Top-*k* Queries on Uncertain Data: On Score Distribution and Typical Answers

Tingjian Ge
Computer Science Department
Brown University
Providence, RI, USA
tige@cs.brown.edu

Stan Zdonik
Computer Science Department
Brown University
Providence, RI, USA
sbz@cs.brown.edu

Samuel Madden
CSAIL
MIT
Cambridge, MA, USA
madden@csail.mit.edu

ABSTRACT
Uncertain data arises in a number of domains, including data integration and sensor networks. Top-*k* queries that *rank* results according to some user-defined score are an important tool for exploring large uncertain data sets. As several recent papers have observed, the semantics of top-*k* queries on uncertain data can be ambiguous due to tradeoffs between reporting high-scoring tuples and tuples with a high probability of being in the resulting data set. In this paper, we demonstrate the need to present the score distribution of top-*k* vectors to allow the user to choose between results along this score-probability dimensions. One option would be to display the complete distribution of all potential top-*k* tuple vectors, but this set is too large to compute. Instead, we propose to provide a number of typical vectors that effectively sample this distribution. We propose efficient algorithms to compute these vectors. We also extend the semantics and algorithms to the scenario of score ties, which is not dealt with in the previous work in the area. Our work includes a systematic empirical study on both real dataset and synthetic datasets.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems – *query processing.*

General Terms
Algorithms, Performance, Design, Experimentation.

Keywords
Top-*k*, Distribution, Uncertain data, Typical.

1. INTRODUCTION
The need to manage uncertain data arises in many applications. Some examples include data cleaning, data integration, sensor networks, pervasive computing, and scientific data management. For example, acoustic sensors (e.g., microphones) are often used to detect the presence of objects. Due to the nature of acoustic sensing, detections produced by microphones are often ambiguous, with an object possibly being at one of several locations. A common approach for storing such sensor data is to produce one record for each of the possible object locations, and assign a confidence (i.e., probability of existence in a table) to each record. Often, a query over such data has a large number of result tuples. In this context, top-*k* (i.e., ranking) queries have proven to be useful [11].

Unfortunately, the semantics of ranking in such systems are unclear, due to the fact that both scores and probabilities of tuples must be accounted for in the ranking. For example, it is unclear whether it is better to report highly ranked items with a relatively low probability of existence or a lower-ranked set of items with a high probability of existence. Thus, the definition of the semantics of top-*k* queries when the data is uncertain is an important issue. We next look at an example.

Tuple ID	Soldier ID	Time	Location	Score for Medical Needs	Conf.
T1	1	10:50	(10, 20)	49	0.4
T2	2	10:49	(10, 19)	60	0.4
T3	3	10:51	(9, 25)	110	0.4
T4	2	10:50	(10, 19)	80	0.3
T5	4	10:49	(12, 7)	56	1.0
T6	3	10:50	(9, 25)	58	0.5
T7	2	10:50	(11, 19)	125	0.3

Figure 1. A table generated by sensors monitoring soldiers' needs for medical attention. The Conf. (confidence) attribute is the probability of existence of the tuple.

Example 1. In the War-fighter Physiologic Status Monitoring application [19], the US military is embedding sensors in a "smart uniform" that monitors key biological parameters to determine the physiological status of a soldier. Under the harsh environment of the battlefield, it is crucial that sufficient medical resources reach wounded soldiers in a timely manner. Sensors in a smart uniform monitor thermal signals, hydration levels, cognitive and life signs, and wound levels. There are a few ways the soldier's physiological states can be estimated with different sensors and with different confidence. An algorithm computes an overall score indicating how much medical attention the soldier needs and how urgent his or her condition is. In a central database, as shown in Figure 1, a table records the information sent out by the sensors in the soldiers' uniforms. Each tuple in the table is one

SIGMOD'09, June 29–July 2, 2009, Providence, RI, USA.

estimate with some confidence. Sensors might be broken in harsh environments. For high availability, there can be two sets of sensors in a soldier's uniform in case one of them breaks down or loses precision. When each sends out an estimate at about the same time and they are inconsistent, at most one of them can be correct (together they form a discrete distribution with the confidence indicating the weight of each). These estimates may differ considerably due to variations in sensors, the possibility of lost network messages, and different estimation algorithms. □

Possible world	Prob.	Top-2
W1 = {T1, T2, T3, T5}	0.064	T3, T2
W2 = {T2, T3, T5}	0.096	T3, T2
W3 = {T1, T2, T5, T6}	0.08	T2, T6
W4 = {T2, T5, T6}	0.12	T2, T6
W5 = {T1, T2, T5}	0.016	T2, T5
W6 = {T2, T5}	0.024	T2, T5
W7 = {T1, T3, T4, T5}	0.048	T3, T4
W8 = {T3, T4, T5}	0.072	T3, T4
W9 = {T1, T4, T5, T6}	0.06	T4, T6

Possible world	Prob.	Top-2
W10 = {T4, T5, T6}	0.09	T4, T6
W11 = {T1, T4, T5}	0.012	T4, T5
W12 = {T4, T5}	0.018	T4, T5
W13 = {T1, T3, T5, T7}	0.048	T7, T3
W14 = {T3, T5, T7}	0.072	T7, T3
W15 = {T1, T5, T6, T7}	0.06	T7, T6
W16 = {T5, T6, T7}	0.09	T7, T6
W17 = {T1, T5, T7}	0.012	T7, T5
W18 = {T5, T7}	0.018	T7, T5

Figure 2. Possible worlds, their probabilities, and top-2 tuples in each world.

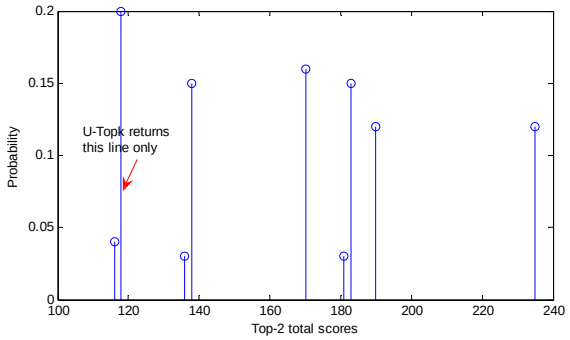


Figure 3. The distribution of top-2 tuples' total scores.

Figure 1 shows tuples that were reported around the same time and thus estimating the same value for each soldier. $T2$, $T4$, and $T7$ are readings for soldier 2 and are mutually exclusive (i.e., at most one of them can be correct), denoted as $T2 \oplus T4 \oplus T7$. Similarly, $T3 \oplus T6$. The last column of the table indicates the confidence of the estimates. Given such a table, a military staff may want to query for the top-k soldiers who require the most medical attention and allocate the appropriate resources to deliver to the battlefield. For this toy example, we simply look at the top-2 result. *Possible worlds* semantics has been used extensively in this context (e.g., [5]). Figure 2 shows the 18 possible worlds, their probabilities, and the top-2 tuples in each world according to the score attribute.

Recently, there has been some work on the semantics of top-k queries on uncertain data, starting from the work of Soliman, Ilyas, and Chang [18]. The proposed semantics roughly fall into two categories: (1) returning k tuples that *can co-exist* 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 (e.g., the probability that a tuple is top-k or at a specific rank in all possible worlds). The *U-Topk* [18] definition belongs to category (1) while the *U-kRanks* [18] and *PT-k* [9] definitions belong to (2). In this work, we propose an extension of the category (1) semantics.

The answer to a U-Top$k$ query is a tuple vector with the highest probability of being the top-k vector when we consider all possible worlds. For example, in Figure 2 we find that $\langle T2, T6\rangle$ has the highest probability ($\Pr(W3) + \Pr(W4) = 0.2$) of being in top-2 together and thus will be the result of U-Top$k$ (for $k = 2$). Notice that U-Top$k$ chooses a k tuple vector based purely on its probability. We observe three things:

1. Although a returned k-tuple vector has the highest probability p of being the top-k, p itself can be rather small (an obvious upper bound on p is the probability that all k tuples exist in the table), and it may not be much bigger than the probability of other top-k vectors.
2. The score distribution of the tuples is usually independent of the distribution of probability values of tuples.
3. U-Top$k$ does not take into consideration the distribution of the *scores* of all possible top-k tuple vectors.

