The VLDB Journal (2009) 18:407–427
DOI 10.1007/s00778-008-0127-9

SPECIAL ISSUE PAPER

Anytime measures for top-*k* algorithms on exact and fuzzy data sets

Benjamin Arai · Gautam Das · Dimitrios Gunopulos · Nick Koudas

Received: 28 February 2008 / Revised: 30 September 2008 / Accepted: 1 December 2008 / Published online: 23 January 2009

Abstract Top-*k* queries on large multi-attribute data sets are fundamental operations in information retrieval and ranking applications. In this article, we initiate research on the anytime behavior of top-*k* algorithms on exact and fuzzy data. In particular, given specific top-*k* algorithms (TA and TA-Sorted) we are interested in studying their progress toward identification of the correct result at any point during the algorithms' execution. We adopt a probabilistic approach where we seek to report at any point of operation of the algorithm the confidence that the top-*k* result has been identified. Such a functionality can be a valuable asset when one is interested in reducing the runtime cost of top-*k* computations. We present a thorough experimental evaluation to validate our techniques using both synthetic and real data sets.

Keywords Approximate query · Anytime · Top-*k* · Fuzzy data

1 Introduction

Top-*k* queries on large multi-attribute databases are commonplace. Such queries report the *k* highest ranking results based on similarity scores of attribute values and specific score aggregation functions. Such queries are very frequent in a multitude of applications including (a) multimedia similarity search (on images, audio, etc.), (b) preference queries expressed on attributes of assorted data types, (c) internet searches on scores based on word occurrence statistics and diverse combining functions, and (d) sensor network applications over streams of sensor measurements.

Several algorithms have been introduced in literature to efficiently perform top-*k* computations. Among the most successful is the TA algorithm discovered independently by Fagin et al. [16], Guntzer et al. [20] and Nepal et al. [30]. In this algorithm, each value of an attribute can be accessed independently via an index in descending order of its score. Such a score is computed with a specific query condition. Numerous algorithms for performing top-*k* computations have been proposed [2–4,14,15,18,24,27,29] depending on the model of data access, stopping conditions, etc. The majority of such computations however can be exhaustive. The algorithms come to a stop only when there is absolute certainty that the correct top-*k* result has been identified.

An *anytime* algorithm is an algorithm whose quality of results improves gradually as computation time increases [21]. Although several types of such algorithms have been proposed, *interruptible* anytime algorithms are highly popular and useful. An interruptible anytime algorithm is an algorithm whose runtime is not determined in advance but at any time during execution can be interrupted and return a result. Moreover, interruptible algorithms have an associated *performance profile* which returns result quality (for suitably defined notions of quality) as a function of time (relative to execution) for a problem instance. Such algorithms are valuable since at any point during the execution a user can obtain feedback regarding the result quality at that point. If one is satisfied with the current feedback one may bring the

B. Arai (✉)
University of California, Riverside, USA
e-mail: barai@cs.ucr.edu

G. Das
University of Texas, Arlington, USA
e-mail: gdas@cse.uta.edu

D. Gunopulos
University of Athens, Athens, Greece
e-mail: dg@di.uoa.gr

N. Koudas
University of Toronto, Toronto, Canada
e-mail: koudas@cs.toronto.edu

algorithm to a halt. Thus, such algorithms provide a graceful trade-off between result quality and response time.

In this article, we initiate a study of *anytime top-k* algorithms. We study the behavior of common top-*k* algorithms at any point of their execution and we reason about top-*k* result quality. Notice that this notion of anytime top-*k* computation is significantly different from the notion of approximate top-*k* algorithms previously introduced in the literature [5,13]. Such models aim to relax the control parameters of the computation (e.g., distance) which are difficult to translate into guarantees perceived by a user. The actual behavior of such models remains largely empirical. In contrast, we wish to monitor a top-*k* algorithm at any point in its execution and reason about result quality. For large data collections such an approach can be significantly beneficial as one may decide to terminate the computation early if one is satisfied with the current quality of the results. In particular, we make the following contributions:

- We initiate the study of anytime top-*k* computations. We present a framework, within which at any point in query execution for suitable top-*k* algorithms, we can compute probabilistic estimates of several measures of top-*k* result quality. Such measures include confidence of having the correct top-*k* result, precision of the results assessed with respect to the correct top-*k* results, as well as the difference between the scores of the current top-*k* result and the exact result.
- We investigate the monotonic properties of these anytime measures for various top-*k* algorithms such as TA and TA-Sorted. We show that such measures are monotone for TA, but for a single instance of a top-*k* computation of TA-Sorted, these measures can be non-monotonic, though in *expectation* such measures are monotonic.
- We present algorithmic enhancements to TA and TA-Sorted by which they can provide such anytime guarantees with small runtime overheads during the course of their execution over large data collections.
- We show how our algorithms can be applied to two types of settings, namely (a) when the underlying data and similarity scores are exact, such as those typically present in information retrieval and (b) when the underlying data are fuzzy, such as those commonly present in sensor data collections [12]. To the best of our knowledge this is the first time that top-*k* computations (not just anytime top-*k* computations) on fuzzy data are considered. Our framework can encompass both settings easily.
- We present the results of a thorough experimental evaluation of our algorithms using both exact and fuzzy data, evaluating several measures of result quality and demonstrate the practical utility and scalability of our approach.

This article is organized as follows: Sect. 2 reviews related work. In Sect. 3, we present our overall framework and demonstrate several important properties of anytime top-*k* computations. In Sects. 4 and 5, we present our algorithms and methodology to enable several popular top-*k* algorithms with anytime behavior. In Sect. 6, we present our algorithm for fuzzy data sets. In Sect. 7, we presents the results of our experimental validation of our overall methodology. We conclude in Sect. 8.

2 Related work

The threshold algorithm (TA) constitutes the state of the art for top-*k* computations [16,17,20,30]. Several variants of the basic TA ideas have been considered in various contexts [4,24,29]. The article [27] deals with top-*k* problems on web accessible data sources with limited sorted access. Nearest neighbor type of approaches have been considered in this context as well [9,10,22,35]. It is assumed that sorted lists of the data items by each attribute are available, and TA scans these lists (performing *sorted accesses*) in an interleaved manner, and computes the items with top-*k* scores using monotone score combining functions. The algorithm has to immediately compute the complete score for each item encountered in these lists. In order to do so, however, it conducts *random accesses* to all relevant lists and thus its overhead may be high depending on the application context. For the rest of this article we refer to this algorithm as TA.

Several variants of this basic idea have been proposed. TA-Sorted [16,17,20] can work in environments where random access is not available. It maintains worst and best scores for items based on partially computed total scores; the algorithm compares the worst case score of the *k*-ranked item with the best score of all candidates as a stopping condition. In this algorithm, items are always accessed sequentially. Since expensive random access is avoided, in certain situations the performance may be much better than TA.

Optimization issues for TA algorithms have been considered as well [2,4,27]. The main thrust has been to reduce the number of random accesses when sources vary in several parameters, such as speed, selectivity etc. Several statistical aids have been deployed, such as histograms and probabilistic estimators for the number of random accesses.

Anytime algorithms have found numerous applications in AI and planning contexts [11,21]. The quality of results of an anytime algorithm improves as the computation evolves. At a high level, anytime algorithms can be categorized as being either interruptible or contract. An interruptible algorithm does not have a set running time and can always be interrupted at any time during execution returning a result. The quality of the result can be determined via a performance profile. A contract algorithm has a time deadline as a param-

eter and no assumption about the results can be made before the deadline.

Theobald et al. [34] presented an approach for probabilistic top-k query evaluation. This work is specifically targeted to the TA-Sorted algorithm. The basic idea is, for a newly seen item, to compute the probability with which it may belong to the top-k result. If that probability is below a user supplied threshold the item is discarded from further consideration. This way, possibly fewer items are considered during top-k query evaluation. Moreover, by carefully maintaining bounds for the scores of the most promising (as far as the top-k result is concerned) items that have been encountered the algorithm may probabilistically decide to terminate earlier than the regular TA-Sorted deterministic computation. Empirical evaluation presented in [34] demonstrated that the algorithm performs well in practice.

The work of [34] has some similarity to our work, however, it is not an anytime algorithm. It applies only to the TA-Sorted algorithm, and offers guarantees only at the end of the execution, i.e., when the algorithm runs out of candidates. Further, since it focuses only on eliminating candidates that are partially seen but unlikely to be in the final top-k result, it is not directly applicable to the TA algorithm. In contrast, our work is more general in that we propose anytime enhancements to both TA and TA-Sorted.

Recent work [1,6,23,25,26,28,32,36] on probabilistic ranking of data, is orthogonal to the work presented here. The model assumed in these works is that of incomplete data and the probabilistic framework is based on possible worlds semantics. In contrast, we assume complete information with or without noise and we are interested in assigning guarantees on early stopping of popular top-k algorithms.

3 Framework

3.1 Anytime measures

Our focus in this article is to upgrade top-k algorithms so that they can exhibit *anytime behavior*. This means that at any point during the execution—i.e., before the algorithm has terminated—we wish to be able to (a) reveal the current top-k results calculated thus far, and (b) associate a "guarantee" with our current answers. For example, we may wish to be able to give *probabilistic guarantees*, such as: "With probability p, the current top-k tuples are likely to be the true top-k tuples". Providing such probabilistic guarantees is the most critical aspect of our approach, and much of the remainder of this article is devoted to developing appropriate guarantee measures and efficient techniques by which such measures can be calculated. Our goal is to provide a mechanism to continuously recompute these guarantees as more data is seen.

- *Confidence.* The algorithms shall be able to determine the probability that the current top-k tuples are indeed the true top-k tuples.
- *Precision.* The algorithms shall be able to calculate a (probabilistic) lower bound on the *precision* of the current top-k tuples—i.e., this bound on the precision will hold with a given probability of p (typically, $p = 0.95$). The precision of the retrieved results is defined as r/k where r is the number of the current top-k tuples that belong to the true top-k tuples.
- *Score distance.* Finally, the algorithms shall be able to compute a probabilistic upper bound on the difference between the smallest score of the true top-k tuples relative to the smallest score of the current top-k tuples.

3.2 Knowledge of the data distribution

To be able to give probabilistic guarantees with our anytime answers, it is critical that we assume some knowledge of the data, such as the number of tuples N, as well as knowledge of the distributional properties of the data. Such knowledge can be obtained via popular parametric or non-parametric techniques (i.e., histograms). These data distribution models are assumed to be either available (e.g., histograms of the data have been pre-computed, to be used multiple times for different top-k queries), or can be computed on demand (e.g., for each top-k query, fresh histograms are computed). Our development of anytime top-k algorithms does not depend on the particular type of distributional knowledge assumed. For this reason, we employ a generic probabilistic model of the data which we assume is known to us. We choose to do so in order to keep the presentation of our techniques generic and independent of specific forms of data distribution models.

To be more specific, let our database D have N tuples over M attributes $A_1, \ldots, A_M$ and let $\mathrm{Dom}_1, \ldots, \mathrm{Dom}_M$ be the respective domains of the attributes. The probability distributional model of the data may either be specified (assuming attribute independence) as a product of known probability density functions $gPDF_i(x)$ associated with each ith attribute (e.g., M single-dimensional histograms), or as a joint distributional model over the space of all possible tuples $\mathrm{Dom}_1 \times \cdots \times \mathrm{Dom}_M$ (e.g., a multi-dimensional histogram). Our actual database D may be assumed to be a specific instance of N tuples independently drawn from this distribution.

3.3 Knowledge of error distributions for fuzzy data

As discussed in the Sect. 1, we also consider *fuzzy* data, such as data collected from sensors, business and scientific data that has not yet been cleaned, and so on. In such datasets, the observed values of any tuple may not necessarily be the true original values, but may have been corrupted through various

processes. We can model such fuzzy data by assuming that the value of the ith attribute of a tuple is $t'[i] = t[i] + e$, where $t[i]$ is the true value modeled via a known probability density $gPDF_i(x)$, which is then "perturbed" by an additive error e drawn from a separate *known error distribution* $ErrPDF_i$ over a domain $[-\epsilon_i, +\epsilon_i]$. As a simple example, a uniform error distribution $[-\epsilon_i, +\epsilon_i]$ may be associated with the ith attribute. Another typical error model in certain applications is the Gaussian distribution [7,8,33]. The *mean* values of these error distributions are usually zero, and the *variance* is indicative of how uncertain we are of the final observed data values in the database. In principle, error models could be associated with each generated value—e.g., larger values may be associated with larger errors—and not just with each attribute as we have shown in the above example. While our methods will work for all such error models, for the sake of simplicity of exposition we only describe the case where error models are associated with each attribute.

