$$

In our initial system workflow as presented in Figure 1, only a single crowd-sourcing iteration is performed, i.e. all unsafe tuples are crowd-sourced in one large batch, distributed over several HITs executed in parallel for minimal response time. This means, tuple ranking relies on many assumptions, as our tuple score is computed for each tuple individually respecting only complete tuples. However, if query response times are not a priority, higher quality with less crowd-sourcing operations (and therefore lower monetary costs) can be achieved by issuing smaller batches, and then dynamically re-ranking all tuples after each batch. As HITs cannot be executed in parallel anymore, this approach obviously takes longer. The effects of various *batch sizes* are also evaluated in section 4.4.

3.6 Measuring Skyline Error

For measuring the actual error of a skyline set using predicted values, we rely on a popular metric from information retrieval. In particular, we expand on the measure of *informedness* [34]. Informedness quantifies how informed a computed result is when compared to a result derived by chance. The informedness measure is based on recall and inverse recall. In contrast to using recall alone, it considers both error types, false positives and false negatives, while at the same time also respecting true positives and true negatives. Therefore, it is a quite fair and unbiased measure, which is also used in our evaluations in chapter 4 and during sample runs for determining k automatically.

Definition 10 (Skyline Error): Let R^P be a dataset with predicted and/or crowd-sourced values, and R^* the respective dataset containing the same tuples, but missing values replaced with their real values, with sky_p and sky_{R^*} being the respective skylines. The error between both sets can be computed by (some arguments omitted):

$$error(sky_{R^{\mathrm{P}}}, sky_{R^*}) = 1 - informdness(..)$$

$$informedness(sky_{R^{\mathrm{P}}}, sky_{R^*}) \\ = recall(..) + invRecall(..) - 1$$

$$recall(sky_{R^{\mathrm{P}}}, sky_{R^*}) = \frac{truePositives(..)}{truePositives(..) + falseNegatives(..)}$$

$$invRecall(sky_{R^{\mathrm{P}}}, sky_{R^*}) = \frac{trueNegatives(..)}{trueNegatives(..) + falsePositives(..)}$$

4. EVALUATIONS

In the following section, we extensively evaluate our proposed heuristics and system workflow with respect to different influence factors and configuration parameters on multiple real world datasets:

a) Our first dataset is the well-known NBA player statistics (http://www.basketballreference.com), as frequently used in skyline research. It consists of 21,961 tuples. For each player, we used 6 attributes, i.e. games played, points scored, rebounds, assists, and goals. We use maximum preferences, resulting in a skyline of 75 tuples.

b) The second dataset contains 1,597 notebooks, crawled in 2010 from Dooyoo.de (http://www.dooyoo.de/notebook). This dataset features 8 attributes: CPU frequency (maximum), CPU type (categorical preference encoded by a score), RAM (max.), HD (max.), display size (max.), weight (min.), and manufacturer (categorical score), resulting in a skyline of 35 tuples.

c) The third dataset describes different car models, crawled from Heise.de (http://www.heise.de/autos/neuwagenkatalog) in 2011, contains 7,755 tuples with the following attributes: price (min.), power (max.), acceleration (max.), fuel consumption (min.), CO_2 emission (min.), and taxes (min.). It results in a skyline of 268 tuples.

Unless explicitly stated otherwise, for the following experiments, we assumed 20% of missing values and α = 0.6 to slightly emphasize false negatives. All experiments but 4.5 rely on simulation; the presented results are average results of 100 simulation runs for each experiment.

4.1 Prediction Quality on Different Datasets

Two prediction algorithms were chosen for the following experiments: k-nearest neighbour prediction (KNN), and median prediction. According to section 3.2, we assessed the prediction quality of those two algorithms for all datasets, resulting in error vectors $E_R^{PA} = (\bar{e}^2, \bar{e}_1, \ldots, \bar{e}_n, \sigma_1^e, \ldots, \sigma_n^e)$.

The results for each dataset are shown in Tables 1-3 (we show the absolute values of $\bar{e}_i$). In general, we can summarize that KNN unsurprisingly shows much better prediction performance for all datasets. Furthermore, on average, missing values within the NBA dataset can be predicted significantly better than values for the other datasets. Moreover, we can see that even within each dataset, some attributes can be predicted more accurately than others, e.g., games played in the NBA set has a mean error of ~0.34, while most other attributes of the NBA set have mean errors ~0.07. This experiment clearly stresses the importance of choosing a correct algorithm (especially when the library contains multiple advanced prediction algorithms besides KNN), and respecting different prediction behaviours between attributes, as this directly reflects on the number of risky tuples which should be crowd-sourced.