As a result of these observations, the *total score* of a U-Top$k$ vector can be rather *atypical*, meaning that the score can vary dramatically from the expected score in the true top k. Figure 3 shows this fact for our toy example. For the U-Top$k$ vector ($k = 2$) $\langle T2, T6\rangle$, although it has the highest probability (0.2) of being in top-2, this probability is not much bigger than other top-2 vectors, and its total score (118) is atypical, for the following reasons:

1. With probability 0.76, the top-2 result has a higher total score than that of U-Top$k$;
2. With probability 0.12 (not much lower than 0.2), the top-2 total score (235) is about twice that of U-Top$k$; and
3. The expected top-2 total score is 164.1, substantially higher than the score of the U-Top$k$ vector.

In our soldier example, score is meant to signify the severity of injury. One could imagine that medical personnel might prefer to send resource to the units with score 235, as their injuries are presumably much more severe than those of the U-top$k$ units, and the probability of the score 235 group is not much less than the U-top$k$ group.

In Example 1, we intentionally limited the number of tuples and value of k so that it was feasible to list all possible worlds. But in reality k is often much bigger. We note that this problem with U-Top$k$ can be worse when k is bigger (i.e., $k > 2$). In other words, it is more likely that U-Top$k$ will return a vector with an atypical score for large k. This is because for a specific k-tuple vector to be U-Top$k$, all k uncertain tuples must appear in the first place, lowering the probability and increasing the likelihood that the score is atypical. More specifically, due to the "curse of dimensionality" [2], no top-k vector likely dominates many possible worlds (or has a significant probability). If we arbitrarily increase the score of a tuple that is not in the most probable top-k vector (e.g., increase the score of T_7 which is not in the most probable vector T_2, T_6), the U-Top$k$ result can be arbitrarily atypical.

This dilemma is analogous to the "typical set" concept in information theory [4]. Example 2 shows that a maximum probability event can be atypical.

Example 2. *Consider a biased coin with head probability 0.6 and tail probability 0.4. Suppose we toss it n = 20 times and assign a score as the number of heads. Clearly, the outcome with maximum probability is all heads (score 20) with probability* $(0.6)^{20} \approx 3.66 \times 10^{-5}$*, which is still a very small value. However, the score of this outcome is very atypical. In fact, with overwhelming probability (*$1-3.66\times10^{-5}$*) we get a score smaller than 20. It can be shown that we most likely get a score of 12, with probability* $(0.6)^{12} \cdot (0.4)^{8} \cdot \frac{20!}{12! \cdot 8!} \approx 0.18$. *It can further be proved that as n increases, with probability approaching 1, we would get score 0.6·n. The set of outcomes with this score is called the* typical set *[4].* □

We shall see more examples of atypical U-Top*k* answers in the experiments on *real* as well as *synthetic* datasets (Section 5).

We also note that category (2) definitions (U-*k*Ranks and PT-*k*) are not suitable for these kinds of applications because the tuples they return are based on their marginal distributions and do not follow the constraints of mutual exclusion rules (U-*k*Ranks may even return the same tuple multiple times if it is the most probable one for more than one rank position).

Let us step back and examine what the issue really is. The *complete* result of a top-*k* query on uncertain data, in fact, is a joint distribution on *k*-tuple vectors. If one were able to return such a joint distribution, it would represent a complete answer, and would provide users with a convenient representation of the tradeoff between probability and score from which they could select the results of interest. Unfortunately, a complete distribution is too expensive to compute, as well as to describe and return as the result. All existing definitions try to provide the most important information of such a distribution. Category (1) and (2) definitions are useful in different situations. Category (1) definitions are needed for scenarios that seek "compatible" *k* tuples (i.e., they can co-exist), which is required when, for instance, further inferences on the whole set of *k* tuples are performed, as in our examples. However, as we have observed, by simple selection of the highest probability, U-Top*k* may pick a *k*-tuple vector that has a highly atypical score. What we propose in this work is a simple two-fold solution:

(1) The application program can optionally retrieve the score distribution of top-*k* vectors at any granularity of precision (e.g., histograms of any bucket width).

(2) We propose a new definition *c-Typical-Topk* which returns *c* typical top-*k* tuple vectors according to the score distribution, where *c* is a parameter specified by queries. Intuitively, the *actual* top-*k*'s score should be close to one of the *c* vectors' score.

We then address the computational challenge of obtaining the score distribution of top-*k* vectors and selecting *c* typical vectors. For the score distribution, we first give two simple and naive algorithms that either explore the state space to reach top-*k* tuple vectors (*StateExpansion* algorithm) or iterate through all *k*-tuple combinations within a bounded set of tuples (*k-Combo* algorithm). These two algorithms establish a baseline for comparisons. We then present our main algorithm which is based on dynamic programming and is much more efficient than the naive algorithms. The presentation of the main algorithm starts with the basic framework and is then extended to handle more complex and realistic scenarios, namely *mutually exclusive* tuples and *score ties* for tuples. Score ties are common when the score is based on an attribute that does not have many distinct values, e.g., year of publication, number of citations, or even non-numeric attributes [7]. Note that extending the semantics and algorithms to score ties (i.e., non-injective scoring functions) for *uncertain data* can be non-trivial [22] (because a single possible world can now have multiple top-*k* vectors) and is not dealt with in previous work. Once we obtain the score distribution of top-*k*, using ideas similar to [8], we apply a two-function recursive approach resulting in another efficient dynamic programming algorithm to select *c* typical vectors for *c*-Typical-Top*k*.

We conducted systematic experiments on a real dataset of road delays in the greater Boston area as measured by the CarTel project team [10, 14], as well as a synthetic dataset. Through the experiments, we verify our motivation, study the performance of our algorithms, and observe interesting behaviors of the results with different characteristics of data. In summary, the contributions of this work are:

- A new semantics for presenting the answers of top-*k* queries on uncertain data to the applications.
- Efficient algorithms to compute the two entities that can be returned to applications: the score distribution of top-*k* vectors and the *c*-Typical-Top*k* answers.
- Extensions of our semantics and algorithms to the realistic scenario of ties in ranking scores.
- A systematic empirical study on a real world dataset and a synthetic dataset.

The remainder of the paper is organized as follows. In Section 2, we present the formal specification of returning score distribution and *c*-Typical-Top*k* and the semantics under score ties. We then cope with the computational challenges and develop efficient algorithms to compute score distributions in Section 3 and *c*-Typical-Top*k* in Section 4. Comprehensive experiments are conducted in Section 5. Finally, we survey related work in Section 6 and conclude in Section 7.

2. PROBLEM FORMULATION

In this section, we present our data model and formal definitions of the Top*k* score distribution and *c*-Typical-Top*k*.

2.1 Data Model and Scoring Function

We follow the well-known *tuple independent/disjoint* data model from the probabilistic database literature [6, 20, 18, 9]. In this data model, an uncertain database *D* contains uncertain tables. An uncertain table *T* has an extra attribute that indicates the *membership probability* of a tuple in T. If a tuple's membership probability is p ($0 < p \leq 1$), it has probability p of appearing in the table and probability $1 - p$ that it does not appear. Table *T* also has a set of mutual exclusion rules. Each rule specifies a set of tuples which we call an *ME group*, only one of which can appear in T. If a tuple has no mutual exclusion constraint, we simply say that it is in its own ME group (of size 1). The sum of the

probabilities of all tuples in an ME group should be no more than 1. The ME groups are assumed to be independent of each other.