4 Anytime TA algorithm

4.1 Preliminaries

We begin with a short description of the threshold algorithm (TA): the algorithm proceeds in iterations, where in each iteration, the next items in each sorted list are retrieved in parallel. For each retrieved tuple-id, the entire tuple is retrieved using random access and its score is computed. The algorithm maintains a bounded buffer of size k in which the current top-k tuples (i.e., among those seen) are maintained. The algorithm terminates when a *stopping condition* is reached, i.e., when the minimum score in the top-k buffer (henceforth referred to as $kMinScore$) is larger than $Score(h)$, where $h = [h_1, \ldots, h_M]$ is a "hypothetical" tuple such that each h_i is the last attribute value read along the sorted order for A_i.

Consider a snapshot of TA after d iterations for a specific database D. Let $Seen_d$ be the "prefix" of the database that has been seen by this algorithm after these d iterations. To be able to estimate the anytime measures, the algorithm will have to make some distributional assumptions about the remaining portion of the database that has not yet been seen. Intuitively, the algorithm determines the pdf of the remainder of the database by *conditioning* the data distributional model (discussed in Sect. 3.2) with the prefix already seen, and then computes estimates of each of the anytime measures based on this conditional pdf. As an example, assume that the data distribution of D is defined using the distributions $gPDF_i$ along the ith attribute assuming independence among the attributes, and let $h_1, \ldots, h_M$ be the last values seen along each attribute respectively. Then the ith attribute of any unseen tuple t in the remainder of the database will be a random variable $t[i]$ distributed according $gPDF_i$ *conditioned by* $t[i] \leq h_i$.

Let $PDF(O|O \in \mathcal{O})$ represents the probability density associated with object O that belongs to a (possibly infinite) set $\mathcal{O}$. Thus if $\mathcal{D}$ refers to the space of all database tables with N tuples that can be generated by the probabilistic data model discussed in Sect. 3.2, then $PDF(D|D \in \mathcal{D})$ is the probability density associated with each specific database D.

Let $OneMore(Seen_d)$ refer to the space of all possible valid prefixes of databases that is defined by extending $Seen_d$ by one more iteration. Consider any specific extension of $Seen_d$ by one iteration, say $Seen_{d+1}$. We note that a pdf over this space of extensions, i.e. $PDF(Seen_{d+1} \mid Seen_{d+1} \in OneMore(Seen_d))$, can be naturally defined. To carry $OneMore(Seen_d)$ even further, let $\mathcal{D}(Seen_d)$ refer to the space of all possible valid complete databases that can be defined by extending $Seen_d$ into complete databases, i.e., after $N - d$ iterations. The pdf of these databases, $PDF(D \mid D \in \mathcal{D}(Seen_d))$, can be naturally defined.

Let $Score(t)$ be the score of a tuple t, defined as a linear additive function on the the individual attribute values in typical top-k algorithms, such as $Score(t) = w_1 t[1] + \ldots w_M t[M]$ where the weights are positive constants. Let the $kMinScore(Seen_d)$ refers to the kth largest score of all tuples in $Seen_d$. We can make the following observation:

Observation 1 The minimum score of the current top-k tuples increases monotonically as the algorithm progresses on any database.

$$kMinScore(Seen_d) \leq kMinScore(Seen_{d+1})$$

Let $kthScore(D)$ refer to kth largest score of all tuples in a specific database D. For algorithm Anytime TA let $Confidence(Seen_d)$ be defined as the probability that

$$kMinScore(Seen_d) = kthScore(D)$$

where D is a random valid extension of $Seen_d$ into a complete database drawn from $PDF(D|D \in \mathcal{D}(Seen_d))$.

Theorem 1 *For all database instances it holds that*

$$Confidence(Seen_d) \leq Confidence(Seen_{d+1})$$

Proof Since the $kMinScore(Seen_d)$ is increasing in each iteration, the probability of the $kMinScore(Seen_d)$ being equal to the $kMinScore(D)$ is also always increasing. □

4.2 The algorithm

The anytime version of TA is shown in Algorithm 1. The algorithm proceeds like the standard TA, selecting attributes in a round-robin fashion, and at each step processes the next value in the sorted list of the selected attribute. In addition, it

Algorithm 1 Anytime TA

```
1: topk = {dummy_1, ..., dummy_k} // topk buffer
2: Score(dummy_i) = 0 // Score of tuple dummy_i
3: kMinScore = 0 // smallest score in topk buffer
4: for d = 1 to N do
5:   for all lists L_i (1 ≤ i ≤ M) in parallel do
6:     Let <tuple-id t, t[i] > be the dth item in L_i
7:     // Compute Score(t) using random access
8:     Score(t) = 0
9:     for j = 1 to M do
10:      Score(t)+ = w_j t[j]
11:    end for
12:    //Update PDFs to model the remaining values
13:    Update-gPDF(gPDF_i, t[i])
14:    //Update topk buffer
15:    if Score(t) > kMinScore then
16:      if t ∉ topk then
17:        Let u be the tuple with the smallest score in topk
18:        topk = topk − {u}
19:        topk = topk ∪ {t}
20:      end if
21:      kMinScore = min{Score(v)| v ∈ topk}
22:    end if
23:    // Compute confidence
24:    Confidence = ComputeConfidence()
25:  end for
26: end for
```

also maintains the information necessary to compute probabilistic guarantees.[1]

For each round of the algorithm a new value $\langle t, t[i]\rangle$ is read along the list L_i corresponding to the ith attribute, i.e., the ith attribute value of tuple t. When this item is read, the algorithm has to (a) resolve $Score(t)$ (which is the sum of the attributes of t and is done by probing the lists using random access), (b) update the pdf of the ith attribute ($gPDF_i$) so that it reflects the distribution of the remaining values of that attribute, and (c) update the top-k buffer with the k tuples with the highest scores. At the end of each round the statistics are updated and the confidence is computed.

4.3 Computing anytime TA measures

In this subsection, we discuss details of how the various anytime measures are computed in each iteration of the algorithm. For an unseen tuple t, its score may be viewed as a random variable. Let $scorePDF_t(x)$ be the pdf of the score of t. In order to compute the anytime measures, we need to compute the pdf of the score of any unseen tuple. If we assume attribute independence, then the score of an unseen tuple is the sum of M random variables. To compute the pdf of this sum, we compute the *convolution* of the $gPDF_i$. We show below how this score can be estimated by convolution of pdfs of M independent attributes.

Definition 1 (Convolution of two distributions) Assume that $f(x)$, $g(x)$ are the probability density functions (pdfs) of the two independent random variables X, Y respectively. The pdf of the random variable $X + Y$ (the sum of the two random variables) is the convolution of the two pdfs:

$$*(\{f, g\})(x) = \int_0^x f(z)g(x - z)\mathrm{d}z$$

This definition can be easily extended to the sum of more than two random variables.

Remark The attribute independence assumption is widely used in database literature, and does help simplify our methods and make them computationally tractable, and therefore we adopt it for the most part of this work. Of course, if attributes are not assumed to be independent, then we cannot use convolutions to compute the pdf of the score of the tuple. However, in principle our approach extends to correlated data, and Sect. 4.6 discusses how we can directly compute the score pdf of a tuple from a joint distributional model of the tuple (in this case, a multi-dimensional histogram). Nevertheless, we do emphasize that obtaining accurate correlated models of data is often very difficult in practice, and therefore this component of our work is expected to have limited practical value, except for certain very specialized application domains.

4.3.1 Computing confidence

Let $Seen$ ($Unseen$) refer to the set of tuples that have been seen (unseen) by the algorithm thus far. It is clear that

$$|Unseen| = N - |Seen|.$$

To execute the function call $ComputeConfidence()$, we have to estimate $Prob(kMinScore > MaxUnseen)$, where $kMinScore$ is the minimum score in the top-k buffer, while the random variable $MaxUnseen$ describes the maximum score of all the unseen tuples.

Recall from Sect. 3.2 that our data model assumes that the database is an instance of tuples drawn randomly and independently from the data distribution. Let $OneUnseen$ be the random variable that describes the score of any one unseen tuple. Then we have

$$Prob(kMinScore > MaxUnseen) = Prob(kMinScore > OneUnseen)^{|Unseen|}$$

Let $OneUnseenPDF$ be the pdf of $OneUnseen$. This can be computed by the convolution of the pdfs of the attribute values:

$$OneUnseenPDF = *\{gPDF_i | 1 \le i \le M\}.$$

[1] Note that unlike the standard TA algorithm our algorithm does not have a termination condition, since the objective is to produce anytime probabilistic guarantees. Our algorithm can be easily modified to terminate, for example when the probabilistic guarantees cross a user defined threshold.

Note that $gPDF_i$ here is the updated pdf that reflects the distribution of the remaining unseen tuples along the ith attribute. Once $OneUnseenPDF$ has been computed, the quantity $Prob(kMinScore > OneUnseen)$ (and hence the quantity $Prob(kMinScore > MaxUnseen)$ can be easily computed.

4.3.2 Computing other anytime measures

In this subsection we outline how, in addition to Confidence, the anytime measures of Precision and Score Distance can be computed.

In the case of Precision, we wish to determine (with a given probability p, say 95%) the fraction of the current top-k tuples that will belong to the true top-k tuples of the database. Let the scores of the current $topk$ tuples be $s_1, s_2, \ldots, s_k$ $(= kMinScore)$. Let $Prob_i$ be the probability that s_i is greater than $MaxUnseen$. These $Prob_i$'s can be computed using the same techniques used for computing Confidence above, except that we have to execute it for each s_i rather that just for $kMinScore$. Let i be the largest integer such that $Prob_i \geq p$. The algorithm outputs i/k as Precision. Note that this is a conservative bound on Precision because we only consider *prefixes* of the current top-k to be overlapping with the true top-k, and not any subset.

In order to compute Score Distance, our task is to find a "high probability" upper bound on the smallest score of the true $topk$ tuples. Thus, we wish to find the smallest positive number δ such that

$$Prob(kMinScore + \delta > MaxUnseen) > p$$

where p is a given probability (e.g., 95%). This can be rewritten as

$$Prob(kMinScore + \delta > OneUnseen)^{|Unseen|} > p$$

Once we have computed the pdf of $OneUnseen$, δ can be computed in a variety of ways. The simplest numerical procedure is to discretize the domain of δ, try out each discrete value and take the smallest value that satisfies the above formula. Alternatively, one can do a binary search along the domain of δ to obtain the smallest satisfying value. In Sect. 4.4, we consider using histograms to approximate probability density functions; the resulting discretization of the domain into buckets makes these numerical procedures especially straightforward to implement.

4.4 Approximating PDFs using histograms

We presented our techniques thus far using a generic probabilistic model of data. In this section, we describe the practical realization of our methodologies using a widely adopted model for approximating data distributions (i.e., pdfs), namely histograms. For simplicity of exposition, we adopt equi-width histograms for our discussion, however, the description is applicable to any histogram technique. We note that histograms can approximate arbitrary functions and thus our use of histograms does not place any restrictions or require any assumptions about the underlying distributions that are being approximated.

The following lemmas detail the running time of the basic convolution operation of the algorithm.

Lemma 1 *The convolution of two pdfs that are represented by two b bucket histograms can be computed in $O(b^2)$ time.*

Proof Consider two random variables, A, B in the domain $[0, 1]$ with corresponding pdfs approximated by two histograms with b buckets, H_A and H_B. Assume that the bucket boundaries are the same: $H_A = [0 = A_1, \ldots, A_b = 1]$; if not we can create two equivalent histograms with $2b$ buckets and the same bucket boundaries. Consider the Cartesian product of the two histograms $C_{A,B} = H_A \times H_B$ where $C_{A,B}[i, j] = H_A[i]H_B[j]$ ($H_A[i]$ is the relative count associated with bucket i.) We can approximate the pdf of $A + B$ with a histogram with $2b$ buckets and boundaries $g_0 = 0$, $g_1 = A_1, \ldots, g_b = 1, g_{b+1} = 1 + A_1, \ldots, g_{2b} = 2$. To compute the histogram we have to compute the probability $Prob(g_k \leq A + B < g_{k+1})$ for the buckets of the new histogram, which may be derived as $\sum_{A_l + B_m = g_{k+1}} C_{A,B}[l, m]$ This histogram can subsequently be approximated by a b bucket histogram by merging neighboring pairs of buckets. This procedure gives an $O(b^2)$ algorithm for computing the convolution of the two pdfs. □

As a corollary, for n histograms, we can perform the convolutions in sequence, with a final running time of $O(nb^2)$.