4.2 Effectiveness of the Ranking Heuristic & Crowd-Sourcing

In the following, we will showcase the effectiveness of our ranking heuristic for all three datasets and both prediction algorithms by incrementally crowd-sourcing one tuple at a time and measuring the skyline error (20% missing values, $\alpha = 0.6$, batch size = 1). Figure 6 and 7 show results for our ranking heuristic to select the next tuple to be crowd-sourced, instead of randomly selecting any incomplete tuple for crowd-sourcing. With the notable exception of the NBA dataset using KNN prediction, it is clearly visible that by pure prediction (i.e. no crowd-sourced tuples), the skyline quality is very low (error for *median*: NBA 28.5%, notebook 45.4%, cars 81%; error for *KNN*: NBA 7.2%, notebook 28.4%, cars 62%).

Also, for most cases, the skyline error is significantly reduced when using our heuristic after just a few crowd-sourced tuples (i.e. by increasing k), while this effect is much less pronounced if tuples are randomly selected. For example, consider cars with KNN prediction: for decreasing the error from 60% down to 20% only 27 tuples needs to be crowd-sourced on average using our heuristic, while for reaching a similar improvement with randomly selected tuples 145 tuples need to be crowd-sourced. Also note that 20% missing values translates to an absolute of 4,392 tuples missing in the NBA dataset, 319 tuples for notebooks, and 1,551 tuples for cars. Therefore, tremendous effort can be saved, if users are willing to accept minor reductions with respect to skyline quality.

The high prediction accuracy of the NBA dataset (which was already visible in the last experiment) leads to an already low initial skyline error when only relying on prediction, leaving only limited room for improvement. By choosing k dynamically our system can take advantage of this fact, e.g. when a user provided quality constraint enforces a maximum error of 10%, no crowd-sourcing needs to be performed.

Furthermore, these experiments show another interesting effect which can be exploited: when using our heuristic, the error is reduced very quickly for the first few crowd-sourcing operations, but the quality improvements will slow down after a while. This means, for most datasets, there is a k, for which we have the optimal trade-off between low error rates, and low query execution costs. We can use this observation for automatically determining the most efficient k during sampling runs prior to the actual crowd-sourcing as described in chapter 3.5 by determining the *inflection point* of the error curve. For example, using the cars dataset with KNN, by crowd-sourcing 36 or 76 tuples (both being inflection points), a very good ratio between quality and costs can be achieved.

4.3 Impact of Missing Value Percentages

Now we examine the impact of the percentage of missing values in the base dataset by measuring skyline error from 1% up to 20% missing values. In Figure 8, the error for the cars dataset is illustrated for $k = 0$ (using only prediction) and $k = 20$ (which is a rather low value given that there are 1,551 tuples with missing values). Comparing the skyline error computed purely prediction-based versus a small number of crowd-sourced tuples, the difference is substantial. At 20% missing values the observed error is almost double, reaching 60% for $k = 0$ and about 33% for $k = 20$. We repeated the same experiment for the NBA dataset, shown in Figure 9. Here, the skyline error increases only slowly for increasing numbers of incomplete tuples due to the high accuracy of the KNN predictor, with only little potential for crowd-sourcing tasks to improve the result ($k = 4$, corresponding to the inflection point of the error curve).

Table 1: Computed error analysis vectors for NBA dataset

NBA	KNN		Median	
	e^{-2}: 0.0510		e^{-2}: 0.053	
	$\lvert\bar{e}_i\rvert$	σ_i^e	$\bar{e}_i$	σ_i^e
Games played	0.34	0.38	0.43	0.41
Points scored	0.08	0.08	0.45	0.24
Total rebounds	0.05	0.07	0.07	0.07
Assists	0.08	0.07	0.08	0.07
Field goals made	0.06	0.07	0.06	0.07

Table 2: Computed error analysis vectors for cars dataset

Cars	KNN		Median	
	e^{-2}: 0.077		e^{-2}: 1.135	
	$\lvert\bar{e}_i\rvert$	σ_i^e	$\bar{e}_i$	σ_i^e
Price	0.25	0.5	0.83	0.65
Power	0.10	0.12	0.75	0.39
Acceleration	0.22	0.34	0.45	0.38
Fuel consumption	0.20	0.35	0.12	0.33
CO2 Emission	0.21	0.35	0.06	0.33
Taxes	0.26	0.4	0.12	0.35