A scoring function s takes a tuple t and return a real number $s(t)$ as its score. In the previous work, the scoring function s is assumed to be *injective* (i.e., each tuple maps to exactly one score, and no score is shared by two tuples), meaning that *ties are not allowed*. In many cases, it is non-trivial to extend the algorithms in the previous work to handle non-injective scoring functions; in fact, the result is undefined when there are ties in tuple scores. In this work, we remove that restriction and allow *non-injective* scoring functions.

2.2 Score Distribution and *c*-Typical-Top*k*

As discussed in Section 1, the scores of the k-tuple vector returned by U-Top$k$ can be rather atypical, severely restricting the usefulness of the U-Top$k$ result. We therefore propose to compute and provide the *distribution of the total scores of top-k tuples*. There are two possible usages of such a distribution:

(1) An application can access the distribution at any granularity of precision (e.g., histograms of any bucket width).

(2) An application can receive c typical top-k vectors (n.b., c-Typical-Top$k$, defined below), where c is a parameter specified by queries.

Intuitively, c-Typical-Top$k$ returns c top-k vectors (for $c \geq 1$) such that the *actual* top-k result (drawn according to its distribution) is close to at least one of the c vectors. When $c = 1$, the result has a score that is the expected score of top-k vectors; on the other hand, a big c value gives c vectors (and their probabilities) that approach the distribution of all top-k vectors. Put another way, the ith vector has a score that is approximately $i/(c+1)$ through the probability distribution of all possible scores.

Definition 1 (c-Typical-Top$k$ scores). *Let the distribution of the total scores of top-k tuples of an uncertain table T be a PMF (Probability Mass Function) D. We call the set of c scores $\{s_1, s_2, ..., s_c\}$, where s_i $(1 \leq i \leq c)$ has non-zero probability in D, the* **c-Typical-Top*k* scores** *if for a score $S \sim D$ (i.e., randomly chosen according to D),*

$$\{s_1, s_2, ..., s_c\} = \arg\min_{\{s_1,...,s_c\}} E[\min_{s_i \in \{s_1,...,s_c\}} |S - s_i|]$$ □

That is to say, over all choices of the c scores, for a random score S chosen according to D, $|S - s_i|$ is minimal in expectation, where s_i is the closest score to S among the c scores.

Definition 2 (c-Typical-Top$k$ tuples). *We call the set of k-tuple vectors $\{v_1, v_2, ..., v_c\}$, where v_i $(1 \leq i \leq c)$ is a vector of top-k tuples of T in some possible world, the* **c-Typical-Top*k* tuples** *if*

$$v_i = \arg\max_{s(v_i)=s_i} \Pr(v_i), 1 \leq i \leq c$$

where $s_1, s_2, ..., s_c$ are c-Typical-Topk scores, $s(v_i)$ is the total scores of the tuples in v_i, and $Pr(v_i)$ is the probability that v_i is a top-k tuple vector of T. □

In other words, v_i is the most probable top-k tuple vector that has a total score s_i (if there is more than one such vector, v_i can be any one of them).

For example, we can find that the 3-Typical-Top-2 scores of the table in Example 1 is {118, 183, 235}, with an expected distance 6.6 for a random top-2 vector. The 3-Typical-Top-2 vectors are {($T2$, $T6$), ($T7$, $T6$), ($T7$, $T3$)}. For comparison, the 1-Typical-Top-2 vector is ($T3$, $T2$), which has a slightly smaller probability (0.16) than that of the U-Top-2 vector ($T2$, $T6$) with probability (0.2), but has a much more typical score of 170, as opposed to 118 of the U-Top-2.

2.3 Non-injective Scoring Function and Ties

Now we consider the case in which the scoring function s is non-injective and there can be ties among the scores of the tuples of an uncertain table. Score ties are common when the score is based on an attribute that does not have many distinct values, e.g., year of publication, number of citations, or even non-numeric attributes [7]. It is also called *partial ranking* in [7], where the authors studied combining several ranked lists to produce a single ranking. We call the set of all tuples that have the same score a *tie group*. When a tuple does not have the same score with any other tuple, it is in a tie group of size one. A tie group in an uncertain table T contains all uncertain tuples that have the same score; a tie group in a possible world contains all tuples that appear in that world and have the same score.

We first discuss what this implies in a single possible world (i.e., without uncertainty). In a possible world w, as usual, a top-k tuple vector still contains a set of k tuples that have the highest scores. When there are ties, it is likely that there are multiple such top-k vectors in w, all ending in some tuples from a tie group. We say that a top-k vector v *contains* a tie group g if all tuples in g belongs to v. We say that a top-k vector v *partially reaches* a tie group g if at least one but not all tuples in g belong to v. We say that g *contributes m* tuples to v if exactly m tuples from g belong to v. We state the following theorem without proof.

Theorem 1. *In a possible world w, all top-k vectors must contain the same set of tie groups. If there is more than one top-k vector, they must all partially reach the same tie group g and g contributes the same number of tuples m to all those vectors. In fact, there are $\binom{|g|}{m}$ such vectors, where $|g|$ is the number of tuples in g.* □

Example 3. *We can order the tie groups according to their scores in descending order. Let us say that g_1 = {T2, T6}, g_2 = {T3, T7, T10}, and g_3 = {T5, T9, T12} are the three tie groups in a possible world with the highest scores. Among the three groups, g_1 has the highest score and g_3 has the lowest. Suppose we want to ask for the top-7 tuples. Then there are $\binom{3}{2} = 3$ top-7 tuple vectors {g_1, g_2, T5, T9}, {g_1, g_2, T5, T12}, and {g_1, g_2, T9, T12}, all containing g_1 and g_2 but partially reaching g_3. g_3 contributes 2 tuples to each vector.* □

It is clear that all top-k tuple vectors of a possible world have the same total score. Thus, in terms of the score distribution, ties would not have any impact: the probability of some score is still the sum of the probabilities of all possible worlds whose top-k vectors have that score. For c-Typical-Top$k$, among possibly multiple vectors that have some score, we choose one of them with the highest probability to appear in the uncertain table.

3. COMPUTING SCORE DISTRIBUTION OF TOP-*k*

A key challenge is to compute the distribution of the total scores of top-*k* tuple vectors. This is inherently computationally expensive because unlike U-Top*k* and U-*k*Ranks, this is not really a search problem (e.g., searching for the highest probability vector), as, in this case, one must account for all top-*k* vectors' scores and probabilities. The goal of such an algorithm is to output the distribution as a set of (*score value*, *probability*) pairs.

3.1 Two Simple Algorithms

We first present two algorithms which establish a baseline for comparison with the algorithm presented in Section 3.2 and 3.3. For now, we do not consider non-injective scoring functions and ties in tuples' scores; these will be discussed in Section 3.4. Figure 4 shows the first algorithm, called *StateExpansion*.

We first initialize the distribution to be an empty set (step 1). S is a set of states and we initialize it to contain one state – containing the empty tuple vector ε (step 2). We then go through all tuples in descending order by score, expanding each current state in S by either include the new tuple or not. When we reach k tuples for a state, we add it to the distribution to be returned (step 10). When the probability of a state gets too small (below a threshold p_τ as an input parameter), it is dropped. Note that the number of (*score*, *probability*) pairs in the output *dist* could potentially be very large. Thus, in step (10), we use a coalescing strategy to limit the size of the output.The details are described in Section 3.2.1. The *StateExpansion* algorithm has an exponential cost in the number of tuples considered (subject to the probability threshold).

We next show a more efficient algorithm. In this algorithm, we first determine an upper bound on the number of uncertain tuples that we have to examine when tuples are in rank order by score. A reasonable stopping condition is that we do not need to consider tuples that have probability less than p_τ being in top-k.

Theorem 2. *Given that we do not need to consider any tuple that has probability less than p_τ being in top-k, the stopping condition of the sequential scan of tuples in rank order by score is at a tuple t satisfying*

$$\mu \geq 1 + k + \ln\frac{1}{p_\tau} + \sqrt{\ln^2\frac{1}{p_\tau} + 2k\ln\frac{1}{p_\tau}}$$

(i.e., we do not need to consider any tuple from t onwards), where $\mu = \sum_{t' \in T(t)} \Pr(t')$ and T(t) is the set of all tuples ranked higher than t, except those in t's ME group. Furthermore, such a stopping condition also guarantees that no k-tuple vector with probability p_τ or more being in top-k is omitted.