4.5 An example

Table 1 shows the sorted lists for a dataset with 2 attributes and 5 tuples. We have a query for the top-k tuples where k is equal to 2, and the score for a given tuple t is computed as a linear additive function of the individual attributes.

Throughout the example, assume we use equi-width histograms with at most 2 buckets. At the start, the buckets of the histogram H_{A_1} have ranges ([0, 0.5), [0.5, 1.0]) and counts

Table 1 Sorted lists of a sample table with two columns A_1 and A_2, tuples $t_1, \ldots, t_5$, and values for each attribute ranging from 0 to 1

A_1 *id,val*	A_2 *id,val*
t_4:0.9	t_5:0.8
t_2:0.8	t_4:0.7
t_3:0.4	t_2:0.6
t_1:0.3	t_1:0.3
t_5:0.2	t_3:0.2

(3, 2) respectively, while the buckets of the histogram H_{A_2} have ranges ([0, 0.5), [0.5, 1.0]) and counts (2, 3) respectively. Note that each histogram can represent the corresponding $gPDF_i$ by normalizing to relative counts.

Assume a snapshot of the algorithm where the first items of each list has been read, and t_4 and t_5 have been fully resolved and loaded into the top-k buffer. Thus t_4 and t_5 belong to the *Seen* group. Clearly $kMinScore = Score(t_5) = 1.0$ is the lowest score in the top-k buffer.

The remaining tuples t_1, t_2, and t_3 are in the $Unseen$ group. We need to estimate *OneUnseenPDF*, the pdf of the score of any $Unseen$ tuple using the $gPDF_i$s for attributes A_1 and A_2. We have to first update the $gPDF_i$s to model the remaining values for each attribute. Since the top value from each list has been read, consequently the buckets of H_{A_1} will now have counts (3, 1), while the buckets of H_{A_2} will have counts (2, 2). We then normalize each histogram by dividing by the sum for each to get relative counts (3/4, 1/4) and (1/2, 1/2) respectively.

We then compute *OneUnseenPDF* by taking the convolution of $gPDF_1$ and $gPDF_2$ as follows. We first compute the Cartesian product C_{A_1,A_2}. Next, we note that the range of the random variable *OneUnseen* is [0, 2], and thus the final histogram representing *OneUnseenPDF* will have two buckets with ranges ([0, 1), [1, 2]) respectively. The count of the second bucket with range [1, 2] is $Prob(1 \leq A_1 + A_2 \leq 2)$ which can be derived by summing up the values of $C_{A_1,A_2}[0, 1]$, $C_{A_1,A_2}[1, 0]$ and $C_{A_1,A_2}[1, 1]$, which is $(3/4)(1/2) + (1/4)(1/2) + (1/4)(1/2) = 5/8$. Likewise, the count of the first bucket with range [0, 1) is 3/8.

We can then compute the confidence of the current top-k buffer as described in Sect. 4.3.1. We need to compute $Prob(kMinScore > OneUnseen)^{|Unseen|}$. In our case, this is equal to $Prob(1 > OneUnseen)^3$. Now $Prob(1 > OneUnseen)$ can be estimated to be 3/8 from *OneUnseenPDF*, and thus confidence can be estimated to be $(3/8)^3 = 0.053$.

4.6 Considering multidimensional distributions

The pdf of the score of a tuple for a given query depends on the joint distribution of the attributes. Many commercial systems make the attribute value independence assumption, and keep statistics only for individual attributes. In our setting as described above, the independence assumption is similarly assumed when we compute the pdf of the score of a tuple by taking convolutions of the histograms of the different attributes. Although the independence assumption is commonly applied and is well validated in practice for a wide variety of applications, it may produce inaccurate results in some cases (whether the confidence curve using one-dimensional histograms is higher or lower than the confidence curve using two-dimensional histograms depends on whether the independence-based approach is overly optimistic or pessimistic). For such cases, joint distribution models involving multiple attributes may be necessary.

Joint distributions can easily be applied in our framework. Suppose for some tuple t we have three attributes A, B and C which are unknown. Earlier we showed that we can compute the convolution of H_A, H_B, and H_C, but with *multidimensional histograms* we can now compute the convolution of the score pdf of $A + B$ and H_C, where the score pdf of $A + B$ may be directly computed from $H_{A,B}$, the two-dimensional histogram representing the join distribution of attributes A and B.

As in the case of one-dimensional histograms, multidimensional histograms are computed as a pre-processing step. Methods for computing multidimensional histograms have been thoroughly researched [19,31] involving sampling and other efficient approximation techniques. In the evaluation section of this work, we consider two-dimensional histograms. Since the number of possible two-dimensional histograms is quadratic in the number of attributes, we have to decide which pairs to take. We use the following simple heuristic: starting with the set of attributes, we find the most correlated pair of attributes, compute a two-dimensional histogram on these attributes, remove these two attributes, and continue with the remaining set. This approach produces a linear number of histograms, and, since there is no overlap of attributes between different histograms, greatly simplifies the selection of the histograms that have to be used to compute the convolution of a set of attributes.

Recall that for one dimensional histograms (histograms covering a single attribute) every time a new item from the sorted list is read, the corresponding bucket has to be decreased by one. In the case of multidimensional histograms, we similarly decrement the histograms as new items are read. Suppose we have a database with two attributes A and B, two one-dimensional equi-width histograms H_A, H_B, as well as one 10×10 equi-width 2-dimensional histogram $H_{A,B}$. If the first tuple that is completely resolved has the values (0.3, 0.9), we decrement the buckets of the histograms as follows: $H_A[3]$ would be decremented, $H_B[9]$ would be decremented, and $H_{A,B}[3, 9]$ would be decremented. This same technique follows through for higher dimensional histograms and can be performed incrementally.

5 Anytime TA-Sorted algorithm

In this section, we describe how the TA-Sorted algorithm can be extended to compute online probabilistic guarantees. In addition to *Seen* and $Unseen$ tuples, TA-Sorted also maintains tuples in which only some of the attributes have been seen. This is a consequence of the inability of TA-Sorted to perform random access operations.

Algorithm 2 Anytime TA-Sorted

```
1: topk = {dummy_1, ..., dummy_k} // topk buffer
2: MinScore(dummy_i) = 0 // Sum attributes seen for dummy_i
3: Partials = { } // Partially seen tuples not currently in topk
4: kMinScore = 0 // smallest score in topk buffer
5: Assume for all tuples t, obs(t) = {}
6: for d = 1 to N do
7:   for all sorted lists L_i (1 ≤ i ≤ M) in parallel do
8:     Let <tuple-id t, t[i] > be the dth item in L_i
9:     obs(t) = obs(t) ∪ {i}
10:     MinScore(t) = 0
11:     for j ∈ obs(t) do
12:       MinScore(t)+ = w_j t[j]
13:     end for
14:     //Update PDFs by conditioning with remaining values
15:     Update-gPDF(gPDF_i,t[i])
16:     //Update topk buffer
17:     if MinScore(t) > kMinScore then
18:       if t ∉ topk then
19:         Let u be tuple with smallest worst case score in topk
20:         Remove u from topk
21:         if |obs(u)| < M then
22:           Partials = Partial ∪ {u}
23:         end if
24:         topk = topk ∪ {t}
25:       end if
26:       kMinScore = min{MinScore(v)| v ∈ topk}
27:     end if
28:     if |obs(t)| < M and t ∉ topk then
29:       Partials = Partials ∪ {t}
30:     else
31:       Partials = Partials − {t}
32:     end if
33:     // Compute confidence
34:     Confidence = ComputeConfidence()
35:   end for
36: end for
```

Consequently, during the operation of TA-Sorted, we need to keep a set of tuples called $Partials$ that are not in the top-k, yet cannot be eliminated because we know only a lower-bound of their true score. The TA-Sorted algorithm must estimate the pdf of the maximum scores of the $Partials$ before giving any probabilistic guarantee on the confidence.

Like TA, the TA-Sorted algorithm as shown in Algorithm 2 selects attributes in a round-robin fashion, at each step processing the next (sorted by decreasing magnitude) value of the selected attribute. The differentiating factor between Anytime TA and Anytime TA-Sorted is the inclusion of $Partials$. Let $Partials$ be the set of tuples that are partially seen (some but not all of the attributes for a given tuple have been resolved), but are not in the top-k buffer.

Let $\langle t, t[i]\rangle$ be the next item read by the algorithm along the sorted list L_i corresponding to the ith attribute, i.e., the ith attribute value of tuple t. When this item is read, the algorithm has to (a) update $MinScore(t)$ (which is the sum of the attributes that have seen for t) (b) update the pdf of the attribute i ($gPDF_i$), and (c) update the top-k buffer with the k tuples with the highest lower-bound scores. After reading $t[i]$, t will either be fully resolved (that is, all attributes of t have been seen and its final score found) and put in the $Seen$ group, or partially resolved and placed in the $Partials$ group.

5.1 Monotonicity for anytime TA-Sorted measures in expectation

Let $kthScore(D)$ refer to the kth largest score of all tuples in a specific database D. The $Confidence(Seen_d)$ for Anytime TA-Sorted may be defined as the probability that

$$kMinScore(Seen_d) > (k+1)thScore(D)$$

where D is a random valid extension of $Seen_d$ into a complete database drawn from $PDF(D|Din\mathcal{D}(Seen_d))$. Because of the use of lower-bound scores, this definition of confidence is actually even more conservative than the earlier definition of confidence in Sect. 3.1.

Example There exist a database instance where

$$Confidence(Seen_d) > Confidence(Seen_{d+1})$$

Assume a database with two columns A_1 and A_2, each with domain [0.0, 1.0] and a uniform distribution model. Let the score function be $Score(t) = t[1] + t[2]$. Let the database have four tuples with tuple-ids $t1, \ldots, t4$, and assume that the task is to return the top-2 tuples.

In the first iteration, assume we encounter $t1 = [0.9, ?]$ and $t2 = [?, 0.9]$, along each of the sorted lists (a ? implies that the corresponding attribute value is unresolved). After this iteration, the top-2 buffer is loaded with $t1$ and $t2$, each with a worst case score of 0.9. Since we have not seen the other two tuples, we assume that each is distributed uniformly in $[0.0, 0.9] \times [0.0, 0.9]$, and hence the probability that the current worst case score of 0.9 is larger than the scores of both these unseen tuples is $(1/2) * (1/2) = 1/4$.

Suppose in the next iteration the algorithm encounters $t3 = [0.8, ?]$ and $t4 = [?, 0.8]$. After this iteration, the top-2 buffer remains unchanged. However, the unresolved attribute of $t3$ has a probability of $7/8$ of having a value in the range $[0.1, 0.8]$, which would enable $t3$ to have larger score than the current worst case score. A similar argument can be made for $t4$. Thus, the probability that the current worst case score of 0.9 is larger than the scores of both these (now partially seen) tuples decreases to $(1/8) * (1/8) = 1/64$.

Similar examples can be constructed to demonstrate that the other anytime measures are non-monotonic for certain database instances. These arguments bring to light a subtle issue. The uncertain (probabilistic) nature of anytime measures should of course be obvious to the reader—i.e., that at any point during execution, we cannot be completely certain that we have discovered the true top-k tuples, and therefore can only make probabilistic guarantees regarding our

anytime measures. However, what the example shows is that as the iterations progress, we may have to revise, and sometimes *even reduce*, our probabilistic guarantees. We note that a similar argument will not suffice in the case of TA, because in that algorithm a tuple is never in a partially resolved state—it is either completely seen or completely unseen.

However, although the anytime measures for TA-Sorted are not monotonic for certain database instances, we can nevertheless show that the measures are monotonic *in expectation* over all database instances. We describe the result for the confidence measure. Similar results for the other anytime measures are straightforward and omitted due to lack of space.