Table 3: Computed error analysis vectors for notebooks dataset

Notebooks	KNN		Median	
	e^{-2}: 0.093		e^{-2}: 1.376	
	$\lvert\bar{e}_i\rvert$	σ_i^e	σ_i^e	σ_i^e
CPU	0.36	0.39	0.67	0.47
CPU Frequency	0.25	0.35	0.14	0.35
RAM	0.25	0.31	0.21	0.29
Hard drive	0.15	0.21	0.12	0.21
Display Type	0.24	0.40	0.39	0.42
Weight	0.15	0.35	0.13	0.34

4.4 Other Influence Factors and Parameters

In the next experiments, we examine the effects of batch sizes (chapter 3,5) and the weighting factor α (chapter 3.4).

We used relative batch sizes, i.e. the number of tuples which are crowd-sourced in one batch before tuples are re-ranked depends on the number of predicted skyline tuples. The results are shown in Figure 10: a batch size of 5% (i.e. 2 tuples per batch) gives a faster improvement per crowd-sourced tuple (and therefore per dollar) than larger batches. However, since less crowd-sourcing tasks can be performed in parallel, this results in longer query times. Measured for different batch sizes, improving the skyline error from 65% to 25%, 12 tuples for 5% batch size, 26 tuples for 25% batch size, and 45 tuples for 50% batch size are required. Therefore, for batch sizes, there is a clear trade-off between quality and costs.

In the next experiment, we examined the effects of different α values. In order to test the impact of different α values, the underlying skyline error measurement needs to weight false negatives and false positives with different emphasis, otherwise changing α values won't be reflected in the computed error. The informedness measure which we used in all other experiments lacks this property; therefore we used the well-known F-measure in this experiment. F-measure is the harmonic mean weighted average of both precision and recall. The computed value falls between [0,1] with 1 being the best and 0 being the worst. The factor β in the following definition puts additional weight on precision or recall, and therefore also on false negatives or false positives. For this experiment, we use β=5 in order to penalize the false negative.

$$F_\beta \; = \; (1+\beta^2) * \frac{Precision*Recall}{(\beta^2 * \, Precision)+Recall}$$

The results are shown in Figure 11, and it can be seen that different α values lead to very similar performance with respect to the f-measure. Therefore, the choice of α is mostly a matter of user preferences and has only little impact on common error measures.

4.5 Crowd-Sourcing Costs: Time and Money

Finally, we measure the efficiency of our approach in terms of time and money in a real crowd-sourcing experiment. For this, we rely on the crowd-sourcing platform CrowdFlower.com. We focus again on the cars dataset with $k = 80$ and 25% batch size (corresponding to an average of 14 tuples per batch). This yielded 6 crowd-sourcing batches, 14 tuples each, with the last batch comprised of only 10 tuples. Those batches have been executed sequentially, and all predicted tuples were re-ranked after each batch. To assure high quality, a majority vote of four crowd judgments was performed for each tuple. Furthermore, we used 28% Gold questions (i.e. tasks for which the correct results are known upfront in order to filter out unreliable workers, see [33]). These quality measures increase the number of crowd judgements to an overall of 416 judgments for obtaining the correct values of 80 tuples. During a calibration run, it turned out that performing a single judgement takes about 60 seconds. Following the general guidelines of CrowdFlower, we paid $0.19 per 5 judgements (i.e. one HIT are 5 judgements), for a total cost of $5.01 per batch (for 72 crowd judgements overall, yielding the correct results for 14 tuples, i.e. ~5 judgements per tuple; also, additional fees for administration and platform usage are incorporated). This results in an average runtime of 25 minutes for one batch.

The end results are shown in Figure 12 and 13, where the error reduction from initially 62% down to 12% costs roughly $28, and takes 125 minutes. The query time required can be decreased by increasing the batch size in order to exploit that more crowd-sourcing tasks can be executed in parallel.

