Final Report of An Implemetaion to Uncertain Databases' Top k Query

Bolong Zhang, Tao Wang

Abstract—Efficient processing of top-k queries is a crucial requirement in many DBMS. Traditional top-k queries are for the certain database system, which will return the k objects with the maximum scores based on some scoring function. Top-k processing in uncertain databases is semantically and computationally different from traditional top-k processing. In this report, we describe our implemetation of the following top-k query semantics: OptU-topk, OptU-kRanks, IndepU-Topk, and IndepU-kRanks.

Index Terms—Database system, Top-k processing, Uncertain database

1 Introduction

Uncertain data management has been used in many practical applications in domains like sensor networks, data cleaning, and location tracking, where data is intrinsically uncertain. Many uncertain (probabilistic) data models [1] [3] have been proposed to capture data uncertainty on different levels. According to many of these models, tuples have membership probability. The probabilistic of the tuple attribute could be multiple values drawn from discrete or continuous domain.

Many uncertain data models use possible worlds semantics, where an uncertain database is viewed as a set of possible worlds associated with probabilities. Each possible world represents a valid combination of database tuples. The validity of some tuple combination is determined based on the underlying tuple dependencies. The detail of the data model can be found in the paper [4] [7] [8].

Top-k queries in probabilistic databases, tuples probabilities arise as an additional ranking dimension that interacts with tuples scores. Both tuple probabilities and scores need to be factored in the interpretation of top-k queries in probabilistic databases. For example, it is not meaningful to report a top-scored tuple with insignificant probability. Alternatively, it is not accepted to order tuples by probability, while ignoring their scores. Moreover, combining scores and probabilities using some score aggregation function eliminates uncertainty completely, which may not be meaningful in some cases, and does not conform with the currently adopted probabilistic query models.

In this report, we chose to implement uncertain top-k semantics metioned in [4], which are OptU-topk, OptU-kRanks, IndepU-Topk, and IndepU-kRanks. Our implementation includes the following steps: 1. Generate datasets. For singlar variable uniformly/normally/exponentially distributed tuples, use C++11s new random number generation methods (Defined in header ¡random¿); for bivarate normally distributed tuples, use C++11 to call matlab's engine to get the designated dataset. 2. Put the dataset into MYSQL database, using MYSQL C++s API. 3. Leverage MYSQLs storage and query processing techniques to compute score-ranked tuples, and output these tuples for further processing, again, using MYSQL C++ API. 4. Realize various uncertain top-k algorithms, including OPTUtopk, OPTUkrank, Indenp-Utopk, Indenp-Ukrank. 5. Display each individual TOP-k result, using GUI interface. 6. Display the comparison of result from different datasets, using C++11 to call matlab's engine to plot the result.

In our implementation, we made the following contributions:

1) **Rule Engine** is used to calculate the state probability. However, in the paper the author not describe the details of implementation of the Rule engine [2]. We design our own engine to calculate the state probability.
2) Implemented all the algorithms mentioned in the paper, including OptU-Topk, Optu-kRanks, IndepU-topk, IndepU-kRanks. To test the efficiency, we did all the experiments in the paper.
3) All the required functions were realized in a single program. For example, you do not need to use R-statistical computing package to generate datasets, then use your C code to calculate the results, and then use some graphical tools to plot the result... , just like what was done in the original paper. In our implementation, all the jobs could be done in a all-in-one program.
4) We realized the realtime dataset generating and realtime result plotting. We can directly compare our implementation result with the result shown on the original paper. Besides good visualization effect, real-time dataset and result presenting could effectively clear up some potentially existing doubt about the realness of the data.

There are some other aspects in our implementation need to be metioned:

1) wxWidgets is used to realize the GUI related function, and there are some code-generating tools for wxWidgets(such as wxcraft in CodeLite). To downsize the overall code size(our code is approximately 4300 lines), we did not use any auto-code-generating tools.
2) In the paper, R-statistical computing package was used to create datasets of different data distributions (the paper was published in 2007); in our code, C++11s new random number generation methods (Defined in header ¡random¿) were used instead to generate singlar variable uniformly/normally/exponentially distributed tuples, we also use C++11 to call matlab's engine to generate bivarate normally distributed tuples.
3) In the paper, the framework RankSQL (based on PostgreSQL, as described in [9]) was used to provide score-sorted tuples; in our code, MYSQL was used due to its being one of the most popular RDBMSs.
4) In our implementation, the following softwares could work together: wxWidgets 3.0.2, MYSQL 5.7.17, MATLAB 2017a, and ubuntu 16.0.4. We found that wxWidgets 3.1.0/3.0.2 was incompatible with macOS Sierra 10.12.4, although it could work with the previous of macOS(such as macOS EI).

2 Related Work

The recent work on the semantics of top-k queries on uncertain data roughly fall into two categories: (1) returning k tuples that can coexist in a possible world (i.e., that must follow the mutual exclusion rules) or (2) returning tuples according to the marginal distribution of top-k results (across all possible worlds). The U-Topk [4] and c-Typical-Topk [7] definitions belong to category (1) , while the U-kRanks [4] and PT-K [8] definitions belong to (2) . Yi [5] proposed an improved algorithm for U-Topk and U-kRanks.

The uncertain data model U-Topk and U-kRanks introduced in paper [4] is based on possible worlds semantics with two pillars. The first pillar is membership uncertainty, where each tuple belongs to the database with a probability, hereafter called confidence. The second pillar is generation rules, which are arbitrary logical formulas that determine valid worlds. Tuples that are correlated with no rules are called independent.Each possible world is a combination of tuples. The probability of each world is computed by assuming the existence of all tuples in the world, and the absence of all other database tuples. The outcome of

this computation is determined based on tuple confidence values and generation rules [4].

An extension of the category (1) named c-Typical-Topk was introduced in [7] on the basis of U-Topk [4]. The authors pointed out in [7] that U-Topk choose the result k tuple vector purely on its probability (without using score as a weight), and thus the total score of the result U-Topk vector can be rather atypical. The semantics of c-Typical-Topk [7] allows users to choose between results with various weights of score and probability freely. In their paper, the authors also extend the semantics and algorithms to the scenario of score ties, which is not dealt with in the previous work in the area.