Let $E[Confidence(Seen_{d+1})]$ be defined as the expected value of $Confidence(Seen_{d+1})$, where $Seen_{d+1}$ is randomly drawn from

$$PDF(Seen_{d+1}|Seen_{d+1} \in OneMore(Seen_d)).$$

Theorem 2 (Expected Monotonicity Theorem)

$$Confidence(Seen_d) \leq E[Confidence(Seen_{d+1})]$$

Proof From the definition of confidence, we know that

$$\begin{aligned} &Confidence(Seen_d) \\ &= \sum_{D \in \mathcal{D}(Seen_d)} (kMinScore(Seen_d) \\ &= kthScore(D)) \cdot Prob(D|D \in \mathcal{D}(Seen_d)) \end{aligned}$$

Partitioning all valid database extensions D as follows, we get

$$\begin{aligned} Confidence(Seen_d) = &\sum_{Seen_{d+1} \in OneMore(Seen_d)} \\ &\times \left(\sum_{D \in \mathcal{D}(Seen_{d+1})} (kMinScore(Seen_d) = kthScore(D)) \right. \\ &\left. \cdot Prob(D|D \in \mathcal{D}(Seen_{d+1})) \right) \\ &Prob(Seen_{d+1}|Seen_{d+1} \in OneMore(Seen_d)) \end{aligned}$$

From Theorem 1 we have

$$kMinScore(Seen_d) \leq kMinScore(Seen_{d+1})$$

for any extension $Seen_{d+1}$. Thus the above reduces to:

$$\begin{aligned} Confidence(Seen_d) \leq &\sum_{Seen_{d+1} \in OneMore(Seen_d)} \\ &\times \left(\sum_{D \in \mathcal{D}(Seen_{d+1})} (kMinScore(Seen_{d+1}) = kthScore(D)) \right. \\ &\left. \cdot Prob(D|D \in \mathcal{D}(Seen_{d+1})) \right) \\ &\cdot Prob(Seen_{d+1}|Seen_{d+1} \in OneMore(Seen_d)) \end{aligned}$$

Thus,

$$\begin{aligned} &Confidence(Seen_d) \\ &\leq \sum_{Seen_{d+1} \in OneMore(Seen_d)} Confidence(Seen_{d+1}) \\ &\cdot Prob(Seen_{d+1}|Seen_{d+1} \in OneMore(Seen_d)) \end{aligned}$$

Thus,

$$Confidence(Seen_d) \leq E[Confidence(Seen_{d+1})].$$

□

5.2 Computing anytime TA-Sorted measures

In this subsection, we discuss how the anytime measures are computed in each iteration of the TA-Sorted algorithm.

5.2.1 Computing confidence

At any instance during the execution of the algorithm, consider the set of tuples $Others = Partials \cup Unseen$. Let $MaxOthers$ be the random variable that denotes the maximum score of all tuples in $Others$. To execute the function $ComputeConfidence()$, we have to estimate

$$Prob(kMinScore > MaxOthers)$$

Let us define two random variables $MaxPartials$ and $MaxUnseen$ that denote the maximum score of all tuples in $Partials$ and $Unseen$ respectively. Thus we have

$$\begin{aligned} &Prob(kMinScore > MaxOthers) \\ &= Prob(kMinScore > MaxUnseen) \\ &\cdot Prob(kMinScore > MaxPartials) \end{aligned}$$

The first factor can be written as

$$\begin{aligned} &Prob(kMinScore > MaxUnseen) \\ &= Prob(kMinScore > OneUnseen)^{|Unseen|} \end{aligned}$$

This can be computed using techniques similar to that developed earlier in Sect. 4.3.1. We thus focus on the computation of the second factor. For any tuple t in $Partials$, since all attributes have not been resolved, $Score(t)$ is a random variable. The second factor can thus be written as

$$\begin{aligned} &Prob(kMinScore > MaxPartials) \\ &= \prod_{t \in Partials} Prob(kMinScore > Score(t)) \end{aligned}$$

For any tuple t in $Partials$, let $ScorePDF_t$ be the probability distribution of the random variable $Score(t)$. The definition is similar to the definition of the score pdf of an unseen tuple (i.e., $OneUnseenPDF$), except that the convolutions are taken only over the pdfs of the unresolved attributes of t, to which the aggregate of the resolved attribute values (i.e.,

$MinScore(t)$) is combined. More formally, given a real number a, let $vPDF_a(x)$ denote the "value distribution" where all the probability mass is concentrated at value a and is 0 elsewhere.[2] Then

$$ScorePDF_t = *(\{vPDF_{MinScore(t)}\} \cup \{gPDF_i | i \notin obs(t)\})$$

Once $ScorePDF_t$ has been computed, the second factor $Prob(kMinScore > MaxPartials)$ can be computed in a straightforward manner in time linear in the number of partially seen tuples. Since this can become slow for large data sets, in the following subsection we present an efficient implementation that is based on clustering partially seen tuples.

5.2.2 *Efficiently computing the second factor*: $Prob(kMinScore > MaxPartials)$

The straightforward way to compute $Prob(kMinScore > MaxPartials)$ is to compute the score pdfs of each of the partially seen tuples. This linear approach can be slow for large datasets. To improve the running time, we employ a technique of clustering the partially seen tuples.

We first describe the high-level idea. Let us assume that $Partials$ has been partitioned into a small number of clusters, $Partials_1, Partials_2, \ldots, Partials_r$, such that the scores of all tuples within each partition have very similar probability distributions. Each partition $Partials_i$ is also associated with an "upper-bound" distribution $UpperPDF_i$ that has the following properties: Let max_i be the random variable defined as $\max\{Score(t) | t \in Partials_i\}$, and let u_i be a random variable with distribution $UpperPDF_i$; then for all constant c we have $Prob(c > max_i) \geq Prob(c > u_i)$. Such an upper-bound distribution is very useful since it can be used to efficiently compute a *lower bound* for the second factor in the confidence calculations, as shown below. Note that if the upper-bound distribution for each partition is available, computing the last expression takes time linear only in the number of clusters, and not in the size of $Partials$.

$$\begin{aligned} &Prob(kMinScore > MaxPartials) \\ &= \prod_{t \in Partials} Prob(kMinScore > Score(t)) \\ &= \prod_{1 \leq i \leq r} \prod_{t \in Partials_i} Prob(kMinScore > Score(t)) \\ &\geq \prod_{1 \leq i \leq r} Prob(kMinScore > u_i)^{|Partials_i|} \end{aligned}$$

We now provide the formal details of how exactly are these partitions and their corresponding upper-bound distributions defined, computed, and maintained. We first describe a coarse partitioning of $Partials$, and then describe how it can be further refined into an even finer partitioning.

For all subsets S of the attributes, let $Partials_S$ be the set of tuples that have exactly these S attributes resolved. That is, $Partials_S = \{t | obs(t) = S\}$. This way we essentially partition all tuples in $Partials$ into at most 2^M clusters (where recall that M is the total number of attributes). Then, let us consider the worst case scores ($MinScore(t)$) of the tuples t in $Partials_S$, and consider an equi-width B-bucket histogram H with these values (where B may be different from the b used to denote the number of buckets in the score/attribute histograms). The tuples in $Partials_S$ may be further partitioned into B clusters, where the cluster $Partials_{S,j}$ represent all tuples t of $Partials_S$ such that $MinScore(t)$ falls within the jth bucket of H. Note that $Partials$ has now been partitioned into at most $2^M B$ clusters. Moreover, any two tuples in $Partials_{S,j}$ have the same set of resolved attributes and approximately the same worst case scores.

We next define $UpperPDF_{S,j}$, the upper-bound distribution for any cluster $Partials_{S,j}$. Let the bucket boundaries of H be $h_0, h_1, \ldots, h_B$. We then have

$$UpperPDF_{S,j} = *(\{vPDF_{h_j}\} \cup \{gPDF_i | i \notin S\})$$

Thus, using the above clusters as well as their corresponding upper-bound distributions, we can efficiently compute a lower bound to $Prob(kMinScore > MaxPartials)$ in time proportional to the number of clusters, i.e., $O(2^M B)$.

We can efficiently maintain the clusters as well as their upper bound distributions as follows. We maintain one counter for each of the $2^M B$ histogram buckets (which are in the beginning initialized at 0). Every time a new value is read in, one of the tuples has one more attribute resolved. If this is a new tuple, we increment the corresponding bucket and add this tuple to the $Partials$ set. If the tuple is already in $Partials$, one bucket will have its counter reduced by one. If the tuple is still not fully resolved, another bucket will have its counter increased by one.

We note that the running time of this update is independent of N, the total number of tuples in the database.

5.2.3 *Computing other anytime measures*

In addition to Confidence, the anytime measures of Precision and Score Distance can also be computed. The proposed techniques are very similar to those described in Sect. 4.3.2 for the Anytime TA algorithm, except that instead of $MaxUnseen$ we use $MaxOthers$, and $kMinScore$ is the smallest worst case score of the current $topk$ tuples. The straightforward details are omitted.

[2] The value distribution is identical to the *Dirac's delta distribution* in physics, or the *impulse function* in signal processing.

6 Considering fuzzy data

In this section, we investigate how to extend our top-k results for fuzzy data. Fuzzy data is typically associated with sensors based applications—e.g., for an environment monitoring application consider the problem of determining the top-k time instances during a given day when the average temperature reading of all sensors is the highest. As defined in Sect. 3.3, our model for fuzzy data is that each value observed by any algorithm is its true value plus an error, the latter modeled via an error distribution. We assume that the error distributions (e.g., those associated with the sensors) are known and is the same for all the values of a given attribute i. Thus, if we are only given the observable attribute values of a tuple t, we can only determine the true score of t using a probability density function. Thus the challenge is to develop algorithms that can retrieve the true top-k tuples with some degree of confidence.

6.1 Impact of fuzzy data on standard top-k algorithms

Before we even start considering how to extend our anytime algorithms to handle fuzzy data, we consider two basic yet compelling problems.

The first problem that we consider is: how can we modify standard top-k algorithms so that they terminate with a 100% confidence of having retrieved the top-k tuples? Recall that we assume that the error along each attribute is distributed within the interval $[-\epsilon_i, +\epsilon_i]$. This enables us to employ standard TA-Sorted algorithmic techniques to possibly achieve a 100% guarantee: subtract (resp. add) ϵ_i to each attribute value of a partially resolved tuple to compute a deterministic lower (resp. upper) bound of its score, and terminate when the kth smallest lower bound is larger than the upper bound of the other tuples. We note that although such techniques may enable early termination to guarantee that a superset of the top-k tuples have been encountered, it still may not be possible to pinpoint which of the encountered tuples are exactly the true top-k. Moreover, if the error distribution is *unbounded*, such as a Gaussian distribution, then we can never achieve 100% certainty that the true top-k tuples have even been encountered, let alone pinpointed, unless we complete a full scan of all lists.

The second problem that we consider is: what happens if we simply execute the standard top-k algorithms TA and TA-Sorted as-is on fuzzy data, without making any modifications to make them anytime algorithms? It is clear that if we were to execute these standard algorithms, due to the fuzziness of the data, there is some chance that even after termination the true top-k tuples would not have been discovered. However, these algorithms can be modified such that after termination they are able to compute a *terminal confidence*, i.e., a probabilistic guarantee that they have indeed computed the true top-k tuples.

The terminal confidence of standard top-k algorithms is *exactly the same* as the value of the confidence that our corresponding anytime algorithms (to be described in the next subsection) would arrive at the moment they completed an identical number of iterations into the lists. Thus, instead of continuously computing the confidence that our anytime algorithms for fuzzy data are required to do, all we need to do is to defer the confidence computation till after the algorithm has terminated. Overall, this will be more efficient than the anytime algorithms, as we can avoid continuous update computations (such as updates to the $gPDF_i$ distributions after each iteration). However, the value of the resulting confidence will be the same.

6.2 Anytime algorithms for fuzzy data

Here we describe the extensions necessary to the anytime algorithms presented in the previous sections to work for fuzzy data. Essentially, these algorithms must be modified to compute the pdf of $Score(t)$, even when all the values of the attributes have been observed. If $obs(t)$ is the set of the attributes of t that have been observed, then the pdf of the $Score(t)$ is defined by Definition 2 (where recall that $vPDF_a$ refers to the pdf where all the probability mass is concentrated at value a, and $ErrPDF_i$ is the error distribution of the ith attribute.

Definition 2 The pdf of $Score(t)$ is computed as

$$\begin{aligned} &ScorePDF_t \\ &= *(\{vPDF_{t[i]} | i \in obs(t)\} \cup \{ErrPDF_i | i \in obs(t)\} \\ &\quad \cup \{gPDF_i | i \notin obs(t)\}) \cup \{ErrPDF_i | i \notin obs(t)\}) \end{aligned}$$

Definition 2 can then be used for estimating the probabilistic measures the algorithms compute online. However, to decide which tuples should be in the top-k buffer, the algorithms have to find the tuples with the highest worst case scores. Consequently, for the computation of Definition 3, we use the ϵ_i bound on the maximum error, and we compute the worst score of a tuple t by replacing the observed values $t[i]$ with the worst case bounds $t[i] - \epsilon_i$.