5. SUMMARY AND OUTLOOK

In this paper, we presented a novel approach for the challenge of dealing with missing information in datasets in connection with skyline query processing. This challenge can be naively approached by using crowd-enabled databases, eliciting all missing values from the crowd before computing the skyline. However, we showed that by using carefully tailored heuristics and prediction algorithms, only a tiny fraction of all incomplete tuples actually need to be crowd-sourced in order to obtain high quality approximate skyline results at very low costs. This was achieved by identifying and crowd-sourcing only those tuples which potentially have a large negative impact on the result set quality, while using standard value prediction algorithms for approximating all other tuples. For instance, by crowd-sourcing only 80 out of 1,551 incomplete tuples for a practical car dealer dataset, the resulting skyline error was only 10% when using our heuristic measures compared to 35% for random crowd-sourcing. Based on this idea, we presented a workflow for a fully self-tuning and adaptable system that can be easily deployed for different prediction algorithms, various datasets, and result quality requirements. In our extensive evaluations, we showed the superior performance of our approach, and identified the impact of relevant influence factors.

In future work, we will expand on this concept by exploring additional heuristics and optimization techniques, and also

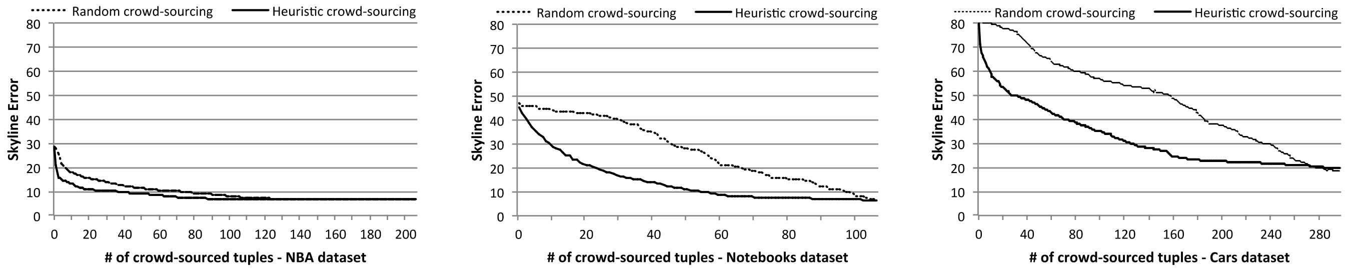

Figure 6: Decreasing skyline error while crowd-sourcing using *Median* prediction (*heuristic* and *random* tuple selection)

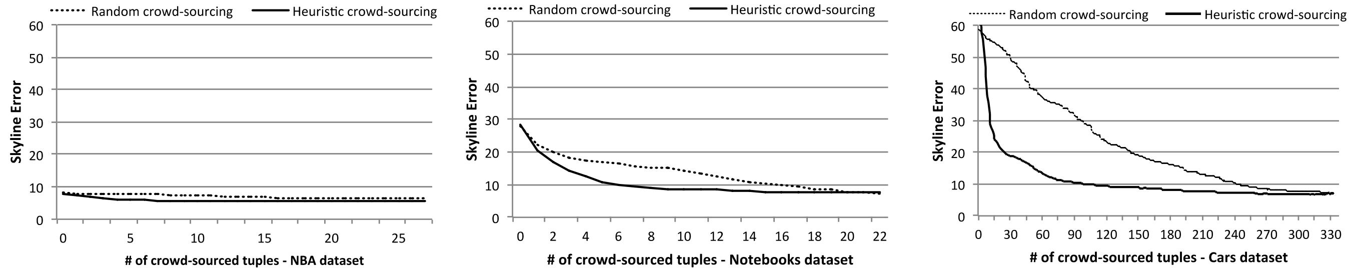

Figure 7: Decreasing skyline error while crowd-sourcing using *KNN* prediction (*heuristic* and *random* tuple selection)

incorporate additional prediction algorithms, like for example crowd-enabled value prediction techniques [35]. Furthermore, we will adapt our system to additional problems domains like top-k retrieval on incomplete data.

6. ACKNOWLEDGEMENTS

Parts of this work were supported by funds provided by the DAAD FIT programme (http://www.daad.de/fit/).

REFERENCES

[1] M. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, “CrowdDB: Answering queries with crowdsourcing,” in *ACM SIGMOD Int. Conf. on Management of Data*, 2011.

[2] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, “Human-powered sorts and joins,” *Proceedings of the VLDB Endowment*, vol. 5, no. 1, pp. 13–24, Sep. 2011.

[3] A. Parameswaran and N. Polyzotis, “Answering queries using humans, algorithms and databases,” in *Conf. on Innovative Data Systems Research (CIDR)*, 2011.

[4] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, “Crowdsourced Databases : Query Processing with People,” in *Conf. on Innovative Data Systems Research (CIDR)*, 2011.