Proof. We use an existing result from [9]. Theorem 8 of [9] says that a slightly different condition $\mu \geq k + \ln\frac{1}{p_\tau} + \sqrt{\ln^2\frac{1}{p_\tau} + 2k\ln\frac{1}{p_\tau}}$ ensures $Pr(t \text{ is in top-}k) < p_\tau$. We note that μ may not be monotonically increasing with more tuples because we have to exclude tuple t's ME group, which can vary from tuple to tuple. However, the sum of the probabilities of t's ME group is no more than 1. Thus, adding 1 to the right hand side of the inequality ensures that once the condition is satisfied at some tuple t, it will always be satisfied for all tuples onwards. We further observe that for any top-k vector v that contains t, because v is top-k implies t is in top-k, we must have $\Pr(v \text{ is a top-}k \text{ vector}) \leq \Pr(t \text{ is in top-}k) < p_\tau$. Thus, the stopping condition also guarantees that no k-tuple vector with probability p_τ or more being in top-k is omitted. □

Input: T: an uncertain tuple set in rank order,
p_τ: a probability threshold – note: a top-k vector with probability below p_τ need not be considered.
Output: The score distribution of top-k vectors.

```
(1)  dist = Φ
(2)  S = {ε}
(3)  for each t from T do
(4)     if S is empty then break end if
(5)     S' = Φ
(6)     for each state s in S do
(7)        Append t to s and get a new state s1.
(8)        Compute s1's score and probability based on s.
(9)        if s1 has k tuples then
(10)          Add its score and probability to dist.
(11)       else if s1's probability is greater than pτ then
(12)             S' = S' ∪ {s1}.
(13)          end if
(14)       end if
(15)       Append ¬t to s and get a new state s2.
(16)       Compute s2's probability.
(17)       if s2's probability is greater than pτ then
(18)          S' = S' ∪ {s2}.
(19)       end if
(20)    end for
(21)    S = S'
(22) end for
(23) return dist
```

Figure 4. Algorithm *StateExpansion*.

Theorem 2 gives us a stopping condition, which also satisfies the requirement in the *StateExpansion* algorithm (i.e., no k-tuple vector with probability p_τ or more being in top-k is missed). Note that we always stop at the end of a tie group because tuples in a tie group either all satisfy the stopping condition or none does. Let the number of uncertain tuples we need to consider be n. We can simply iterate through all k-combinations of the n tuples using a standard algorithm that generates all k-combinations in lexigraphical order [16], but exclude those that violate the mutual exclusion rules. For each k-combination, we can compute its total score and probability, and eventually we get the distribution. We call this algorithm *k-Combo*. Its cost is $O(n^k)$.

3.2 The Main Algorithm

We now present our main algorithm, which is based on dynamic programming. Our presentation is done in several steps. In this subsection (3.2), we introduce the basic framework of the algorithm. In Section 3.3 and 3.4, we extend this algorithm to handle mutually exclusive tuples and score ties, respectively.

Consider the table in Figure 5. The rows correspond to n (determined by Theorem 2) uncertain tuples in rank order by score. The columns are labeled from k to 1. A cell at row T_i column j contains the score distribution of top-j tuples starting from row T_i. Thus, our goal is to get the distribution in the cell at

the upper left corner of the table (marked with a "?"), i.e., the score distribution of top-k tuples starting from T_1. We first consider the basic case in which tuples are independent (i.e., no mutual exclusion rules) and there are no ties in score.

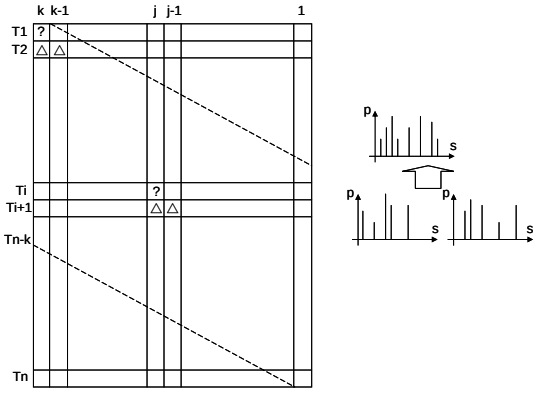


Figure 5. Illustrating the basic dynamic programming algorithm, as explained in the text below.

Our goal, the distribution of top-k starting from T_1 (upper left corner cell), can be composed using the distributions of two cells below it (marked with triangles in Figure 5): the distribution of top-k starting from T_2 (when T_1 does not exist) and the distribution of top-(k-1) starting from T_2 (when T_1 exists). In general, the distribution $D_{i,j}$ at row T_i and column j (top-j starting from T_i) is composed from the distribution $D_{i+1,j}$ at row T_{i+1} and column j (top-j starting from T_{i+1}) and the distribution $D_{i+1,j-1}$ at row T_{i+1} and column j-1 (top-(j-1) starting from T_{i+1}) in the following way:

(1) For each value and probability pair (v, p) in $D_{i+1,j}$, we transform it to (v, $p(1-p_i)$), where p_i is the probability that T_i exists.

(2) For each value and probability pair (v, p) in $D_{i+1,j-1}$, we transform it to ($v+s_i$, $p \cdot p_i$), where s_i is T_i's score and p_i is the probability that T_i exists.

(3) Merge the value and probability pairs resulting from (1) and (2) by taking their union except for the following: if two pairs have the same value, they become one pair with that value and with the new probability being the sum of the two original ones.

The right hand side of Figure 5 shows pictorially the merging process. Since *all* top-k tuples (there are k of them) must be among the n tuples T_1 to T_n, we only need to fill in the distributions in the table of Figure 5 between the two dotted lines. For example, we do not need to get the distribution of top-(k-1) starting from T_1; nor do we need top-2 starting from T_n, etc.

The recursive process described above fills in the table in a *bottom-up* manner. For the *boundary conditions* of the recursion, we add an *auxiliary* column 0 at the right border of the table. The distribution at a cell of column 0 has only one (*value*, *probability*) pair: (0, 1), i.e., score 0 with probability 1. For the a boundary cell (at row T_{n-i+1} and column i, for $i = 1, \ldots, k$) immediately above the bottom dotted line, its distribution also has only one (*value*, *probability*) pair: $\left(\sum_{j=n-i+1}^{n} s_j, \prod_{j=n-i+1}^{n} p_j \right)$.

In the algorithm we also keep track of one tuple vector for each (v, p) pair, which is needed for obtaining c-Typical-Top$k$. The vector is one (among possibly many) that has score v and has the highest probability of being the top vector. The recorded tuple vector is initially empty at column 0 and contains only T_n for the cell at row T_n and column 1. Thereafter, step (1) of the distribution merging process does not change the tuple vector while step (2) prepends T_i to the vector. In step (3), when two pairs have the same value and get combined, we keep the vector that has the higher probability.