Definition 3 The worst case score is defined to be $MinScore(t) = \sum_{i \in obs(t)} w_i(t[i] - \epsilon_i)$.

6.3 The algorithm

Our discussion mainly focuses on the anytime version of TA for fuzzy data (shown as Algorithm 3); we only make brief mentions of the anytime version of TA-Sorted whenever appropriate. The algorithm proceeds like Anytime TA

Algorithm 3 Anytime TA for Fuzzy Data

```
1: topk = {dummy_1, ..., dummy_k} // topk buffer
2: Score(dummy_i) = 0 // Score of the current tuple dummy_i
3: kMinScore = 0 // smallest score in topk buffer
4: for d = 1 to N do
5:   for all lists L_i (1 ≤ i ≤ M) in parallel do
6:     Let <tuple-id t, t[i] > be the dth item in L_i
7:     // Compute MinScore(t) using random access
8:     MinScore(t) = 0
9:     for j = 1 to M do
10:      MinScore(t)+ = w_j(t[j] − ε_j)
11:    end for
12:    //Update PDFs to model the remaining values
13:    Update-gPDF(gPDF_i, t[i])
14:    //Update topk buffer
15:    if MinScore(t) > kMinScore then
16:      if t ∉ topk then
17:        Let u be the tuple with the smallest score in topk
18:        topk = topk − {u}
19:        topk = topk ∪ {t}
20:      end if
21:      kMinScore = min{MinScore(v)| v ∈ topk}
22:    end if
23:    // Compute confidence
24:    Confidence = ComputeConfidence()
25:  end for
26: end for
```

described earlier in Algorithm 1, maintaining the information necessary for computing probabilistic guarantees.

The main changes that we must make are: (a) in computing the worst case score of any tuple, we must subtract the maximum error associated with each observed attribute of the tuple, and (b) in computing the pdf of any tuple, we must make sure that the error distributions along each attribute are taken into consideration.

For each round of the algorithm a new value $\langle t, t[i]\rangle$ is read along the list L_i corresponding to the ith attribute. The algorithm has to (a) resolve $MinScore(t)$ (which is achieved by probing the lists using random access and subtracting the max error from each attribute value); (b) update the pdf of the ith attribute ($gPDF(i)$) such that it reflects the distribution of the unseen values for that attribute; and (c) update the top-k buffer with the k tuples with the highest worst case scores. At the end of each round the statistics are updated and the confidence is computed. We discuss the confidence computation next.

6.4 Computing confidence

In this subsection, we will describe how to compute the confidence that the current top-k buffer is indeed the actual top-k tuples in the database. To execute the function call $ComputeConfidence()$ for fuzzy data, we have to estimate $Prob(kMinScore > MaxUnseen)$ (equivalently, $Prob(kMinScore > MaxOther)$ in the case of TA-Sorted), where $kMinScore$ is the minimum worst case score in the top-k buffer, and the random variable $MaxUnseen$ (respectively $MaxOther$) describes the maximum score of all remaining tuples. To accomplish this we need to be able to compute the pdf of any of the tuples that are not in the top-k buffer. Unlike the case of non-fuzzy data, the computation of this pdf requires us to first compute the distribution of the tuple's actual values along each attribute, and then employ convolutions as suggested in Definition 2, including convolution of the error distributions. The rest of the steps are similar to that used in the corresponding algorithms (both anytime TA and anytime TA-Sorted) for non-fuzzy data.

The computation of other anytime measures such as precision and Score Distance can be accomplished using techniques similar to that described in Sect. 4.3.2. We omit the straightforward details.

7 Experimental evaluation

In this section, we present an experimental evaluation of our framework. The implementation of our techniques is in C++ and our evaluations are performed on a dual AMD Opteron 280 processor system with 8GB of memory.

We have conducted series of experiments using synthetic and two real-world data sets varying the distribution and size. The data sets range in size from 4,990 to 1,000,000 rows, and four to ten attributes (we vary the number of attributes when we report on performance). Our experiments focus on the comparison of the accuracy of our estimated results with the expected performance of the TA and TA-Sorted algorithms.

We used both exact and fuzzy datasets in our evaluation. The fuzzy data points are derived using an exact data set by distorting each exact value with samples from Gaussian and Uniform distributions (we evaluate our approach testing several levels of fuzziness for both the Uniform and Gaussian error models).

7.1 Real world data sets

In our experiments, we use two real-world data sets. Our first data set is atmospheric data collected from several independent sensor locations in Washington and Oregon by the Department of Atmospheric Science at the University of Washington. The second is the Internet Movie Database IMDB.[3]

For the sensor data, 25 sensors independently obtained temperature readings on an hourly basis between June 2003 and June 2004, for a total of 208 days. For each sensor there is a total of 4,990 readings.

It is easy to see potential scenarios for data generated from sensor networks utilizing top-k algorithms. One of the most

[3] http://www.imdb.org.

apparent scenarios include the ability to monitor air pollution or other weather-related features across large geographical areas. In our setting, top-k algorithms are utilized to answer relevant queries about temperature levels over extended periods to locate instances of high average temperature.

Each of the readings taken from a sensor were combined with readings from other sensors which had taken a reading during the same time period. These readings were grouped to make individual rows based on their time-stamps. Sensor data such as the temperature data provided can specifically benefit from our algorithms due to the *anytime* behavior. For our experiments we use the readings from five to ten randomly selected sensors.

The IMDB database is composed of more than 860,000 titles and details about each. For the IMDB data set, we extracted a list totaling 863,049 titles.

It is difficult to quantify the performance of a movie looking from only a single perspective. In our setting, we have chosen several attributes to represent different facets of movie performance.

For each title, we queried the following attributes: budget, gross income, opening weekend gross income, and number of keywords describing the title. We chose these attributes because each offers some insight regarding the popularity of the movie from multiple perspectives including: producers, critics, and fans. Specifically, the budget can be seen as a valuation of the film from the producers standpoint, the gross income and weekend gross income translates into critic and fan popularity, and the number of keywords represent information regarding fan popularity (i.e., popular movies are likely to have a larger number of keywords submitted from fans than less popular movies). This allows us to perform queries across a broad set of perspectives for measuring top performing movies.

We experimented with several different histogram sizes; we found that the accuracy did not improve much with histograms of more than 20 buckets for our real-world experiments.

7.2 Anytime measures

Our experimental evaluation validates our measures on real-world and synthetic data sets. As a baseline we compare our approach against the actual confidence, TA, and TA-Sorted algorithms.

In the case when the distribution of scores is skewed, the confidence of the algorithm may stay relatively low for a large portion of the data set. This is due to a high density of values keeping the $kMinScore$ and $MaxOthers$ close for a larger portion of the running time (i.e., there is a low-sloping increase in the confidence, but eventually it reaches 100% confidence). In cases when the distribution of the data set contains a distinct cluster of K or more high scores (row-level correlation) the confidence quickly climbs. For the IMDB data set, there are few large values with the majority of the scores being clustered toward the lower end of the value range for each attribute. This is reasonable considering that there are only a few big budget movies and of these movies an even smaller subset that gross a large sum of money. This creates a data set with a small number of high score tuples. Similarly in the case of the sensor data there is row-level correlation around temperature spikes with the majority of the readings being located around the average temperature for each sensor. As shown in Figs. 1 and 9, both the IMDB and sensor data sets illustrate how correlation of attributes can quickly cause the Anytime TA algorithm to climb to 100% confidence, this can be accounted for by the fact that the correlation of data cause the the $kMinScore$ and $MaxOthers$ groups to quickly diverge.

Accuracy: Our results show good performance for both real-world and synthetic data sets. In Figs. 1 and 2, we show

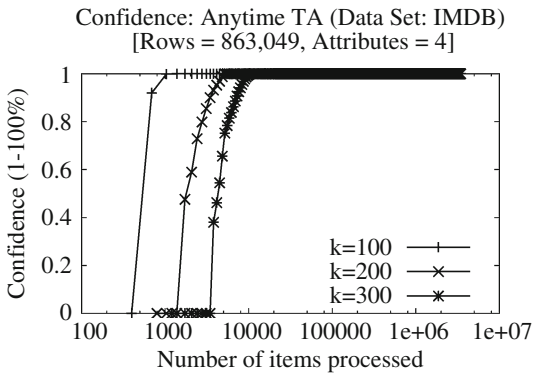

Fig. 1 In this experiment, we evaluate the confidence for varying k as the number of seen tuples is increased for the IMDB data set

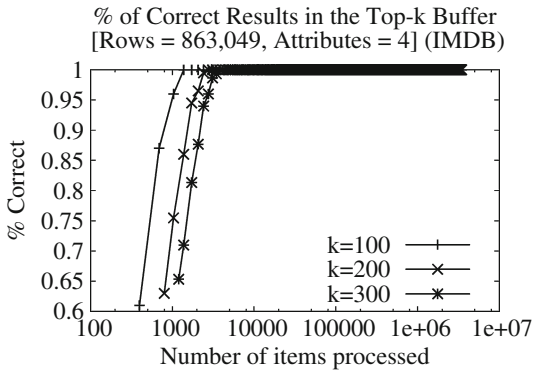

Fig. 2 In this experiment, we show the precision, defined as the percentage of the current top-k buffer that is actually in the top-k result for the IMDB dat set

the confidence and percentage of correct results in the top-k buffer during the execution of the algorithm. These figures illustrate how our estimates coincide with the number of correct results in the top-k buffer.

Further, in Fig. 3 we show that our estimates for the confidence accurately approximates the actual confidence. In order to compare the accuracy of our estimations, we computed the actual confidence by running the TA algorithm for 10 independent runs (we generated 10 randomly distributed synthetic data sets and ran the algorithm for each) building a vector for each run where each element of the vector contains one of two values (1 = "Top-k found", 0 = "Top-k not found yet"). We then computed the average over all runs (i.e., we built a new vector that represents the element-wise average of the vector set) creating a new vector of real values where each element of the vector represents the actual confidence for each respective run.

We evaluate the accuracy of readings by comparing the number of items read given a user-defined confidence using Anytime TA with the number of items retrieved had the actual confidence (defined above) been known. We can estimate the accuracy of a reading by comparing the number of items read for Anytime TA and the actual confidence. In Fig. 3, we shown the error percentage for confidence levels of 0.80 through 0.95. Our algorithm performs well for various levels of confidence. Notice that the Anytime algorithms can either underestimate or overestimate the confidence (viz., in Fig. 3: positive error = underestimate & negative error = overestimate). It just so happens in that particular experiment the algorithm never underestimated.

The results suggest that there is little correlation between the confidence level and the accuracy of our results. For the experiment presented in Fig. 3, the number of items read by the Anytime TA algorithm never deviates more than 16% from the number of items read for the corresponding actual confidence.

7.3 Run-time overhead and performance

Efficiency: Our results show that sizable savings can be achieved in comparison to the TA and TA-Sorted algorithms. As a baseline we ran TA and TA-Sorted on the IMDB and sensor data sets. In each case, we computed how many tuples were read before the TA or TA-Sorted stopping condition was reached. We then compared these results with our algorithm. As shown in Fig. 4 Anytime TA provides sizable savings over TA. We achieve a saving of over 70% (1,200 tuples) for a confidence level of 99% using the IMDB data set. Similarly, Anytime TA works well for high dimensional (sensor) data sets. As shown in Fig. 5, we achieve savings of over 50% (3,000 tuples) for a confidence level of 99% using the sensor data set. Since TA-Sorted does not allow for random accesses, the number of tuples read is usually much greater than TA (allowing for greater savings). As shown in Fig. 6, we compare the Anytime TA-Sorted algorithm with TA-Sorted. In this case, for TA-Sorted and a confidence level of 99% we achieve an even greater savings of over 95% (14,000 tuples).

Run-time Overhead: To evaluate the overhead of our approach we ran experiments with a synthetic data set totaling 1,000,000 rows and 4 attributes. We used histograms of 5–25 buckets to describe attribute distributions. In this set of experiments, we set $K = 1{,}000$, but similar results were obtained for different values.

Table 2 shows the run-time performance of the Anytime TA algorithm, as well as the overhead that the technique imposes over the TA algorithm. In the first column, we report the running time of the TA algorithm. In the second column, we report the running time overhead of our implementation of the Anytime TA algorithm—i.e., the running time had Anytime TA processed the same number of tuples as TA.