In [8], the authors pointed out that both U-Topk and U-kRanks mentioned in [4] are "rank sensitive" , and thus could neglect some tuples which have a high probability to appear in the top-k list(because the neglected tuples may not be the top in some positions). The authors introduced PT-K, which address an application scenario other than the semantics in [4]. Given a probability threshold p, PT-K finds the set of records where each takes a probability of at least p to be in the top-k lists across all the possible worlds.

In [5], the authors improved the algorithms shown in [4] for the x-relation database system. Besides that an uncertain data set, which they call an x-relation (consists of a number of x-tuples), was introduced. Each x-tuple includes a number of alternatives, associated with probabilities, which represent a discrete probability distribution of these alternatives being selected [5].

3 Definition for Uncertain Top-k

Assuming some scoring (ranking) function to order tuples, the probability of a $k-length$ tuple vector T to be the topk is the summation of possible worlds probabilities where T is the $topk$. Similarly, the probability of a tuple t to be at rank i is the summation of possible worlds probabilities where t is at rank i. We now formally define uncertain $topk$ queries based on marriage of possible worlds and traditional $topk$ semantics.

	Time	Radar Loc	Car Model	Plate No	Speed	Conf
t1	11:45	L1	Honda	X-123	130	0.4
t2	11:50	L2	Toyota	Y-245	120	0.7
t3	11:35	L3	Toyota	Y-245	80	0.3
t4	12:10	L4	Mazda	W-541	90	0.4
t5	12:25	L5	Mazda	W-541	110	0.6
t6	12:15	L6	Chevy	L-105	105	1.0

$Rules: (t2 \oplus t3), (t4 \oplus t5)$

(a)

World	Prob.
PW^1={t1,t2,t6,t4}	0.112
PW^2={t1,t2,t5,t6}	0.168
PW^3={t1,t6,t4,t3}	0.048
PW^4={t1,t5,t6,t3}	0.072
PW^5={t2,t6,t4}	0.168
PW^6={t2,t5,t6}	0.252
PW^7={t6,t4,t3}	0.072
PW^8={t5,t6,t3}	0.108

(b)

Fig. 1. Uncertain Database and Possible Worlds Space

Definition 1 ***Uncertain Top-k (U-Topk):*** Let $\mathcal{D}$ be an uncertain database with possible worlds space $\mathcal{PW} = PW^1, \cdots, PW^n$. Let $T = T^1, \cdots, T^m$ be a set of $k - length$ tuple vectors, where for each $T^i \in \mathcal{T}$: (1) Tuples of T^i are ordered according to scoring function $\mathcal{F}$, and (2) T^i is the $topk$ answer for a non empty set of possible worlds $PW(T^i) \subseteq \mathcal{PW}$. A $U-Topk$ query, based on $\mathcal{F}$, returns $T^* \in \mathcal{T}$, where $T = argmax_{T^i \in \mathcal{T}} \sum_{\omega \in PW(T^i)}(Pr(\omega)))$.

U-Topk query answer is a tuple vector with the maximum aggregated probability of being topk across all possible worlds.

Definition 2 ***Uncertain k Ranks Query (U-kRanks):*** Let $\mathcal{D}$ be an uncertain database with possible worlds space $\mathcal{PW} = PW^1, \cdots, PW^n$. For $i = 1, \cdots, k$, let $x_i^1, \cdots, x_i^m$ be a set of tuples, where each tuple x_i^j appears at rank i in a non empty set of possible worlds $PW(x_i^j) \subseteq \mathcal{PW}$ based on scoring function $\mathcal{F}$. A UkRanks query, based on $\mathcal{F}$, returns $\{x_i; i = 1, \cdots k\}$, where $x_i = argmax_{x_i^j}(\sum_{\omega \in PW(x_i^j)}(Pr(\omega)))$.

$U-kRanks$ query answer is a set of tuples that might not form together the most probable topk vector. However, each tuple is a clear winner at its rank over all worlds.

Figure 1 is an uncertain database example. Table (a) is the list of tuples and Table (b) is all the possible worlds. According to above definitions. We can get the answer for $Utop-2$, the answer ist_1, t_1. For $U-2Ranks$, the answer is t_2, t_6.

4 ALGORITHMS FOR THE TOP-K PROCESSING

4.1 U-Topk algorithm

The problem of finding top-k query answers in uncertain databases is formulated as searching the space of states that represent possible top-k answers, where a state is a possible prefix of one or more worlds ordered on score. Each state has a probability equal to the aggregate probability of the possible worlds prefixed by this state.

We now describe $OPTU-Topk$, an optimal algorithm in the numbers of accessed tuples and visited search states to answer a U-Topk query. In the algorithm, for each steps we extend the state with highest probability. We overload the definition of a search state s_l to be $s_{l,i}$, where i is the position of the last seen tuple by $s_{l,i}$ in the score-ranked tuple stream. We define e the empty state of length 0. Algorithm start by initializing e with $s_{0,0}$, where $\mathcal{P}(S_{0,0}) = 1$ Let Q be the priority queue of states ordered on their probabilities. We initialize Q with e. Let d be the number of seen tuples from $\mathcal{D}$ at any point. Then the algorithm iteratively retrieves the top state of Q , say $s_{l,i}$ extends it into the next possible states and inserts two states back to Q based on their probabilities. If $i = d$, consuming new tuple from D, otherwise extended it with buffered tuple pointed to by $i + 1$ The termination condition: when the top state Q is completed state.

The algorithm to compute U-Topk is shown in Algorithm 2.