3.2.1 The Need for Approximation

Thus far, it appears that the cost of this algorithm is $O(kn)$. However, there is one potential problem. For a cell at row T_i and column j (i.e., the distribution of the total scores of top-j starting from row T_i), there are $\binom{n-i+1}{j}$ possible combinations that make up the top-j scores ($1 \le i \le n$, $1 \le j \le k$). In the worst case, each combination has a distinct total score, resulting in a distribution that has the same number of discrete values (vertical lines in the PMF) in the cell. Thus, the number of vertical lines of a distribution is upper bounded by $\binom{n}{k}$, which is $O(n^k)$. Recall that the distribution merging process described above goes through each vertical line (v, p), increasing the worst case complexity of the main algorithm to $O(n^k)$. Note that in most applications, in reality, scores are not too far apart, and total scores of different combinations are often very close or even the same. Even if they were all distinct, it would often be unnecessary to keep all $O(n^k)$ lines in the PMF. It is more desirable to have a slight sacrifice in the accuracy of the distribution in exchange for a gain in efficiency. Imagine that the range of total scores of top-k is $[s_{min}, s_{max}]$. The range can be easily determined: s_{max} is the total score of T_1 to T_k and s_{min} is the total score of T_{n-k+1} to T_n since they are sorted. Note that the *span* $s_{max} - s_{min}$ is relatively insensitive to the problem size n. We divide the span into a constant number c' of same-size intervals (e.g., $c' = 200$). Each interval size is $\delta = (s_{max} - s_{min}) / c'$. Suppose for the application we can coalesce vertical lines that are no more than δ away from each other in the distribution (i.e., differ by no more than δ in total scores). Then the cost to describe the output distribution is a constant.

We call the distribution at row T_1 and column k (i.e., upper left corner) the *final* distribution and those at other cells *intermediate* distributions. We can have a "line coalescing" strategy as follows. At any intermediate or final distribution, whenever the algorithm results in more than c' vertical lines, (1) pick two lines that are closest to each other and coalesce them into one: the score value is their average and the probability is their sum; (2) repeat the first step until we have c' vertical lines. As for the recorded top vector, when we coalesce two lines, we keep the tuple vector that has the higher probability.

We first observe that in the bottom-up process of computing the dynamic programming table of Figure 5, two lines (v_1, p_1) and (v_2, p_2) in an intermediate distribution are always going to change in a synchronized way: either they both stay at the same scores (step 1 of the distribution merging process) or the two lines get "shifted" with the same offset by adding the same score (step 2 of the merging process). In both cases their probabilities are scaled by the same factor. Thus, coalescing two lines in an intermediate

distribution effectively is equivalent to coalescing them in the final distribution since they would have the same distance in scores, had we not coalesced them in any of the intermediate distributions.

Secondly, it is not hard to see that the span of any intermediate distribution is no more than that of the final distribution ($s_{max} - s_{min}$). This is because intermediate distributions either only consider top-j ($j < k$) or they use a subset of the n tuples. Thus, if an intermediate distribution has more than c' lines, by picking the two lines with minimum distance, we must be coalescing two lines that are no more than δ apart.

Now given that we have a constant cost of distribution merging, our basic algorithm so far has $O(kn)$ time complexity. In the next two subsections, we extend our basic algorithm to more complex and realistic scenarios in which there are mutual exclusion rules and possible score ties among tuples.

Note that we do this line coalescing similarly for the *StateExpansion* and *k-Combo* algorithms in Section 3.1 as well. For example, in step (10) of *StateExpansion*, we make sure *dist* has no more than a constant number of score/probability pairs. This, however, does not change the complexity of those two algorithms.

3.3 Handling Mutually Exclusive Rules

The problem gets more complicated when there is correlation among the tuples. We now describe how to handle mutually exclusive tuples. The original algorithm would not work in the presence of mutually exclusive tuples because the final distribution would be wrong if more than one tuple in an ME group simultaneously contributes to a top-k score.

3.3.1 Two False Starts

In the bottom-up dynamic programming algorithm, one might first be tempted to do the bookkeeping of which ME groups have contributed a tuple to a score (and with what probability). In this case, we do not add additional tuples from those ME groups into the intermediate distributions. Unfortunately, this is combinatorial and is too costly.

Another approach compresses all tuples in a mutually exclusive set into one tuple. We use the terminology in [9] and call it a *rule tuple*. A rule tuple has a composite score and a probability of the sum of the original tuples. At a row of a rule tuple, step (1) of the distribution merging process stays the same and step (2) changes to adding each score/probability of the original tuples of the rule separately. For example, if a rule tuple has three original tuples, we do step (2) three times. However, the problem with this approach is that we have nowhere to place the rule tuple in the dynamic programming table since it has a composite score. Wherever we place it, we are unable to compute the probability of a top-k score correctly because we have lost the information of exactly which original tuples appear (or do not appear) in a strict score order.

3.3.2 A Good Start

Although the second strategy above fails, it provides the following inspiration: suppose we require that the last tuple (i.e., the k-th) of the top-k *has to be* T_n, then the tuples in the dynamic programming table can be in any *arbitrary* order (i.e., they do not have to be ordered by scores as stated earlier). This is because for any tuple i with a score higher than the last tuple of the top-k, if i is in the top-k, we simply multiply the current probability by its probability p_i ; if i is not in top-k, we multiply by $(1 - p_i)$. The earlier order requirement simply prevents us from multiplying the $(1 - p_i)$ for any tuple i with a score *smaller* than the *last one* in top-k. But if the last one in top-k is T_n , we know for sure all other tuples have a higher score. Now without the order constraint, we can then modify the original tuples in the following way:

(1) Remove all other tuples (if any) that are in the same ME group as T_n from the table.

(2) Compress all other ME groups into rule tuples and leave them in any order. Remember the constituent original tuples' scores and probabilities for a rule tuple. A rule tuple also has a probability that is the sum of those of the constituent tuples.

The next trick ensures that the dynamic programming algorithm only considers the top-k vectors that end with T_n. Recall that we added an auxiliary column 0 at the right border of the dynamic programming table of Figure 5. Each cell in column 0 holds a distribution (0, 1) – score 0 with probability 1. We call a cell in column 0 an *exit point* because it indicates that we do not need to select any more tuples as top-k from that tuple and below. In order to only incorporate top-k vectors that end with T_n, all we need to do is simply "block" those exit points by letting them have a distribution of (0, 0) instead – score 0 with probability 0. It can be easily verified that such a distribution cannot be propagated by the distribution merging process. With that change, the dynamic programming algorithm can proceed as before.

The change on the distribution merging process to the main algorithm is the same as that described in the second attempt in Section 3.3.1.

What we have achieved so far is only the distribution of total scores of top-k vectors that end with T_n. To get the distribution for all top-k vectors, an easy extension is simply to repeat this for each tuple from T_k to T_n (i.e., truncate the dynamic programming table at each of those tuples and treat them as the last tuple of the top-k, respectively) and then we merge all the final distributions together. For a truncated table, an ME group may be truncated as well. That is, if the table is truncated at T_i ($k \le i \le n$), an ME group now only contains tuples in the remaining table (i.e., from T_1 to T_i). The compression step now applies to the reduced ME groups.

3.3.3 Refinement

It turns out that we can do better than the simple extension above. We call a tuple a *lead tuple* if it is the first one (i.e., with the highest score) in an ME group. If an ME group has only one tuple (i.e., not mutually exclusive with any other tuple), that tuple is a lead tuple. In a score-sorted sequence T_1 to T_n, a maximal contiguous subsequence of lead tuples T_i, T_{i+1}, …, T_j is called a *lead tuple region*. For a subsequence to be maximal, it must be satisfied that (1) either $i = 1$ or T_{i-1} is not a lead tuple; and (2) either $j = n$ or T_{j+1} is not a lead tuple.

We can see that we do not need to do the dynamic programming procedure for each tuple. Instead, we only need to do it once for every lead tuple region and once for every non-lead tuple. This is

because when the dynamic programming table ends with a lead tuple region, tuples in it behave exactly as independent tuples and they will not interfere with any other tuples above. Thus, for a lead tuple region, we can simply do one dynamic programming to get the score distribution of top-k vectors that end with *any* tuple in that lead tuple region. We achieve this by setting the boundary conditions properly. For the distributions in the cells of the auxiliary column 0, we set it to be (0, 1) at the rows of a lead tuple region in question and set it to be (0, 0) for other rows. Recall that (0, 0) is to block an exit point and (0, 1) is to enable it. Everything else, including the rule tuple compression, stays the same. This is illustrated in Figure 6.