[5] C. Lofi, J. Selke, and W.-T. Balke, “Information Extraction Meets Crowdsourcing: A Promising Couple,” *Datenbank-Spektrum*, vol. 12, no. 1, 2012.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker, “The Skyline Operator,” in *Int. Conf. on Data Engineering (ICDE)*, 2001.

[7] R. Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for middleware,” in *Symposium on Principles of Database Systems (PODS)*, 2001.

[8] P. Godfrey, R. Shipley, and J. Gryz, “Algorithms and analyses for maximal vector computation,” *The VLDB Journal*, vol. 16, no. 1, pp. 5–28, Sep. 2007.

[9] I. Bartolini, P. Ciaccia, and M. Patella, “Efficient sort-based skyline evaluation,” *ACM Transactions on Database Systems*, vol. 33, no. 4, 2008.

[10] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “Progressive skyline computation in database systems,” *ACM Trans.*

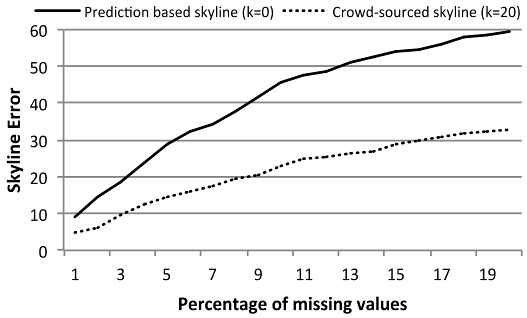

Figure 8: Skyline error for different %missing values (cars)

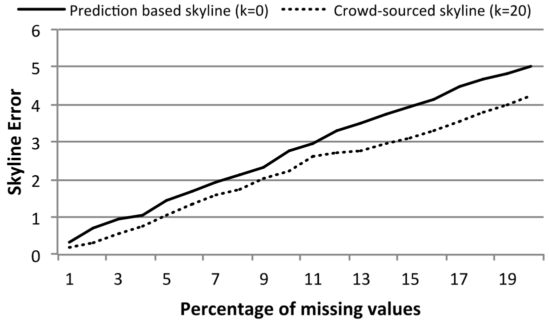

Figure 9: Skyline error for different %missing values (NBA)

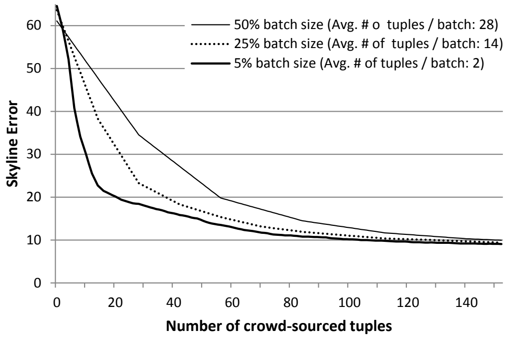

Figure 10: Quality with different batch sizes

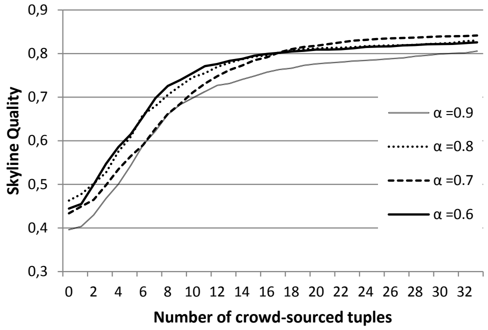

Figure 11: Effect of α values (cars dataset – y-axis uses f-measure)

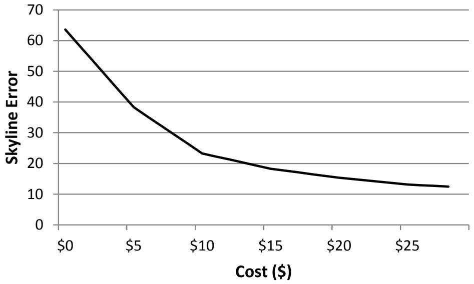

Figure 12: Crowd-Sourcing Costs in Dollars (Cars data)

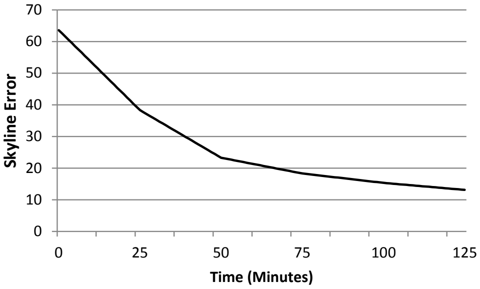

Figure 13: Time required for crowd-sourcing

Database Syst., vol. 30, no. 1, pp. 41–82, 2005.