Algorithm 1 OptU-Top$k$($source$,k)

Require:
$source$: Score-ranked tuple stream
k: Answer length
Ensure: U-Top$k$ query answer
1: $Q \leftarrow$ empty priority queue for states ordered on probabilities
2: $e \leftarrow s_{0,0}$ where $\mathcal{P}(e) = 1$ {init empty state}
3: $d \leftarrow 0$ {scan depth in $source$}
4: Insert e into Q
5: **while** ($source$ is not exhausted AND Q is not empty) **do**
6: $s_{l,i} \leftarrow$ dequeue (Q)
7: **if** ($l = k$) **then**
8: **return** $s_{l,i}$
9: **else**
10: **if** ($i = d$) **then**
11: $t \leftarrow$ next tuple from $source$
12: $d \leftarrow d + 1$
13: **else**
14: $t \leftarrow$ tuple at pos $i + 1$ from seen tuples
15: **end if**
16: Extend $s_{l,i}$ using t into $s_{l,i+1}$, $s_{l+1,i+1}$ {Section 4.1}
17: Insert $s_{l,i+1}$, $s_{l+1,i+1}$ into Q
18: **end if**
19: **end while**
20: **return** dequeue(Q)

Fig. 2. The description for OptU-Topk algorithm

4.2 U-kRanks algorithm

In this section, we describe $OPTU-kRanks$, an optimal algorithm in the number of accessed tuples to answer UkRanks queries. Algorithm OPTU-kRanks extends maintained states based on each seen tuple. When a new tuple is retrieved, it is used to extend all states causing all possible ranks of this tuple to be recognized. Let t be a tuple seen after retrieving d tuples from the score-ranked stream. Let $P_{t,i}$ be the probability that tuple t appears at rank i, based on scoring function $\mathcal{F}$, across all possible worlds. It follows from our state definition that $P_{t,i}$ is the summation of the probabilities of all states with length i whose tuple vectors end with t, provided that t is the last retrieved tuple from D. In other words, we can compute $P_{t,i}$, for $i = 1 \cdots k$, as soon as we retrieve t from the database. For each rank i, we need to remember only the most probable answer obtained so far, since any unseen tuple u cannot change $P_{t,i}$ of a seen tuple t because u can never appear before t in any possible world. Stopping condition. Let t^i be the current $U-kRanks$ answer for rank i, let S_{i1} be the set of all maintained states with length $i1$. The termination condition of $OPTU-kRanks$ algorithm, at rank i, is $P_{t^i,i} > \sum_{s \in S_{i1}} \mathcal{P}(s)$. Algorithm 2 formally describes OPTU-kRanks. The next Theorem formalizes the optimality of OPTU-kRanks.

The algorithm is shown in Algorithm 3.

4.3 Rule Engine

In order to compute the state probability in above two algorithms , a very import part called Rule Engine is used. However, in the paper, the author did not discuss the details to

Algorithm 2 OptU-kRanks($source$,k)

Require:
$source$: Score-ranked tuple stream
k: Answer length
Ensure: U-kRanks query answer
1: $answer[\,] \leftarrow$ empty vector of length k
2: $ubounds[\,] \leftarrow$ vector of length k initialized with 1's {prob. upper bound of any unseen tuple at each rank $1 \ldots k$}
3: $reported \leftarrow 0$ {No. of reported answers}
4: $depth \leftarrow 1$
5: $space \leftarrow \phi$ {current set of states}
6: **while** ($source$ is not exhausted AND $reported < k$) **do**
7: $t \leftarrow$ next tuple from $source$
8: **for** i=1 to $min(k, depth)$ **do**
9: Extend $space$ states with length $i-1$ using t
10: Compute $P_{t,i}$
11: Update $ubounds[i]$ based on states of length $i-1$
12: **if** ($answer[i]$ was previously reported) **then**
13: continue
14: **end if**
15: **if** ($P_{t,i} > answer[i].prob$) **then**
16: $answer[i] \leftarrow t$
17: $answer[i].prob \leftarrow P_{t,i}$
18: **if** ($answer[i].prob > ubounds[i]$) **then**
19: Report $answer[i]$
20: $reported \leftarrow reported + 1$
21: **end if**
22: **end if**
23: **end for**
24: $depth \leftarrow depth + 1$
25: **end while**

Fig. 3. The description for U-kRanks algorithm

implement the Rule Engine. Instead, we design our own Rule Engine to Compute the State Probability. The main idea is generating all the possible worlds contain the current state as the top-k answer via the given tuples each time when we are trying to calculate the state probability. For example shown in Fig. 1, we can use the Rule Engine to enumerate all the possible world shown in the table (b). But Since we don't know how the author design their own Rule Engine, our design just a simply model, which is very time cost. The details for how the author design the engine, we can find at the paper [2], where Bayesian network was used to design the Rule Engine.

4.4 U-Topk Queries with Tuple Independent

In this section, we introduce other algorithms that make use of tuple independence to cut down the state materialization significantly. With tuple independence, state space can be aggressively pruned to keep only the states that could lead to the answer. In general, an incomplete state s can be pruned if there exists a complete state c with $\mathcal{P}(c) > \mathcal{P}(s)$. Hence, if we can compute the maximum probability of a complete state generated from s, denoted $p_{max}(s)$, we can safely prune all states with probability less than $p_{max}(s)$.

We describe Algorithm $IndepU - Topk$, which prunes the space based on the following criterion.

Lemma 1 Comparable States. Under tuple independence, a state x_n is probability-comparable to any state y_m, where x_n and y_m are maintained after seeing the same set of score-ranked tuples, and $n \geq m$

Lemma 1 states that for comparable states x_n and y_m, the most probable complete states derived from each of them would be obtained using the same set of existence absence events of unseen tuples. That is, $x_n$and y_m would follow the same path to reach a complete state. However, x_n will reach a complete state at most at the same time as y_m, since$n \geq m$. Based on the above and according to Property 1, we have guarantees that if $\mathcal{P}(x_n) > \mathcal{P}(y_m)$, the complete state derived from x_n would have a higher probability than the one derived from y_m, and therefore we can safely prune y_m from our search space.