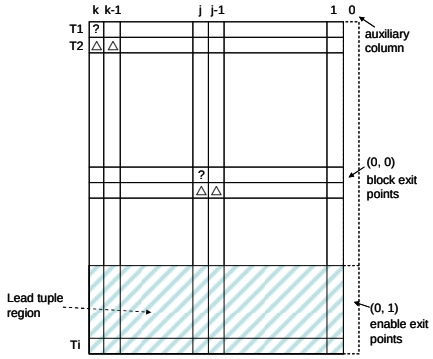


Figure 6. One dynamic programming for a lead tuple region.

With this improvement we can see that the time complexity of our algorithm that handles mutually exclusive tuples is $O(kmn)$, where m is the number of tuples (among T_1 to T_n) that are mutually exclusive with other tuples. In many applications, mutually exclusive tuples are only a small proportion of the total. The computational cost is proportional to this fraction.

3.4 Handling Ties

In many real applications, the scoring function s is non-injective which leads to ties among the tuple scores [7]. We discussed the semantics of top-k vectors and score distributions at the end of Section 2. We now extend the dynamic programming algorithm that we have developed so far to take care of the case of score ties. We shall prove that the following simple extension of the algorithm satisfies our requirements:

> Recall that before, the sort order was on scores. Now, sort tuples in descending order by (*score*, *probability*). When two tuples have the same score, they are in descending order of probability; when they have the same probability as well, break ties arbitrarily.

Aside from this adjustment, the algorithm works the same as before. The next theorem shows that this modification is correct.

Theorem 3. *With the above extension to the dynamic programming algorithm, we achieve our two goals: (1) we obtain the correct final score distribution of top-k and (2) among vectors that have the same score, the one that is captured at the end of the algorithm is the one with the highest probability.*

For the proof of Theorem 3, we first need the following definition and lemma.

Definition 3 (Configuration of top-k). *A* configuration of top-k *is a set of $(k - g)$ uncertain tuples plus g tuples from a tie group in non-increasing score order, with the ending tie group having the lowest score (the $k - g$ tuples are not in that tie group).*

Note that a configuration has a fixed total score and two configurations may have the same total score. The *probability of a configuration* is the probability that such a configuration is the top-k tuple vector.

Lemma 1. *Let A be the set of $(k - g)$ uncertain tuples and T be the ending tie group of a configuration. Let B be the set of tuples that have higher scores than those in T but are not in the configuration. The probability of the configuration is the probability that (1) tuples in A appear, and (2) those in B do not, and (3) at least g tuples from T appear.*

Proof (Lemma 1). Clearly, (1) and (2) must be true for the configuration to be top-k. Except for the case that fewer than g tuples from T appear, this configuration will be top-k. Thus, we have (3). □

Proof (Theorem 3). A top-k score distribution is made up of different configurations. Therefore, to prove goal (1) of Theorem 3, we only need to show that our algorithm computes the probability correctly for each configuration.

For the ending tie group T of a configuration, our algorithm puts the tuples in probability descending order. In fact, we can see that for any arbitrary order, as long as it is fixed, the dynamic programming will compute the probability of the configuration correctly. Let the ending tie group T have t tuples in total: T_1, T_2, …, T_t in some fixed order. The event (3) in Lemma 1 (i.e., at least g tuples from T appear) can be decomposed into $\binom{t}{g}$ sub-events as follows. Imagine a t-bit binary string. We choose g bits and set them to 1; the other bits are all 0. Clearly there are $\binom{t}{g}$ such strings. We use each of them to construct a sub-event: we truncate the string at the last 1 bit; then starting from the 1st bit until the last bit (which is 1), if the i'th bit is 1 (or 0), we add "T_i appears" (or "T_i does not appear", respectively) into the sub-event. It is easy to see that the dynamic programming procedure computes the probability of each such sub-event and adds them up to be the probability of the event (3) in Lemma 1. Thus, the algorithm computes the probability of the configuration correctly and we finish the proof of goal (1) of Theorem 3.

Example 4. *Consider the scenario that the first seven uncertain tuples are:*
$(T_1, 10, 0.5)$,
$(T_2, 8, 0.3)$, $(T_3, 8, 0.2)$, $(T_4, 8, 0.1)$,
$(T_5, 7, 0.5)$, $(T_6, 7, 0.4)$, $(T_7, 7, 0.2)$.
That is, T_1 has score 10 and probability 0.5, and so on. Consider a top-5 configuration c that includes T_1, T_2, T_4, and two tuples from the tie group $g = \{T_5, T_6, T_7\}$. Then

$$Pr(c) = Pr(T_1)Pr(T_2)(1-Pr(T_3))Pr(T_4)Pr(\geq 2 \text{ tuples in } g \text{ appear})$$

We can compute that $Pr(\geq 2$ tuples in g appear$) = 0.5 \cdot 0.4 \cdot 0.2 + 0.5 \cdot 0.4 \cdot (1-0.2) + 0.5 \cdot (1-0.4) \cdot 0.2 + (1-0.5) \cdot 0.4 \cdot 0.2 = 0.3$. On the other hand, our dynamic programming algorithm will calculate the probability of this part of c to be: $0.5 \cdot 0.4 + 0.5 \cdot (1-0.4) \cdot 0.2 +$

(1−0.5)·0.4·0.2 = 0.3 as well. Thus, our algorithm computes the probability of the configuration c correctly.

We next show that our algorithm achieves goal (2), i.e., the vector recorded is the one with the highest probability. Note that the algorithm may not compute the probability correctly for all vectors in a top-k configuration, but it does compute it correctly for the one with the highest probability, due to the fact that we order the probability in non-increasing order in the ending tie group. In Example 4, our algorithm computes the probability of the vector that ends with T_5 and T_6 correctly: 0.5·0.4 = 0.2 (for the part in tie group g). On the other hand, for the vector ending with T_5 and T_7, the algorithm computes 0.5·(1−0.4)·0.2 = 0.06, but the actual probability should be 0.5·0.2 = 0.1. This is fine because we only need to return the vector that has the maximum probability.

Note that the extension of our algorithm to handle mutually exclusive tuples as discussed in Section 3.3 would not affect the results of our proof above. This is because for a given configuration of top-k, after removing tuples in set T that are mutually exclusive with any tuple in set A (sets T and A as defined in Lemma 1), our proof holds in the same way. This concludes the proof of Theorem 3. □

It is not hard to see that the same method can be applied to the algorithm *StateExpansion* in Section 3.1 as well to handle score ties: we just need to sort the tuples in (score, probability) descending order.

4. COMPUTING *c*-TYPICAL-TOP*k*

Given a distribution of the total scores of top-k vectors as computed in Section 3, we now study how to compute c-Typical-Top$k$ vectors. We first formalize the problem. Let the score distribution be $\{(s_1, p_1), (s_2, p_2), \ldots, (s_n, p_n)\}$ and each score s_i ($1 \le i \le n$) is associated with a top-k tuple vector v_i. The vector v_i is the one with the highest probability of being top-k, among those having the same total score. Our goal is to choose from the n vectors and output c of them such that their scores satisfy the optimality requirement in Definition 1. We call s_i a *typical score* if its vector is chosen by the algorithm.