[11] J. Selke, C. Lofi, and W.-T. Balke, "Highly Scalable Multiprocessing Algorithms for Preference-Based Database Retrieval," in *Int.Conf.on Database Systems for Advanced Applications (DASFAA)*, 2010.

[12] W. T. Balke, U. Güntzer, and W. Siberski, "Restricting skyline sizes using weak Pareto dominance," *Informatik - Forschung und Entwicklung*, vol. 21, no. 3–4, pp. 165–178, May 2007.

[13] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline Query Processing for Incomplete Data," in *Int. Conf. on Data Engineering (ICDE)*, 2008, vol. 30, no. 2.

[14] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and Simple Relational Processing of Uncertain Data," in *Int. Conf. on Data Engineering (ICDE)*, 2008.

[15] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu, "MYSTIQ - A system for finding more answers by using probabilities," in *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, 2005.

[16] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom, "ULDBs: databases with uncertainty and lineage," in *Int. Conf. on Very Large Data Bases (VLDB)*, 2006.

[17] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng, "Database Support for Probabilistic Attributes and Tuples," in *Int. Conf. on Data Engineering (ICDE)*, 2008.

[18] Y. Zhang, W. Zhang, X. Lin, B. Jiang, and J. Pei, "Ranking uncertain sky: The probabilistic top-k skyline operator," *Information Systems Journal*, vol. 36, no. 5, pp. 898–915, 2011.

[19] C. Böhm, F. Fiedler, A. Oswald, C. Plant, and B. Wackersreuther, "Probabilistic skyline queries," in *Int. Conf. on Information and Knowledge Management (CIKM)*, 2009.

[20] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Int. Conf. on Very Large Data Bases (VLDB)*, 2007.

[21] M. J. Atallah and Y. Qi, "Computing all skyline probabilities for uncertain data," in *ACM SIGMOD Symp. On Principles Of Database Systems (PODS)*, 2009.

[22] K. L. Wagstaff and V. G. Laidler, "Making the Most of Missing Values : Object Clustering with Partial Data in Astronomy," *Analysis*, vol. 347, pp. 1–5, 2005.

[23] M. Saar-tsechansky and F. Provost, "Handling Missing Values when Applying Classification Models," *Journal of Machine Learning Research*, vol. 8, no. 1625–1657, pp. 1625–1657, 2007.

[24] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*. Springer, 2001, p. XVI, 533 N5 – Supervised learning (Machine learni.

[25] J. R. Quinlan, *C4.5: Programs for Machine Learning*, vol. 240, no. 3. Morgan Kaufmann, 1993, p. 302.

[26] Z. Ghahramani and M. I. Jordan, "Supervised learning from incomplete data via an EM approach," in *Advances in Neural Information Processing Systems 6*, 1994, vol. 6, pp. 120–127.

[27] E. Acu, "The treatment of missing values and its effect in the classifier accuracy," *Classification clustering and data mining applications*, no. 1995, pp. 1–9, 2004.

[28] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, "Missing value estimation methods for DNA microarrays.," *Bioinformatics*, vol. 17, no. 6, pp. 520–5, 2001.

[29] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[30] M. Giavalisco, "Lyman-Break Galaxies," *Annual Review of Astronomy and Astrophysics*, vol. 40, no. 1, pp. 579–641, 2002.

[31] J. L. Schafer, *Analysis of Incomplete Multivariate Data*, vol. 11, no. 3. Chapman & Hall, 1997, pp. 164–165.

[32] R. J. A. Little and D. B. Rubin, *Statistical Analysis with Missing Data*, vol. Second. Wiley, 1987, p. 408.

[33] C. Lofi, J. Selke, and W.-T. Balke, "Information Extraction Meets Crowdsourcing: A Promising Couple," *Datenbank-Spektrum*, vol. 12, no. 1, 2012.

[34] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation," *Flinders University Adelaide SIE07001*, 2007.

[35] J. Selke, C. Lofi, and W.-T. Balke, "Pushing the Boundaries of Crowd-Enabled Databases with Query-Driven Schema Expansion," in *38th Int. Conf. on Very Large Data Bases (VLDB)*, 2012, pp. 538–549.