$IndepU - Topk$ exploits Lemma 1 by grouping states into equivalence classes based on their lengths. $IndepU - Topk$ keeps at most one state for each length value $0, \cdots, k$ in a candidate set. The candidate set is extended on receiving each new tuple from $\mathcal{D}$. $IndepU-Topk$ terminates when at least k tuples have been retrieved, and the probability of any current state is not above the probability of the current complete candidate.

Consider the example shown in Figure 4. Where we want the $U-top3$ answer. In step (a), after retrieving the first tuple t1, we construct two states $<\neg t_1>$ and $< t_1 >$ with length values 0 and 1, respectively. In step (b), the candidate set is updated based on the new tuple t_2, where two possible candidates with length

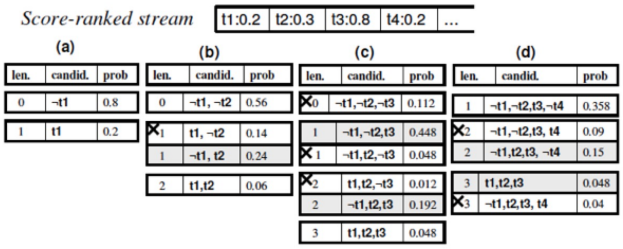

Fig. 4. $IndepU - Topk$ Processing

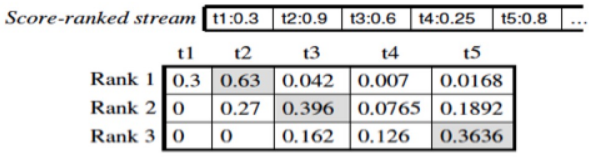

Fig. 5. $IndepU - kRanks$ Processing

1, $< t_1, \neg t_2 >$ and $< \neg t_1, t_2 >$, are generated. However, we keep only the candidate with the highest probability since both candidates are probability comparable. Step (c) continues in the same manner by updating the candidate set based on tuple t_3, and pruning the less probable candidate from each equivalence class. Note that the candidate $< \neg t_1, \neg t_2, \neg t_3 >$ is pruned because there is another candidate $< \neg t_1, \neg t_2, t_3 >$ with a larger length and higher probability. In step (c) we have constructed the first complete candidate, $< t_1, t_2, t_3 >$, and the first termination condition is met. In step (d) we update the candidate set based on t4. Notice that we cannot stop after step (d) because the second termination condition is not met yet−there are candidates with higher probabilities than the current complete candidate and so, there is a chance that $< t_1, t_2, t_3 >$ will be beaten. Space reduction by exploiting state comparability property results in huge performance improvements for large values of k.

4.5 U-kRanks Queries with Tuple Independence

When tuples are independent, a U-kRanks query exhibits the optimal substructure property, i.e. the optimal solution of the larger problem is constructed from solutions of smaller problems. This allows using a dynamic programming algorithm. We next describe $IndepU - kRanks$, a dynamic programming algorithm with optimality guarantees in the number of accessed tuples.

Consider the example depicted by Figure 4.5, where we are interested in U-3Ranks query answer. In the shown table, a cell at row i and column x indicates the value of $P_{x,i}$. The value of $P_{x,1}$ is computed as $Pr(x) \times \prod_{z:\mathcal{F}(x)<\mathcal{F}(z)}(1 - Pr(z))$. , which is the probability that x exists and all tuples with higher scores do not exist. The values of $P_{x,i}$, where $i > 1$, are computed based on the following formula. $P_{x,i} = Pr(x) \times \sum_{y:\mathcal{F}>\S}(\prod_{z:\mathcal{F}(x)<\mathcal{F}(z)<\mathcal{F}(y)}(1 - Pr(z)) \times P_{y,i-1}$, which means that with independent tuples, for a tuple x to appear at rank i, we need only to consider the probability that x is consecutive to every other tuple y at rank $i1$.

$$M[i,t] = \begin{cases} i = 1: \ Pr(t) \times \prod_{z:\mathcal{F}(t)<\mathcal{F}(z)}(1 - Pr(z)) \\ \\ i > 1: \ Pr(t) \times \sum_{y:\mathcal{F}(y)>\mathcal{F}(t)}((\prod_{z:\mathcal{F}(t)<\mathcal{F}(z)<\mathcal{F}(y)} \\ (1 - Pr(z))) \times M[i-1, y]) \end{cases}$$

The shaded cells in Figure 4.5 indicate the $U - 3Ranks$ query answers at each rank. Notice that the summation of the probabilities of each row will be 1 if we completely exhaust the tuple stream. This is because each row actually represents a horizontal slice in all the possible worlds. This means that we can report an answer from any row whenever the maximum probability in that row is greater than the row probability remainder.

The above description gives rise to the following dynamic programming formulation. We construct a matrix M with k rows, and a new column is added to M whenever we retrieve a new tuple from the score-ranked stream. Upon retrieving a new tuple t, the column of t in M is filled downwards based on the following equation: For example in Figure 4.5, $M[2,3] = Pr(t_3) \times (M[1,2] + (1 - Pr(t_2)) \times M[1,1]$. Algorithm $IndepU - kRanks$ returns a set of t tuples $t_1, \cdots . t_k$, where $t_i = argmax_x M[i,x]$.

4.6 Algorithm analysis

There are totally four algorithms listed in the paper. The complexity analysis is as following:

- For $OptU - topk$ and $OptU - kRanks$, the majority time is spent in by using the Rule Engine to compute the State probability. For the Rule Engine, the Time complexity is $O(N!)$, where N is the total number of the tuples retrieved from the Database. So for the algorithms $OptU - topk$ and $OptU - kRanks$ the time cost is a least $O(N!)$, which is less efficient.
- For the $IndepU - topk$ and $IndepU - kRanks$, the time cost is nearly $O(kN)$, where k is the required tuple size,N is the total tuples retrieved from the database.