Using ideas similar to [8], we can derive an efficient $O(cn)$ time dynamic programming algorithm to solve this combinatorial optimization problem. We use a two function recursive approach. Let $F^a(j)$ be the optimal objective value of the subproblem reduced to the set $\{s_j, \ldots, s_n\}$, for $j = 1, \ldots, n$, where a is the maximum number of typical scores and let $G^a(j)$ be the respective value for the same subproblem, provided that s_j is a typical score. We have, for $j = 1, \ldots, n$, and $a \le c$,

$$F^a(j) = \min_{j \le k \le n} \left[\sum_{b=j}^{k} p_b (s_k - s_b) + G^a(k) \right] \quad (1)$$

$$G^a(j) = \min_{j < k \le n+1} \left[\sum_{b=j}^{k-1} p_b (s_b - s_j) + F^{a-1}(k) \right] \quad (2)$$

In equation (1), k iterates over the possible *first* typical score's positions, and in (2), k is the first position that is closest to the *second* typical score (i.e., s_j to s_{k-1} are closest to the first typical score, s_j). The solution for our original problem is thus given by $F^c(1)$. The boundary conditions are

$$G^1(j) = \sum_{b=j}^{n} p_b (s_b - s_j),\ j = 1, \ldots, n, \quad F^a(n+1) = 0,\ a \ge 1 \quad (3)$$

Input: A top-k score distribution (s_i, p_i, v_i), $1 \le i \le n$, where s_i is a score, p_i is its probability, and v_i is a top-k tuple vector that has score s_i and has the highest probability; an integer c

Output: c tuple vectors that are c-Typical-Top$k$.

```
(1)  P[0] = PS[0] = 0
(2)  for j = 1 to n do
(3)     P[j] = P[j-1] + p_j
(4)     PS[j] = PS[j-1] + p_j*s_j
(5)  endfor
(6)  for j = 1 to n do
(7)     G[1][j] = 0
(8)     for b = j to n do
(9)        G[1][j] = G[1][j] + p_b*(s_b − s_j)
(10) endfor endfor
(11) for a = 1 to c do
(12)    F[a][n+1] = 0
(13) endfor
(14) a = 1
(15) for j = 1 to n do
(16)    F[a][j] = MAX_DOUBLE
(17)    f[a][j] = 0
(18)    for k = j to n do
(19)       tmp = (P[k] − P[j-1])*s_k − PS[k] + PS[j-1]+G[a][k]
(20)       if tmp < F[a][j] then
(21)          F[a][j] = tmp
(22)          f[a][j] = k
(23) endfor endfor endif
(24) for a = 2 to c do
(25)    for j = 1 to n do
(26)       G[a][j] = MAX_DOUBLE
(27)       g[a][j] = 0
(28)       for k = j+1 to n+1 do
(29)          tmp = PS[k-1] − PS[j-1] − (P[k-1] − P[j-1])*s_j +
           F[a-1][k]
(30)          if tmp < G[a][j] then
(31)             G[a][j] = tmp
(32)             g[a][j] = k
(33)    endfor endfor endif
(34)    Do the for loop between line (15) and (23)
(35) endfor
(36) k = 1
(37) for a = c down to 1 do
(38)    i = f[a][k]
(39)    output v_i
(40)    k = g[a][i]
(41) endfor
```

Figure 7. The algorithm to select *c*-Typical-Top*k*.

We define, for $j = 1, \ldots, n$,

$$P(j) = \sum_{b=1}^{j} p_b, \quad PS(j) = \sum_{b=1}^{j} p_b s_b \quad (4)$$

Then we can rewrite (1) and (2) as

$$F^a(j) = \min_{j \le k \le n} \left[(P(k) - P(j-1)) s_k - PS(k) + PS(j-1) + G^a(k) \right] \quad (5)$$

$$G^a(j) = \min_{j < k \le n+1} \left[PS(k-1) - PS(j-1) - (P(k-1) - P(j-1)) s_j + F^{a-1}(k) \right] \quad (6)$$

With some preprocessing of (4) that takes $O(n)$ time, we can first get all $P(j)$ and $PS(j)$ values. Then the dynamic programming algorithm based on (5) and (6) will just take $O(cn)$ time.

We show the algorithm in Figure 7. We first pre-compute all the $P(j)$ and $PS(j)$ values (lines 1 to 5). Lines 6 to 13 set the boundary conditions according to Equation (3). Lines 14 to 35 iteratively apply Equation (5) and (6) in turn to fill in the two dynamic programming tables (i.e., all F and G values). Note that f and g values (lines 22 and 32) keep track of the k values that minimize the r.h.s. of Equations (5) and (6). This is needed to trace back and output the c typical top-k tuple vectors (lines 36 to 41).

Note that a user could examine the edit distances [24] between the vectors and potentially try different values of c. Once the dynamic programming that computes the score distribution is performed, the database system has finished the majority of the computation and contains the whole distribution and associated vectors. Changing the c value only implies a re-computation of the algorithm in this section, which is much cheaper. The magnitude of the distances indicates the span of the k-dimensional vector space. Smaller distances indicate that the result is less uncertain while bigger distances indicate larger uncertainty. Depending on the application, a user can do different things with the result. For example, in the Soldier Physiologic Status Monitoring application (Example 1), medical personnel would probably examine the high score range of the distribution since a score indicates the severity of injury. A different application might weight the probability values more.

5. EMPIRICAL STUDY

In this section, we conducted a systematic empirical study addressing the following questions:

- What does the score distribution of top-k tuple vectors look like for real-world data? Furthermore, where does the U-Top$k$ vector stand in the distribution, and where do c-typical vectors stand in the distribution?
- What is the performance of our main algorithm that computes the score distribution? How does it compare with *StateExpansion* and *k-Combo*? What are the *scan depth* (i.e., the number of tuples n that need to be read by our algorithms) values for various k values as determined by Theorem 2? How does the proportion of mutually exclusive tuples affect performance? By trading off accuracy for performance, how does the line coalescing strategy presented in Section 3.2 improve performance?
- What is the impact on score distribution and typicality of U-Top$k$ as we alter the following system parameters: (1) the correlation between scores and confidence, (2) the score range (variance), (3) the score range within ME groups and the size of ME groups?

5.1 Setup and Datasets

We performed the study using the following two datasets:

- A real-world dataset collected by the CarTel project team [10]. It consists of measurement of actual traffic delays on roads in the greater Boston area performed by the CarTel vehicular testbed [14], a set of 28 taxis equipped with various sensors and a wireless network.
- A synthetic dataset generated using the R-statistical package [23]. With the synthetic dataset we can control the various parameters of the data and study their impact on results.

We implemented all the algorithms presented in this paper and the U-Top$k$ algorithm presented in [18] to study the results. All the experiments were conducted on a 1.6GHz AMD Turion 64 machine with 1GB physical memory and a TOSHIBA MK8040GSX disk.

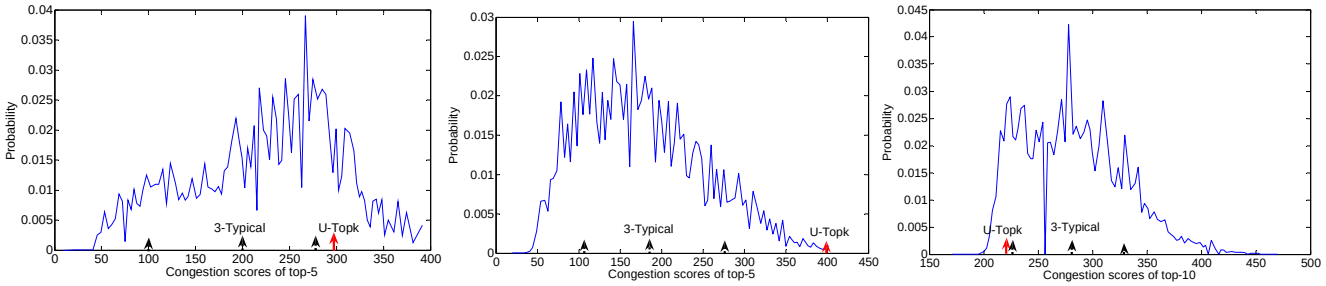
5.2 Results on the Real-world Dataset

In the first experiment, we examine the score distribution of top-k tuple vectors as computed by the main algorithm presented in the paper using the CarTel data. We execute the following query over some random areas taken from the whole dataset:

SELECT segment_id,
speed_limit / (length / delay) **AS** congestion_score
FROM area
ORDER BY congestion_score **DESC**
LIMIT k

Each tuple of the relation *area* is a measurement record of the actual travel delay of a road segment. In this query, we define $congestion\_score = \frac{speed\_limit}{length/delay}$, where the denominator is the actual travel speed and the numerator is the speed limit of the road segment. Thus, the congestion score is an indication of the travel speed degradation at a road segment (up to a constant factor: in the dataset, the *speed_limit* is in km/hour while the *length* is in meters and *delay* is in seconds). A higher congestion score implies a more congested road segment. The query selects the top-k most congested road segments in an area (say, a city). City planners might want to first locate the k most congested roads and their total (or equivalently, average) scores to give them an idea of how serious the situation is. For example, when the total scores exceed some threshold, the city planners will spend some funding to fix the traffic problem on the most congested road segments (e.g., by adjusting traffic light cycles, adding parallel roads or widening existing ones). Each road segment contains one or more measurement record. In general, each record is considered uncertain and the delay of a road segment is probabilistic [14]. If a road segment contains multiple measurements, we bin the samples and collect the statistics of the frequencies of the bins and obtain a discrete distribution, in which each bin is assigned a value that is the average of the samples within the bin. Bins in a distribution are mutually exclusive so that at most one of them may be selected in a possible world. Thus, a top-k tuple vector always contains distinct road segments.