For example, if TA takes 100 s to process the necessary tuples to compute the top-k and equivalently it takes Anytime

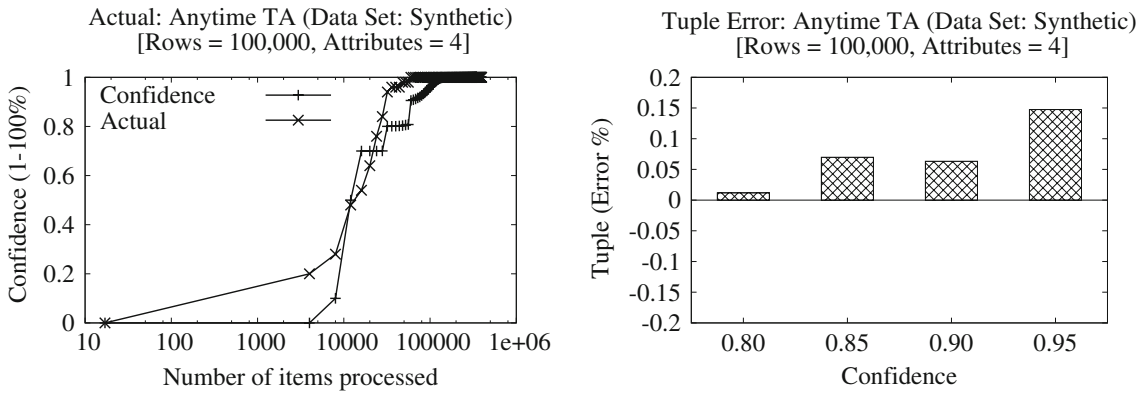

Fig. 3 In this experiment, we compare the actual and Anytime TA confidence. The two figures show the difference in the number of items read for various levels of confidence using the synthetic data set

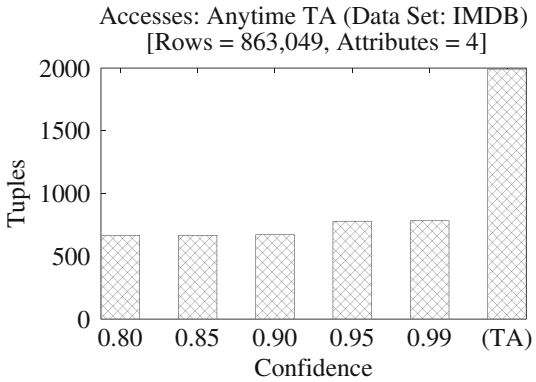

Fig. 4 In this experiment, we compare the number of tuples retrieved for Anytime TA with various levels of confidence using the IMDB data set where $K = 100$

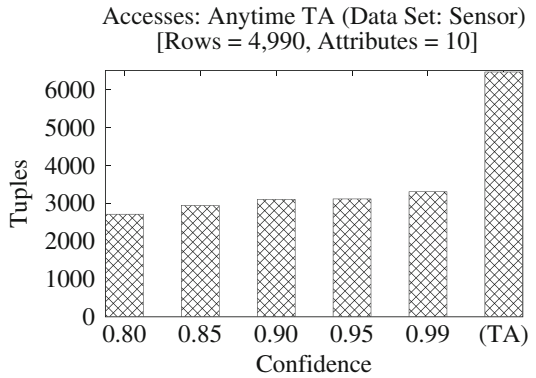

Fig. 5 In this experiment, we compare the number of tuples retrieved for Anytime TA with various levels of confidence using the sensor data set where $K = 300$

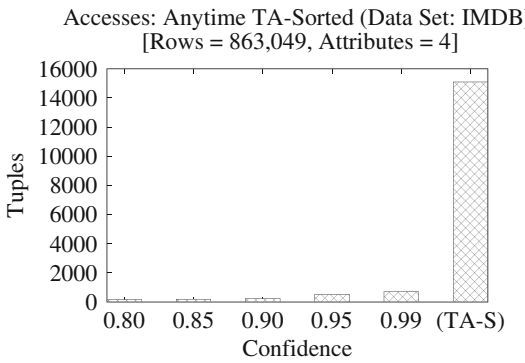

Fig. 6 In this experiment, we compare number of tuples retrieved for Anytime TA-Sorted with various levels of confidence using the IMDB data set where $K = 100$. (TA-S) = TA-Sorted

TA 110 seconds to locate the top-k with 100% confidence (i.e., read the same number of tuples as TA), then TA would be faster. However, if we only needed the top-k with 95% confidence then Anytime TA may terminate after only 90 s while TA would still require the full 100 s.

Table 2 Run-time performance for synthetic data set comparing Anytime TA, TA and time required to take an Anytime TA measure

TA	Anytime TA	Estimation time average time per readings	Histogram size
1.0908	1.1238	0.0001	5
	1.1598	0.0004	10
	1.1778	0.0009	15
	1.2068	0.0014	20
	1.2107	0.0020	25

Synthetic data set (1,000,000 tuples, 4 attributes, histograms size 20, random distribution). Time is reported in seconds

Table 3 Run-time performance for synthetic data set comparing Anytime TA-Sorted, TA-Sorted and time required to take an Anytime measure

TA-Sorted	Anytime TA-sorted	Estimation time average time per readings	Histogram size
2.7696	20.8258	0.0001	5
	26.2369	0.0004	10
	32.3550	0.0007	15
	41.4386	0.0013	20
	51.6661	0.0019	25

Synthetic data set (1,000,000 tuples, 4 attributes, histograms size 20, random distribution). Time is reported in seconds

The second column does not include the time that it takes to compute the anytime measures. In other words, column two only includes the time it takes to run TA and the time it takes to maintain the *gPDF*s for each round. Note that this time is dependent upon the users confidence bound. The third column shows the average time for computing the anytime measure (confidence, precision, and so on) every time this computation is invoked. The total running time of our algorithm is the fraction of the time it takes to run Anytime TA (column 2) and the time it takes to compute the anytime measures (column 3) times the number of times the anytime computation is invoked.

The experimental results in Table 2 suggest that the overhead of our approach is relatively small for Anytime TA. There is little variation in run-time between the TA and Anytime TA algorithm (this is attributed to the fact that histograms are not utilized for computation until a reading is taken). Varying the histogram size between 5 and 25 buckets make little difference in effecting the run-time of the Anytime TA algorithm.

For the Anytime TA-Sorted algorithm as shown in Table 3 there is a sizable difference in the running time for TA-Sorted and Anytime TA-Sorted algorithms. This is attributed to the overhead incurred from the maintenance of the partially seen tuples. In other words, this includes the time it takes to run

Table 4 Run-time performance comparing TA and Anytime TA for varying confidence levels

Confidence	TA tuples	TA time	Anytime TA tuples	Anytime TA time
0.85	1,992	0.1010	700	0.0386
0.90			700	0.0407
0.95			800	0.0441

IMDB data set (863,049 tuples, 4 attributes, histograms size 20, skip size = 100). Time is reported in seconds

Table 5 Run-time performance comparing TA-Sorted and Anytime TA-Sorted for varying confidence levels

Confidence	TA-S tuples	TA-S time	Anytime TA-S tuples	Anytime TA-S time
0.85	15,094	0.1920	200	0.0106
0.90			300	0.0128
0.95			600	0.0247

IMDB data set (863,049 tuples, 4 attributes, histograms size 20, skip size = 100). Time is reported in seconds

Anytime TA-Sorted, update the *gPDF*s and maintain partially seen clusters for each round as defined in Sect. 5.2.2. Varying the histogram size between 5 and 25 buckets make little difference in effecting the run-time of the Anytime TA-Sorted algorithm. Overall, the overhead for the partials remains a fixed cost over Anytime TA and increases when the size of the histograms increases, as expected.

Performance: We evaluate performance in terms of how many tuples we read, and how long it takes to run the algorithm using our implementation. We compare Anytime TA with TA. To evaluate our approach we ran experiments using the IMDB data set totaling 863,049 rows, 4 attributes; we use a histogram size of 20 to describe the distribution. In this set of experiments, we set $K = 100$, but similar results were obtained for different values. Proper selection of skip size (i.e. the number of tuples sampled between readings) can greatly affect the run-time and total number of tuples sampled. A large skip size ensures that the number of readings is minimal. If the skip size is too large then there is a coarsening of the confidence levels between readings, generally causing additional tuples to be read from the database. On the other hand, if the skip size is small then fewer tuples may be sampled but the run-time will increase due to the inflation of reading overhead.

Tables 4 and 5 offer a comparison of run-time performance for Anytime TA, Anytime TA-Sorted and TA for several confidence levels. For each confidence level we report both the run-time and number of tuples retrieved for each algorithm. Anytime TA and Anytime TA-Sorted complete in about the same amount of time. The experimental results

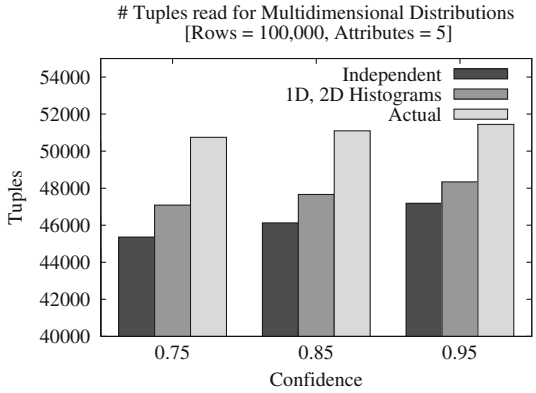

Fig. 7 In this experiment, we compare the number of tuples retrieved from the database using one- and two-dimensional histograms with the actual confidence using a synthetic data set where $K = 100$

in Tables 4 and 5 show that sizable gains can be achieved over TA and TA-Sorted for both run-time and the number of tuples read from the database. However, due to the non-monotonic properties of the Anytime TA-Sorted confidence function, in analyzing the results it is apparent that the Anytime TA-Sorted algorithm terminated at a local maximum. For our experiments we achieved a reduction of approximately 1,100–1,200 tuples and over 14,000 tuples for Anytime TA and Anytime TA-Sorted, respectively. Overall, we have found that our approach works well in a variety of settings.

7.4 Multidimensional histograms

We consider the effects of joint distributions (multidimensional histograms) by comparing the accuracy and performance of our algorithm using one- and two-dimensional histograms. Like the one-dimensional *gPDF*s we have used thus far, we assume that multidimensional histograms are provided as a pre-processing step. We want to compare the accuracy of our results using various levels of knowledge about the scores in the database. For the experiments using joint distributions we assume all combinations of two attribute joint distributions $gPDF_i$s are available as described in Sect. 4.6.

As shown in Fig. 7, we compared the performance of one- and two-dimensional histograms with the actual confidence. Similar to the process described in Sect. 7.2, we computed the actual confidence to compare the number of tuples retrieved assuming independence versus using joint histograms.

The inclusion of multidimensional histograms did not greatly affect the number of tuples read from the database. We experimented with confidence levels (75, 85, and 95%) and in each case we compared the number of tuples retrieved for the Anytime TA algorithm using independent and joint distributions. In each of the trials, using joint distributions

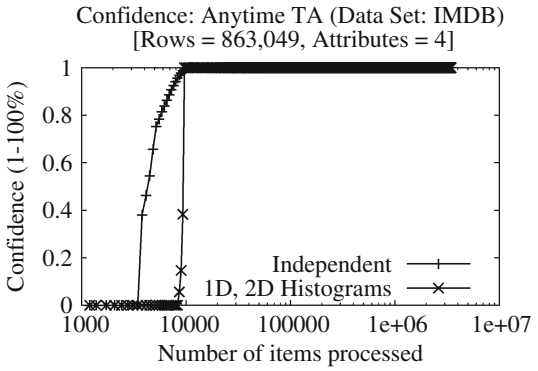

Fig. 8 In this experiment, we compare the confidence for one- and two-dimensional histograms using the IMDB data set where $K = 300$

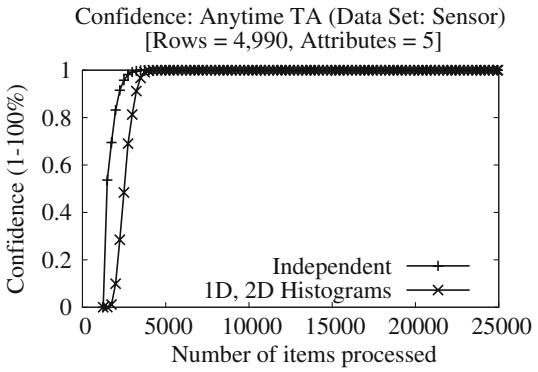

Fig. 9 In this experiment, we compare the confidence for one- and two-dimensional histograms using the sensor data set where $K = 100$

our results were consistently closer to the actual confidence than when assuming independence.