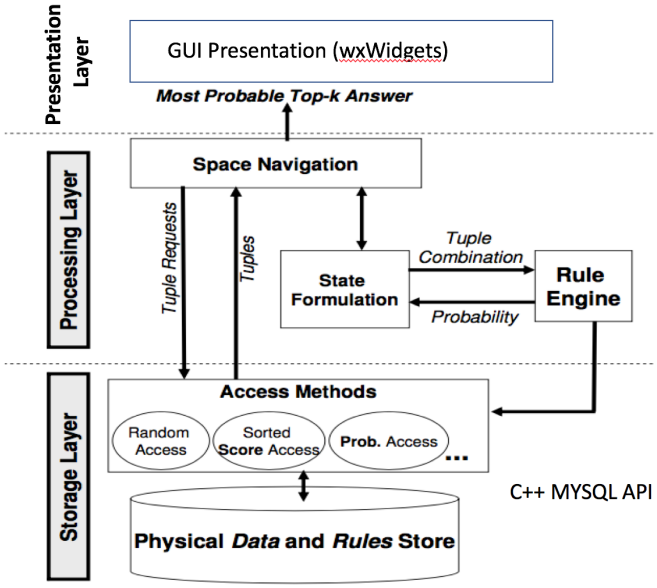

Fig. 6. framework

5 Technical Description of Our Implementation

Our implementation adopted the framework mentioned in the paper, which leveraged RDBMS storage, indexing, and query processing techniques to compute uncertain $topk$ query answers. Our proposed processing framework added an extra presentation layer(using GUI) to present our results more effectively, it consists of three main layers(Fig.6):

The Processing flow is as follows(from bottom up): Generate dataset according Generation Rules; Load the dataset into MYSQL(C++ API); Leverage MYSQLs query processing ability, generate score-ranked tuples; Processing Layer Read the score-ranked tuples; Processing, output Most Probable Top-k answers; Results presented with GUI.

5.1 Storage Layer

The main function of this layer includes dataset generating, put generated dataset into MYSQL, and then use MYSQL's ability to provide score-ranked tuples to upper layer(the processing layer).

There are several kind of dataset(with different number of tuples, in our implementation, the typical number of independent tuples is 30000, and the typical number of mutual exclusive tuples is 20) needed to be generated: uniformly distributed scores, uniformly distributed confidence, normally distributed confidence, exponentially distributed confidence, bivariate normally distributed score and confidence.

In the paper, the authors used the R-statistical computing package to generate dataset. At first, we planned to use C++11's random number generating ability (include in the ¡random¿ header) to generate all the dataset. But we found that it was too complicated to generate bivariate normally distributed dataset using C++11(For example, Boost and Eigen phenomenal libraries need to be installed, and the code is messy), we finally decided to use C++11 to call MATLAB's engine, and produce the dataset with a clean succinct MATLAB code (resides in the root directory of the project). From the user's point of view, the calling to the MATLAB engine is transparent.

In the paper, the framework RankSQL (based on PostgreSQL, as described in [9]) was used to provide score-sorted tuples; in our code, MYSQL was used due to its being one of the most popular RDBMSs. First, we use MYSQL's C++ API to insert the designated tuples into MYSQL, and then retrieve score-

ranked tuples from MYSQL. At this point, the retrieved scored-sorted tuples is ready for the processing layer to process.

5.2 Processing Layer

After retrieving the tuples from the database, we process it to get the desirable $Topk-k$ results. According to the tuples type, i.e. there are exists exclusive rules or independent between each other. There are four algorithms listed in the paper,including $OptU-topk, OptU-kRanks, IndepU-Topk, IndeU-kRanks$. The implementation techniques are list as following:

- Newly designed Rule Engine is applied in the OptU-topk and OptU-kRanks algorithms to calculate the state probability after we extend the old state with the new retrieved tuples.
- Priority Queue is used in the implementation of $OptU-topk$ algorithm to maintain the current states formed from the all the retrieved states so far.
- For OptU-kRanks to save the insert time and delete time, we use the linked list to maintain the current states space.
- The separate list is used in the $IndepU-topk$, to maintain the maximum probability state for each length from $i=1,\cdots,k$
- Dynamic programming has already used in the the $IndepU-kRanks$ to speed up the search time.

The details for the algorithm, we have described in the previous section.

5.3 Presentation Layer

To facilitate the users, we provide GUI interface, which was developed with the GUI developing tools: wxWidgets. The reason we chose wxWidgets (instead of Qt or GTK+) is wxWidgets uses the native platform SDK and system provided widgets, and thus more likely to look, behave and feel native.

Generally, a user starts with a new dataset. Click on the "dataset" menu, we can create a new dataset (FIG.7):

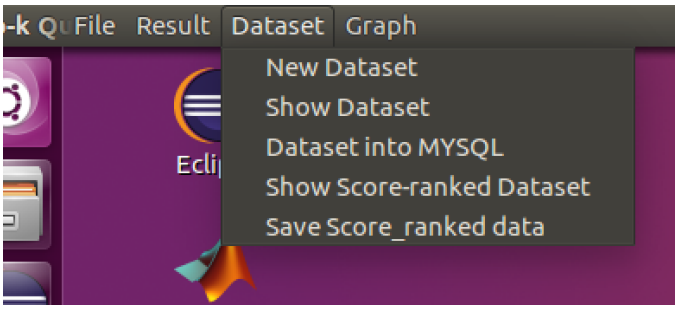

Fig. 7. Dataset menu

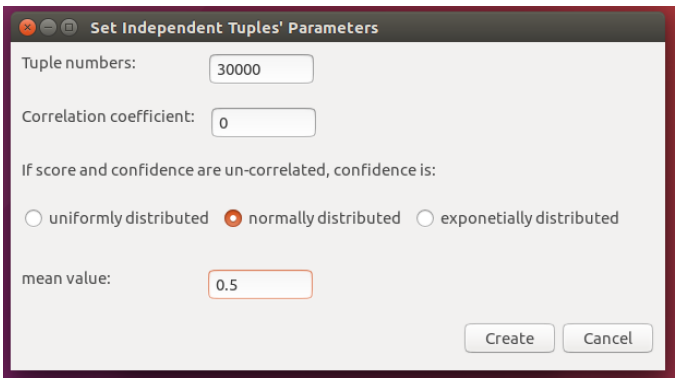

Fig. 8. Input parameters of dataset to be generated

Then we can input parameters for the dataset(Fig.8):

When dataset is successfully created, we should put the dataset into MYSQL using the menu "Dataset into MYSQL" in Fig.7. Now we have several selections: we could look at the score-ranked dataset, or save the score-ranked dataset for future use, or make a query, or output a single dataset graph of query time/depth' trend with the change of "k" values(4 "k" values could be selected for top-k query). If we make a query (Fig.9) and input the "k" values (of top-k)(Fig.10), we could see the result (Fig.11). If we choose to output a single dataset graph of query time/depth' trend (Fig.12), and input the parameters (Fig.13), we will could see the graph like (Fig.14).