In addition, in Figs. 8 and 9, we show how the inclusion of multidimensional histograms can significantly effect the performance of the algorithm. This is shown in the (1D,2D Histogram) results by the increase in slope for the confidence. This is expected since joint distributions offer more information pertaining to the scores of the unseen tuples. Specifically, given that multidimensional histograms contain additional information about the underlying data, it is easy to see that confidence remains low because the Anytime algorithm knows that it is not the top-k. Further, multidimensional histograms offer additional insight as to how close the algorithm is to the actual top-k. Consider the scenario where we have a perfect understanding of the data, this would lead to a situation where we stay at 0% confidence as we see data, and at the point we find the true top-k the confidence should spike to 100%. Therefore, as we introduce multidimensional histograms (i.e., more information) the model should look less-and-less like a gradual transition from 0 to 100%.

Accuracy: As shown in Figs. 8 and 9 as the dimensionality of the *gPDF*s increases, the confidence measure for both the sensor and the IMDB data sets become increasingly accurate as expected. For the IMDB and sensor data sets there is a strong correlation among the high score values. This correlation is not detected well assuming independence as illustrated in the quick rise in confidence. In contrast, the two-dimensional histograms can better predict high score values for unseen tuples. As shown in Figs. 8 and 9, the confidence stays low until a sufficient number of high value scores have been seen by the algorithm.

Table 6 Run-time performance for anytime TA using one- and two-dimensional histograms

Histogram size	One-dimensional histograms	Two-dimensional histograms
5	1.1238	11.5802
10	1.1598	11.6392
15	1.2068	11.5983
20	1.1778	11.7991
25	1.2107	11.7562

Synthetic data set (1,000,000 tuples, 4 attributes, histograms size 20, random distribution). Time is reported in seconds

Performance: In order to evaluate the performance of multidimensional histograms for Anytime TA we used a synthetic data sets and compared the running times for one- and two-dimensional histograms (*gPDF*s). Each of our results are averaged over five independent experiments. As shown in Table 6, using multidimensional *gPDF*s requires a significant overhead regardless of the size of the histograms. This is due to the increased number of updates required to maintain the multidimensional *gPDF*s for each round.

7.5 Fuzzy data sets

Thus far we have demonstrated the feasibility of our approach for exact data. We now extend our evaluation to include fuzzy data (where values deviate from the true value according to some error model). For our evaluation we explore two error models, viz., Gaussian and Uniform as described earlier in the article.

For the fuzzy experiments we evaluate our approach by adjusting the variance (standard deviation squared) σ^2 and error ϵ of the Gaussian and Uniform error models respectively. The mean values of each attribute ranges between 0.0 and 10.0.

For the Gaussian error model, we evaluated our approach using several levels of fuzziness (variance), viz.: $\sigma^2 = 0.0$, 0.1, and 0.2 ($\sigma^2 = 0.0$ is used for comparison against exact data). Further, since Gaussian distributions are unbounded, the portion of the Gaussian that were beyond the histogram lower and upper bounds was truncated.

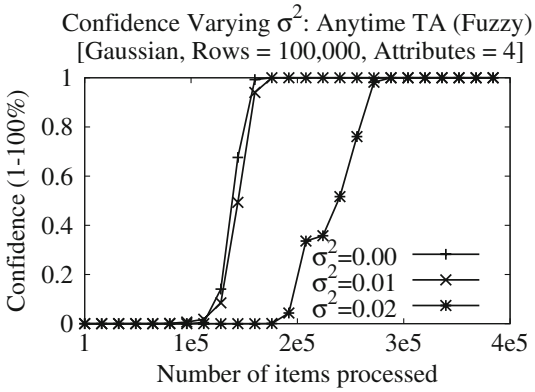

Fig. 10 In this experiment, we illustrate Confidence for varying levels of fuzziness as the number of tuples is increased using a synthetic data set

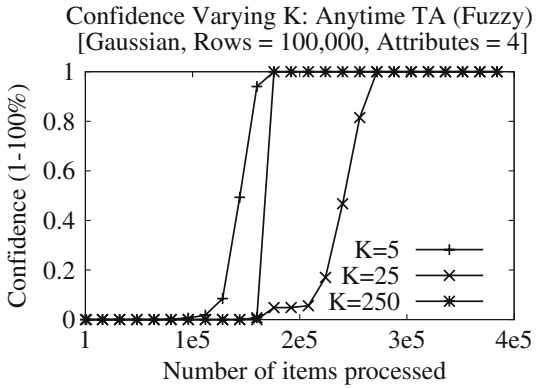

Fig. 12 In this experiment, we evaluate Confidence for varying k as the number of seen tuples is increased using a synthetic data set with a variance σ^2 (i.e., fuzziness) of 0.1

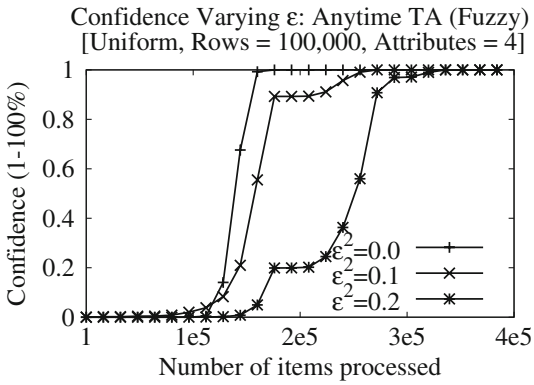

Fig. 11 In this experiment, we illustrate Confidence for varying levels of fuzziness (ϵ) as the number of tuples is increased using a synthetic data set

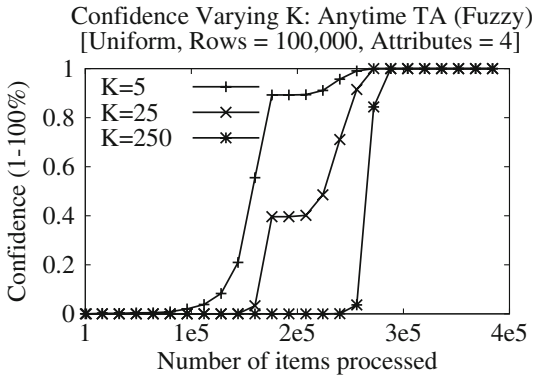

Fig. 13 In this experiment, we evaluate Confidence for varying k as the number of seen tuples is increased using a synthetic data set with an error of ϵ (i.e., fuzziness) of 0.1

Similarly, for the Uniform error model, recall that the mean value is perturbed by the error defined as ϵ. For this model we evaluated our approach using several levels of fuzziness, viz.: $\epsilon = 0.0$, 0.1, and 0.2 ($\epsilon = 0.0$ is used for comparison against exact data).

Next we shall evaluate the performance of our approach for computing Confidence, Precision, and Score Distance as described in Sect. 6.4.

Confidence: As shown in Figs. 10 and 11, the fuzziness of the data plays a key role in how quickly the confidence reaches 100%. As shown in Fig. 10, more than 50,000 additional tuples must be read at a fuzziness of $\sigma^2 = 0.02$ to achieve results similar to that of exact data ($\sigma^2 = 0.0$). Similarly for Uniform data (Fig. 11), as many as 100,000 tuples must be read when $\epsilon = 0.2$ to achieve comparable results to that of exact data ($\epsilon = 0.0$). The results suggest that fuzziness in the data can greatly affect the number of tuples that must be read to reach some predefined Confidence.

Further, it is easy to see that as the fuzziness increases, more tuples must be read to reach 100% Confidence. This is attributed to the fact that as the data is increasingly perturbed (made more fuzzy), it becomes more and more difficult for the algorithm to achieve an adequate gap between the $kMinScore$ and the $MaxOthers$. Further, as σ^2 and ϵ increase, the tails of the error distributions increase as well. For large values of σ^2 and ϵ, it is easy to see that scenarios may occur where 100% Confidence may not be reached.

Next we evaluate Confidence testing several values of k with a fixed fuzziness ($\sigma^2 = 0.01$ and $\epsilon = 0.01$). As shown in Figs. 12 and 13, as the size of k increases the time required to reach 100% Confidence increases steadily as expected. These figures illustrate how our estimates coincide with the number of correct results in the top-k buffer.

Precision: In Figs. 14 and 15, Precision is shown to be more pessimistic than Confidence when $p = 0.95$. Notice the relatively sharp increase in Precision as opposed to the more gradual increase in Confidence. This is understandable since Precision requires a tunable parameter p. The pessimism can be reduced by reducing p (the minimum probability that any

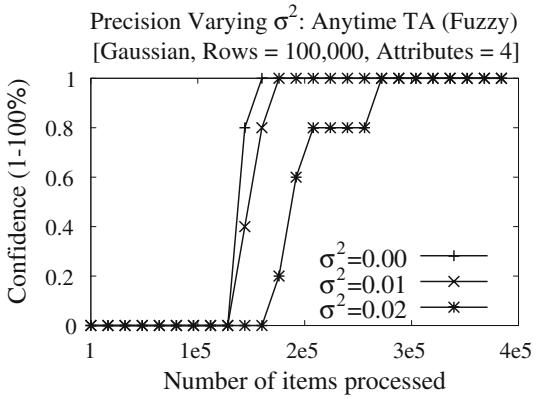

Fig. 14 In this experiment, we illustrate Precision for varying levels of fuzziness as the number of tuples is increased using a synthetic data set

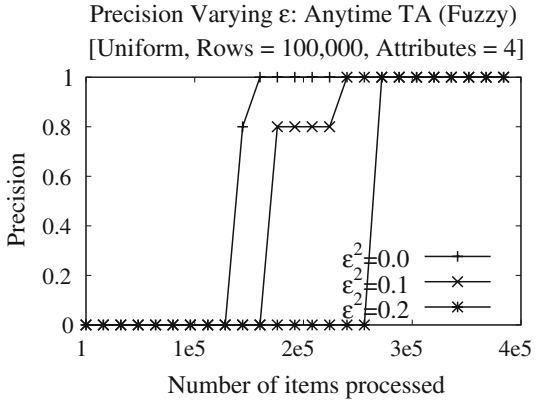

Fig. 15 In this experiment, we illustrate Precision for varying levels of fuzziness (ϵ) as the number of tuples is increased using a synthetic data set

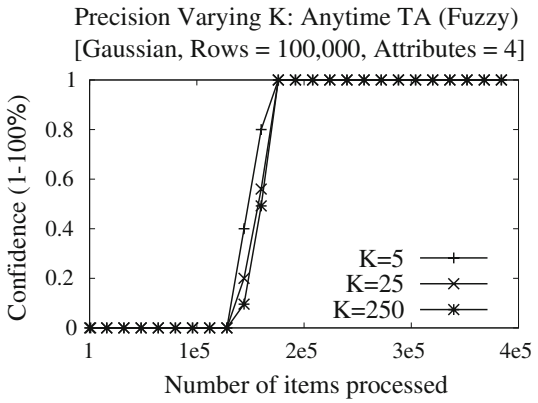

Fig. 16 In this experiment, we evaluate Precision for varying k as the number of seen tuples is increased using a synthetic data set with a variance σ^2 (i.e., fuzziness) of 0.1

tuple in the top-k buffer is assumed to be in the actual top-k set).

We evaluate our approach testing several values of k with a fixed fuzziness ($\sigma^2 = 0.01$ and $\epsilon = 0.01$). As shown in

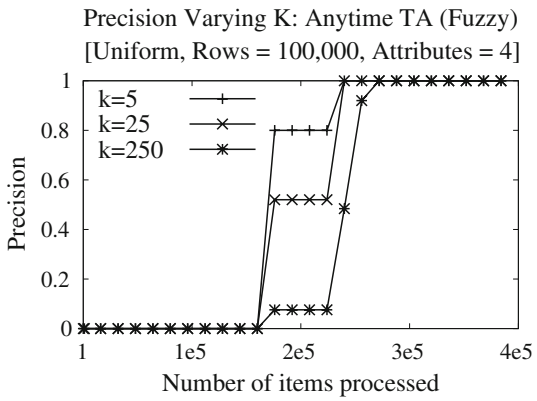

Fig. 17 In this experiment, we evaluate Precision for varying k as the number of seen tuples is increased using a synthetic data set with an error of ϵ (i.e., fuzziness) of 0.1

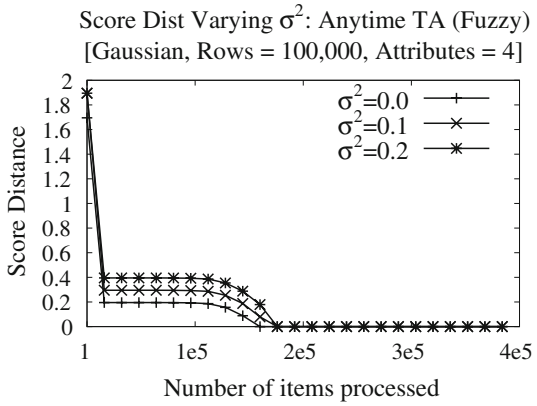

Fig. 18 In this experiment, we illustrate Score Distance for varying levels of fuzziness as the number of tuples is increased using a synthetic data set

Figs. 16 and 17, as the size of k increases the time required to reach 100% increases steadily as expected. These figures illustrate how our estimates coincide with the number of correct results in the top-k buffer.