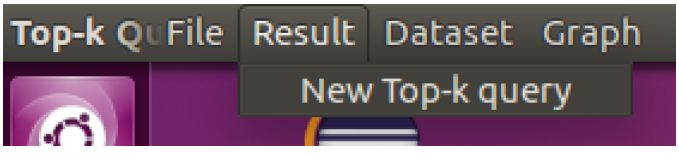

Fig. 9. Make a query

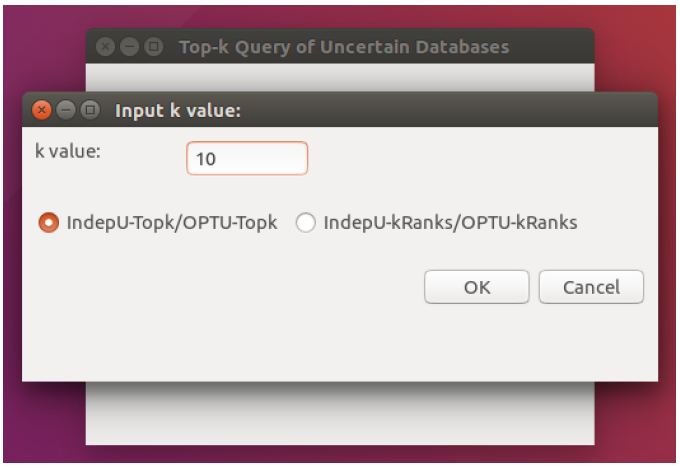

Fig. 10. Input ”k” value, and select algorithm

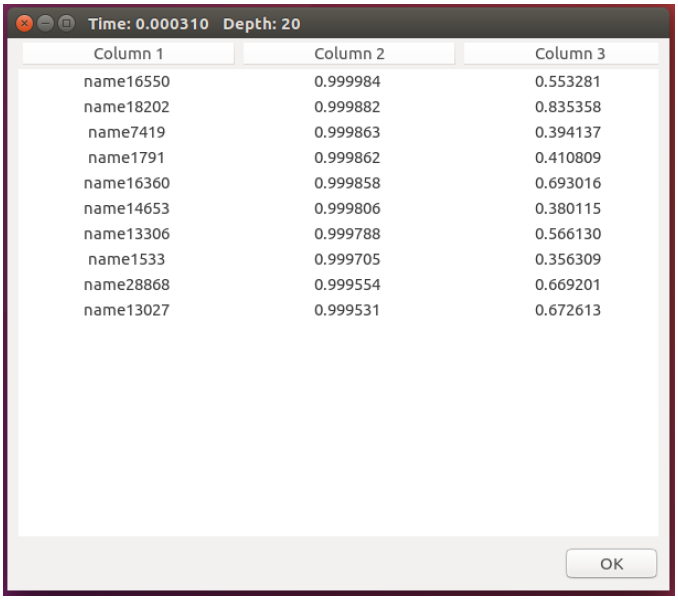

Fig. 11. Query result, with query time and depth

If we want to output different datasets' query time/depth's distribution, and make a comparison with the graphs in the paper, we could select the ”Create comparison ... graph ...” submenu in Fig.12. After input the needed parameters as shown in Fig15, we could see the result as shown in Fig.16.

Due to the limitation of software developing time and software size, we select to create the

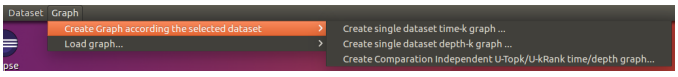

Fig. 12. Output graphs of various results

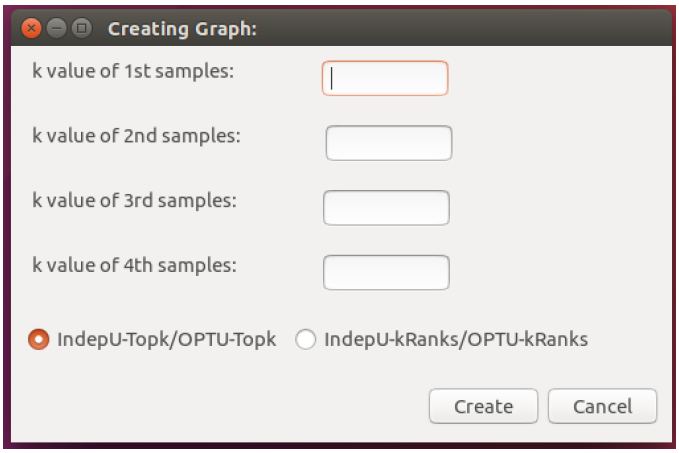

Fig. 13. Input ”k” value of samples

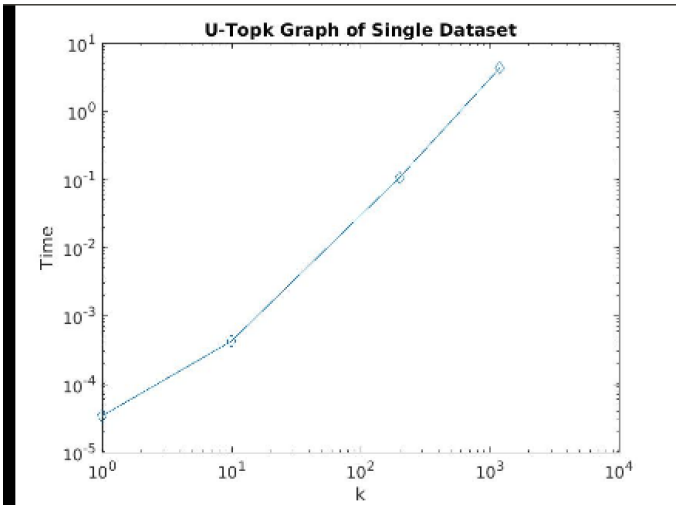

Fig. 14. Result graph of query time/depth's trend

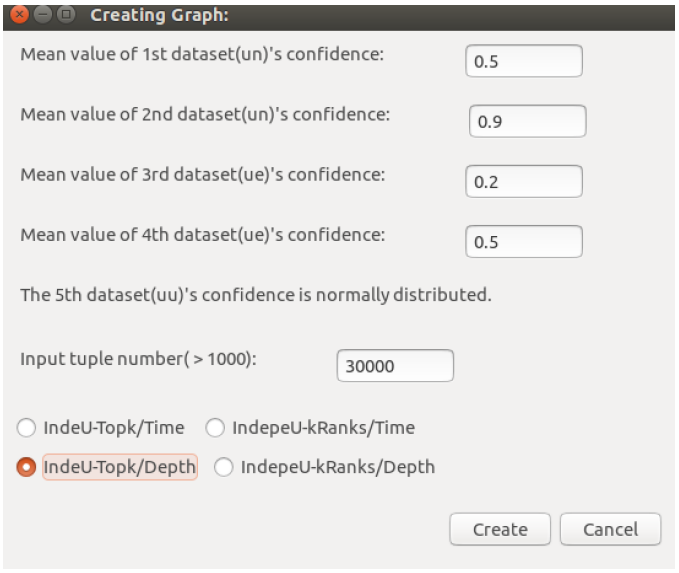

Fig. 15. Input parameters of designated comparison

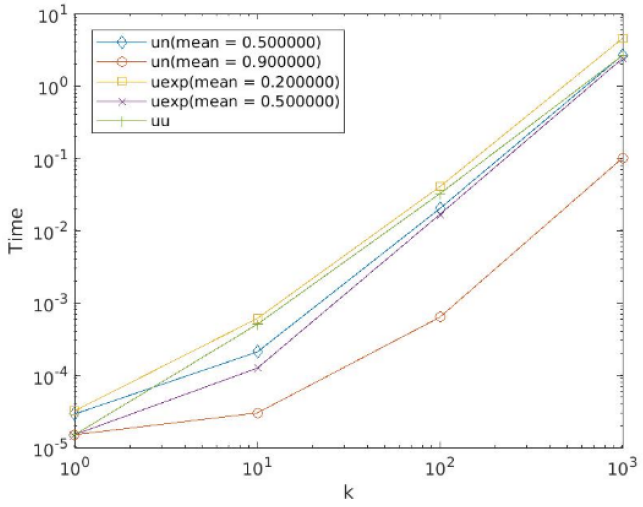

Fig. 16. Input parameters of designated comparison

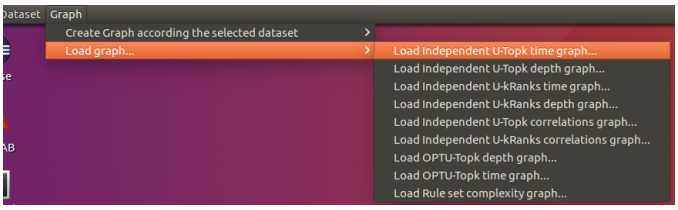

Fig. 17. Load the pre-drawn graph

first four kinds of comparison graph in the paper (figure 6 to figure 9). We also prepared all the comparison graphs in paper, which can be loaded in Fig.17.

6 Evaluation

In this part, we will compared our results with the results in the paper. **Difference between Experiments environments settings** are list as following:

- Our experiments are carried on 2.7 GHz Intel Core i5 CPU machine with 8GB Memory. The experiments in the paper were carried on 3GHZ Pentium IV PC with 1GB memory.
- We have implemented all algorithms in the paper by C++. However, in the paper, the author implemented the algorithms by C.
- Since the author did not mention the details to design Rule Engine. For OPTU-Topk, and OPTU-kRanks algorithms, we design our own Rule engine to compute the state probability. IndepU-Topk, and IndepU-kRanks algorithms to deal with the independent data, which don't need the rule engine to calculate the state probability.
- Different data set. We generate our own data sets for the testing according to what has already described in the paper. Although, the distributions are the same, the final data sets are still not exact the same as the original paper due to the randomness. For the last Rule Set complexity test, the author even didn't mention the details of the data so we have to generate the rule and data set according to our own understanding.

The main results are shown in the Fig 18. Compared with the original results. The difference are as following:

- **Test the effect of confidence distribution** The first two pictures (a) and (b) in above Fig 18, are the time and scan depth for IndepU-Topk, while the (c) and (d) show the time and scan depth of IndepU-kRanks, respectively. We test the different distribution here. The results show exactly the same results as shown in the paper originally. However, Our implementation is several ten times faster than the original paper. The reason mainly because our computer right now is faster than the platform in the paper.
- **Test Score-confidence Correlations** We have also evaluated the effect of score-confidence correlations. Our results also show what have already shown in the original paper: Positive correlations result in large savings in this case high scored tuples are attributed with high confidence, which allows reducing the number of needed to see tuples to answer the top-k queries. The results are shown in the (e) and (f) in the Figure 18.
- **Evaluating the space navigation techniques** In this part, we evaluation $OPTU-Topk$ and $OPTU-kRanks$ algorithms. We tested three cases with no,