Score distance: For all of our experiments Score Distance has a required confidence of 95%. As shown in Figs. 19 and 21, the score interval decreases as more tuples are seen by the algorithm and eventually reaches an interval of 0 signifying the top-k group has been found. The minimum score interval is bounded by the uncertainty of the data (Figs. 18, 20).

The behavior of Score Distance is consistent for both exact and fuzzy data (refer to Figs. 19, 21) with score distance improving quickly as the number of items encountered increases. Interestingly, unlike Confidence and Precision as shown above, Score Distance has a much smoother transition as more tuples are read.

Next we evaluate Score Distance testing several values of k with a fixed fuzziness ($\sigma^2 = 0.01$ and $\epsilon = 0.01$). As shown in Figs. 19 and 21, as the size of k increases, the number of

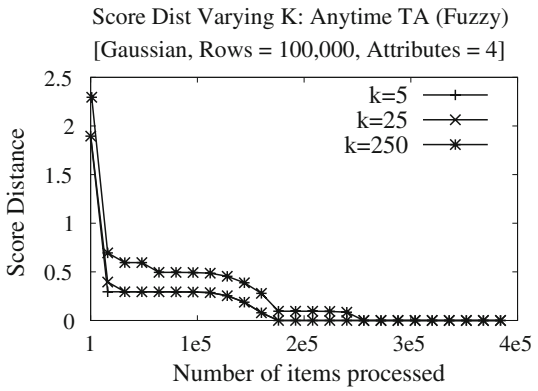

Fig. 19 In this experiment, we evaluate Score Distance for varying k as the number of seen tuples is increased using a synthetic data set with a variance σ^2 (i.e., fuzziness) of 0.1

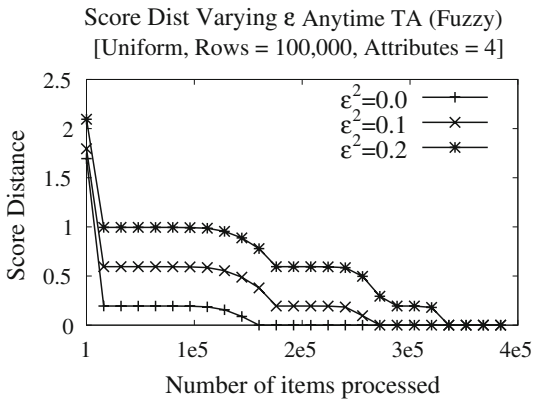

Fig. 20 In this experiment, we illustrate Score Distance for varying levels of fuzziness (ϵ) as the number of tuples is increased using a synthetic data set

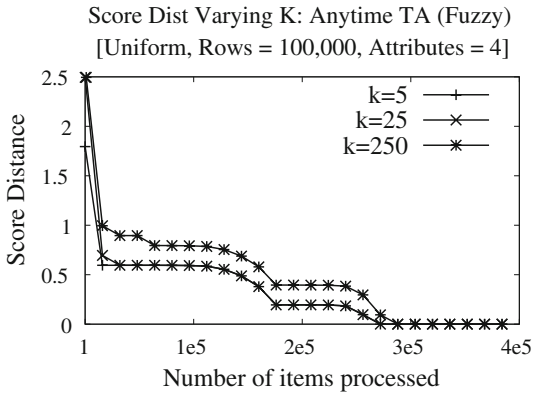

Fig. 21 In this experiment, we evaluate Score Distance for varying k as the number of seen tuples is increased using a synthetic data set with an error of ϵ (i.e., fuzziness) of 0.1

tuples read to reach a Score Distance of 0 decreases steadily as expected. In addition, it appears that the size of k makes little difference for similar values of k such as $K = 5$ and $K = 25$ as shown in Figs. 19 and 21. These figures illustrate how our estimates coincide with the number of correct results in the top-k buffer.

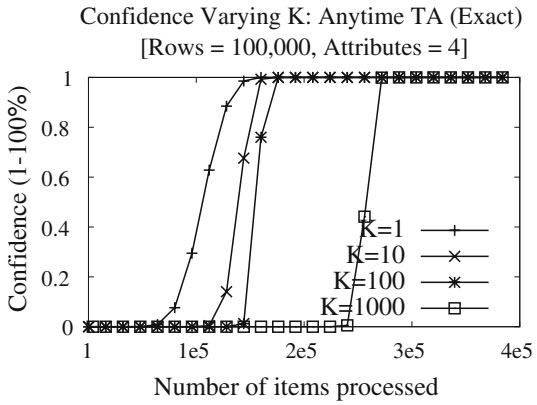

Fig. 22 In this experiment, we illustrate Confidence for varying values of K using a synthetic data set

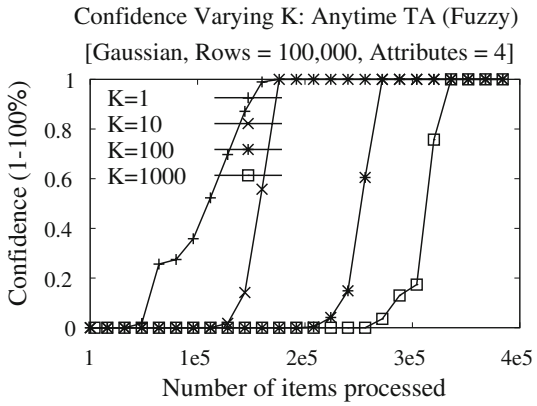

Fig. 23 In this experiment, we illustrate Confidence for varying values of K using a synthetic data set with a variance σ^2 (i.e., fuzziness) of 0.1

Varying K: In the next set of experiments, we offer a sensitivity study on K—keeping all other parameters constant. As shown in Figs. 22 and 23, as K increases the number of tuples required to reach 100% confidence increases as well.

Notice the difference between Figs. 22 and 23: for exact data, the slope of the confidence curves are very steep, while for fuzzy data (shown in Fig. 23) the rate at which the confidence climbs from 0 to 100% is less so. This is due to the introduction of doubt in the data—i.e., exact data will tend to reach 100% confidence quickly, where fuzzy data tends to build confidence more steadily.

8 Conclusions

In this article, we have presented an anytime framework for top-k computations on exact and fuzzy data. Our framework can be applied on a variety of popular top-k algorithms (TA

and TA-Sorted) to enable anytime behavior. We have discussed and analytically demonstrated several properties of our framework regarding the behavior of several measures of interest to anytime top-k computations. Through a detailed experimental study we have demonstrated the practical promise of our approach for certain important classes of data sets and applications.

There are several interesting future research directions that can be pursued to take this work forward. Even for the real-world scenarios where our techniques can in principle be applied, much work needs to be done before we can claim truly practical solutions, as many of our assumptions may be compromised (available data distributional models are inaccurate, scoring functions may not be linear, and so on). In the extreme case, in many real-world applications (e.g., top-k merging of information from hidden web sources), knowledge of data distribution is often not available at all. In such applications, it would be of interest to investigate if any sort of anytime guarantees (even alternate weaker versions) are at all possible.

Acknowledgments The work of Dimitrios Gunopulos was supported by NSF (IIS 0330481, IIS 0534781). The work of Gautam Das was supported by unrestricted gifts from Microsoft Research and start-up funds from the University of Texas, Arlington.

References

1. Barbará, D., Garcia-Molina, H., Porter, D.: The management of probabilistic data. IEEE Trans. Knowl. Data Eng. **4**(5), 487–502 (1992)
2. Bruno, N., Chaudhuri, S., Gravano, L.: Top-k selection queries over relational databases: mapping strategies and performance evaluation. TODS **27**(2) (2002)
3. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web accessible databases. In: Proceedings of ICDE, April 2002
4. Chang, K., Hwang, S.: Minimal probing: supporting expensive predicates for top-k queries. In: SIGMOD (2002)
5. Chaudhuri, S., Gravano, L.: Evaluating top-k selection queries. In: VLDB, pp. 397–410 (1999)
6. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: SIGMOD (2003)
7. Cheng, R., Kalashnikov, D., Prabhakar, S.: Querying imprecise data in moving object environments. In: IEEE TKDE (2004)
8. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: VLDB (2004)
9. chi Chang, Y., Bergman, L., Castelli, V., Li, C., Lo, M.L., Smith, J.: The onion technique: indexing for linear optimization queries. In: Proceedings of ACM SIGMOD, pp. 391–402 (2000)
10. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search metric spaces. In: Proceedings of VLDB, pp. 426–435, August 1997
11. Dean, T., Boddy, M.: An analysis of time dependent planning. In: Proceedings of the National Conference on AI (1988)
12. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: VLDB (2004)
13. Donjerkovic, D., Ramakrishnan, R.: Probabilistic optimization of top-N queries. In: Proceedings of VLDB, August 1999
14. Fagin, R.: Combining fuzzy information from multiple systems. In: PODS, pp. 216–226, June 1996
15. Fagin, R.: Fuzzy queries in multimedia database systems. In: PODS, pp. 1–10, June 1998
16. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS, June 2001
17. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. JCSS **66**(4), 614–656 (2003)
18. Fagin, R., Wimmers, E.: Incorporating user preferences in multimedia queries. In: ICDT, pp. 247–261, Jan 1997
19. Gunopulos, D., Kollios, G., Tsotras, V.J., Domeniconi, C.: Approximating multi-dimensional aggregate range queries over real attributes. In: SIGMOD, pp. 463–474 (2000)
20. Guntzer, U., Balke, W.-T., Kiesling, W.: Optimizing multi-feature queries for image databases. VLDB, pp. 419–428 (2000)
21. Horvitz, E.: Reasoning about belifs and actions under computational resource constraints. In: Proceedings of the Third Workshop on Uncertainy in AI (1987)
22. Hristidis, V., Koudas, N., Papakonstantinou, Y.: Prefer: a system for the efficient execution of multi-parametric ranked queries. In: SIGMOD Conference, pp. 259–270 (2001)
23. Hua, M., Pei, J., Zhang, W., Lin, X.: Efficiently answering probabilistic threshold top-k queries on uncertain data. In: ICDE (2008)
24. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting Top-k join queries in relational databases. VLDB J. **13**(3), 207–221 (2004)
25. Lakshmanan, L.V.S., Leone, N., Ross, R., Subrahmanian, V.S.: ProbView: a flexible probabilistic database system. ACM Trans. Database Syst. **22**(3), 419–469 (1997)
26. Lian, X., Chen, L.: Probabilistic ranked queries in uncertain databases. In: EDBT (2008)
27. Marian, A., Bruno, N., Gravano, L.: Evaluating Top-k Queries Over Web Accesible Sources. TODS **29**(2) (2004)
28. Mohamed Soliman, K.C.C.: Ihab Ilyas. Top-k query processing in uncertain databases. In: ICDE (2007)
29. Natsev, A., Chang, Y.-C., Smith, J.R., Li, C.-S., Vitter, J.S.: Supporting incremental join queries on ranked inputs. In: VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, pp. 281–290 (2001)
30. Nepal, S., Ramakrishna, M.V.: Query processing issues in image (multimedia) databases. In: ICDE, pp. 22–29 (1999)
31. Poosala, V., Ioannidis, Y.E.: Selectivity estimation without the attribute value independence assumption. VLDB, pp. 486–495 (1997)
32. Re, C., Dalvi, N.N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: ICDE, pp. 886–895 (2007)
33. Tao, Y., Cheng, R., Xiao, X., Ngai, W., Kao, B., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density. In: VLDB (2005)
34. Theobald, M., Weikum, G., Schenkel, R.: Top-k query evaluation with probabilistic guarantees. In: Proceedings of VLDB (2004)
35. Tsaparas, P., Palpanas, T., Kotidis, Y., Koudas, N., Srivastava, D.: Ranked join indices. In: ICDE (2003)
36. Yi, K., Li, F., Kollios, G., Srivastava, D.: Efficient processing of top-k queries in uncertain databases. In: ICDE (2008)