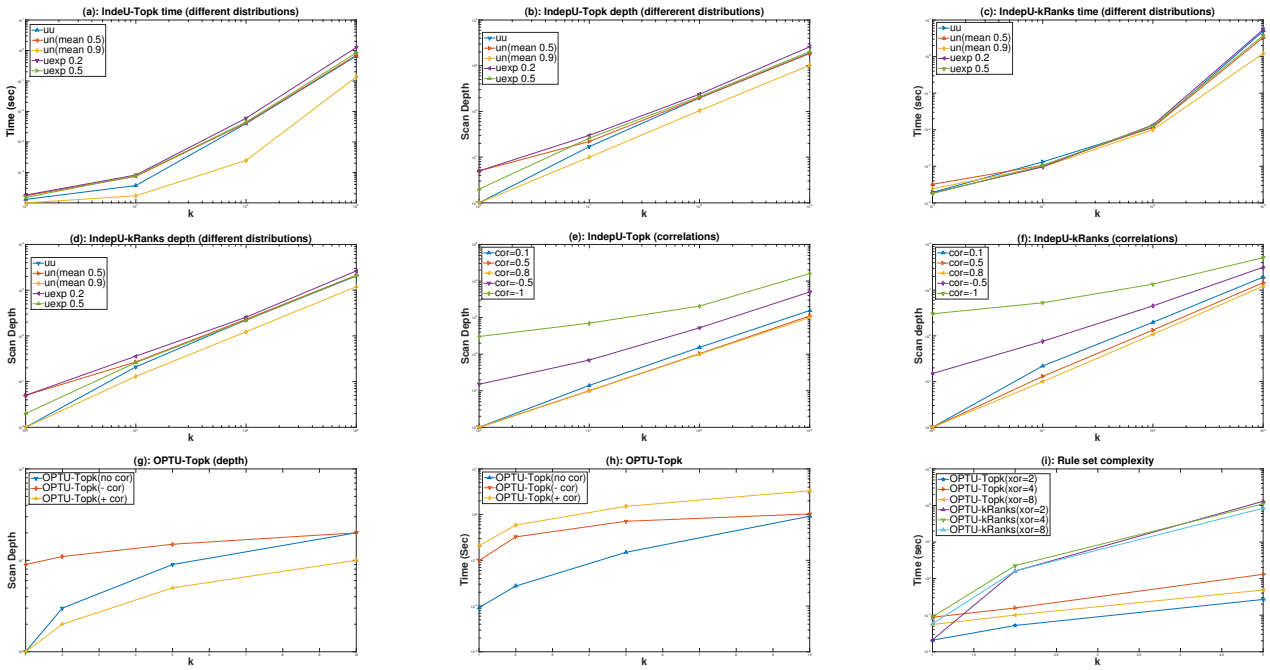

Fig. 18. Experiment results carried out by our implementation

positive and negative correlations. there 20 tuples for this three cases. The main results are shown in the (g) and (b) in Figure 18. Compared with the results in the paper, our results are almost the same. However, the different is that Our k is smaller than the original size. The reason behind this is that the different rule engine design. In our implementation, we use the brute force method, which is very time cost ($O(N!)$). So we can not test too large cases here.

- **Rule Set Complexity**, we evaluate the impact of module rules complexity on system performance. We have tested several case with different XOR degrees to show the efficiency of our rule engine. (i) in Figure 18 shows the results of our test. From the results, we see our test cases are smaller than the results shown in the paper duo to the less efficiency rule engine design. Besides that, since the author didn't give us the details to generate the test set here. We don't know the size of tuples and how many XOR =2, XOR=4, or XOR =8 tuples existed in the test cases. In this experiments, the size of tuples is 20. For XOR=2, there are 6 exclusive tuples pair in the case. For XOR =4, there are 2 exclusive tuples group with size 4 in the case. For XOR=8, there are 1 exclusive tuples group with size 8 in the test case.

The experiments of our implementation shows that for $IndepU - topk$ and $IndepU - kRanks$ algorithms, our efficient is very efficiency and achieve much better results than the original paper due to the different test platforms. However, For $OPTU - Topk$ and $OPTU - kRanks$, our method is less efficiency due to the less efficiency rule engine designed by ourself. Since the author didn't give the details to implement the efficient rule engine [2].

7 Others

The devision of labor of our implementation is as follows: We collaborate to determine the software framework, write report, and do peer review for each other.

In the coding phase, Bolong Zhang is responsible for realizing all core algorithms in processing layer, including U-Topk, U-kRanks, IndepU-Topk, and IndepU-kRanks.

Bolong Zhang also plot the algorithms' results for future use. Tao Wang is responsible for all the remaining work, including dataset generating, deal with MYSQL API, deal with MATLAB engine, and wxWidgets GUI developing.

Our source code, datasets, and runnable software could be downloaded at the following link: https://github.com/bolongz/utopk

REFERENCES

[1] Fuhr, Norbert. A probabilistic framework for vague queries and imprecise information in databases. Technische Hochsch., Fachgebiet Datenverwaltungssysteme II, 1990.
[2] Friedman, Nir, et al. "Learning probabilistic relational models." IJCAI. Vol. 99. 1999.
[3] Barbar, Daniel, Hector Garcia-Molina, and Daryl Porter. "The management of probabilistic data." IEEE Transactions on knowledge and data engineering 4.5 (1992): 487-502.
[4] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. "Top-k query processing in uncertain databases." Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007.
[5] Yi, Ke, et al. "Efficient processing of top-k queries in uncertain databases with x-relations." IEEE transactions on knowledge and data engineering 20.12 (2008): 1669-1682.
[6] TingJian Ge, Stan Zdonik, Samuel Madden. "Top-k Queries on Uncertain Data: On Score Distribution and Typical Answers". IEEE, 2009.
[7] Ilyas, Ihab F., George Beskales, and Mohamed A. Soliman. "A survey of top-k query processing techniques in relational database systems." ACM Computing Surveys (CSUR) 40.4 (2008): 11.
[8] M. Hua, J. Pei, W. Zhang, X. Lin. Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach. In SIGMOD, 2008.
[9] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: query algebra and optimization for relational top-k queries. In SIGMOD, 2005.