Figure 8 shows the distributions of the total congestion scores of top-k roads at three random areas from the dataset. We use our main algorithm presented in Section 3.2 to 3.4 to compute the score distributions and the algorithm in Section 4 to compute c-Typical-Top$k$. We also examine where the resulting vector from the U-Top$k$ algorithm [18] stands in the distribution. We show the U-Top$k$ result as a solid (red) arrow and the three dotted arrows are 3-Typical-Top$k$ results. The height of an arrow roughly indicates the probability of the corresponding k-tuple vector. We



(a) (b) (c)

Figure 8. Congestion score distribution of top-*k* tuple vectors in three random areas and the results of U-Top*k* and 3-Typicals.

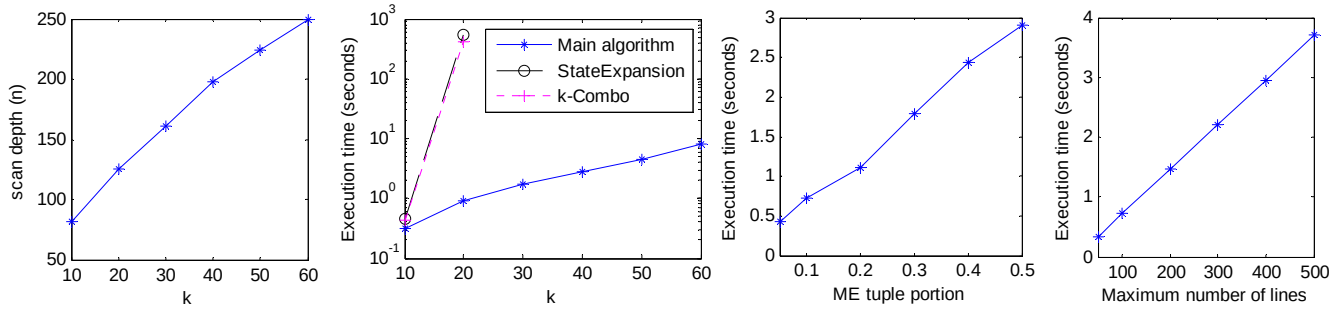


Figure 9. *k* vs. scan depth (*n*) **Figure 10. *k* vs. execution time** **Figure 11. ME portion vs. time** **Figure 12. # of lines vs. time**

can see that in all three subplots, the score of the U-Top*k* result is rather atypical. In Figure 8 (a) and (b) it is higher than the three typical scores while in Figure 8 (c) it is lower. Although being the highest probability vector, the U-Top*k* result still has a very small probability, and it may only be slightly bigger than many other *k*-tuple vectors. By the definition of *c*-Typical-Top*k*, the *actual* top-*k* vector (drawn according to its distribution) is more likely to have a score that is close to one of the *c* typical vectors. Informed by the score distribution and typical vectors, the city planners will have a much more accurate picture of how serious the top-*k* most congested road segments are.

5.3 Performance on the Real-world Dataset

In the second experiment, we examine the performance of our algorithms. We run the same query as shown in Section 5.2, but try different system parameters. Since the performance of both our main algorithm and *k*-Combo relies on the scan depth *n* as determined by Theorem 2, it is interesting to study what are the actual values of *n* for various *k*'s with the real-world dataset. We set p_τ to be 0.001. Figure 9 shows the result that *n* grows roughly linearly with *k* as is expected from the theorem.

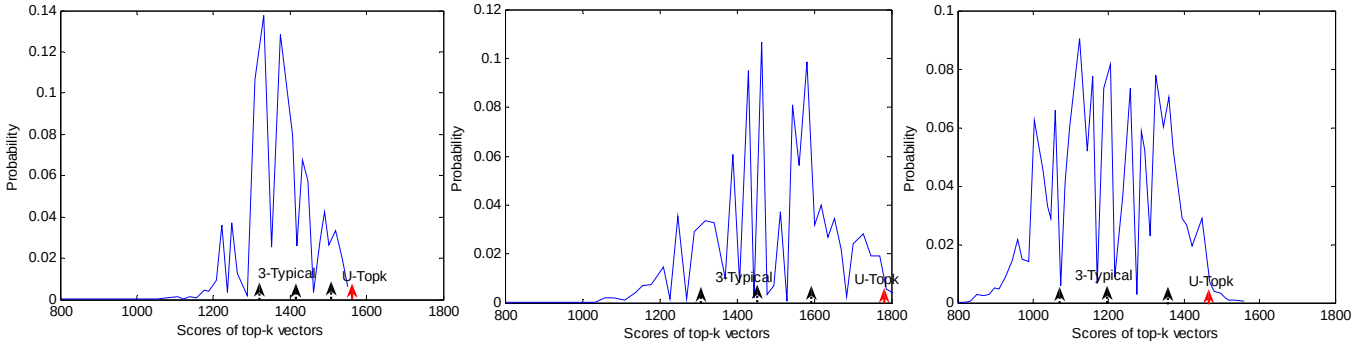
We next compare the performance of our main algorithm that computes the score distribution with the two simple algorithms presented in Section 3.1, namely *StateExpansion* and *k-Combo*. For all three algorithms, we limit the number of lines in the output distribution to be more than 100. We try different *k* values in the query and compare the execution times of the three algorithms, as shown in Figure 10. We can see that both *State-Expansion* and *k-Combo* have an exponential growth on the running time as *k* increases, with *k-Combo* being slightly better. On the other hand, our main algorithm which uses dynamic programming techniques is significantly more efficient.

Next we examine the performance of our main algorithm as we vary the portion of mutually exclusive tuples by first selecting a subset of road segment records and run our query against it. The result is shown in Figure 11. As expected, the computation cost increases as we increase the portion of tuples that are mutually exclusive with other tuples, as discussed in Section 3.3.

Finally, recall that in Section 3.2 we devised a line coalescing strategy in order to trade off accuracy for performance. The parameter here is the maximum number of lines allowed in the distributions. We vary this parameter from 50 up to 500 and the result is shown in Figure 12. We can see that the runtime varies linearly as the number of lines grows. The reason is that as the dynamic programming algorithm progresses bottom-up, very soon line coalescing takes effect, and the amount of computation thereafter is proportional to the number of lines in the distributions.

5.4 Results on the Synthetic Dataset

In this section, we use synthetic datasets because they give us control over various characteristics of the data. We further examine the impact of different kinds of data on score distribution and on how typical U-Top*k* results are. We first study different correlations between score and probability of tuples. We generate scores and probabilities as bivariate normal distributions with different correlation coefficients for the cases of independence ($\rho = 0$), positive correlation (we use $\rho = 0.8$), and negative correlation (we use $\rho = -0.8$). We show the top-10 results for these three cases in Figure 13 (a), (b), and (c) respectively. We



(a) (b) (c)

Figure 13. Score distribution of top-10, U-Top*k*, and 3-Typical for different score & probability correlations: ρ=0 (a), 0.8 (b), -0.8(c).

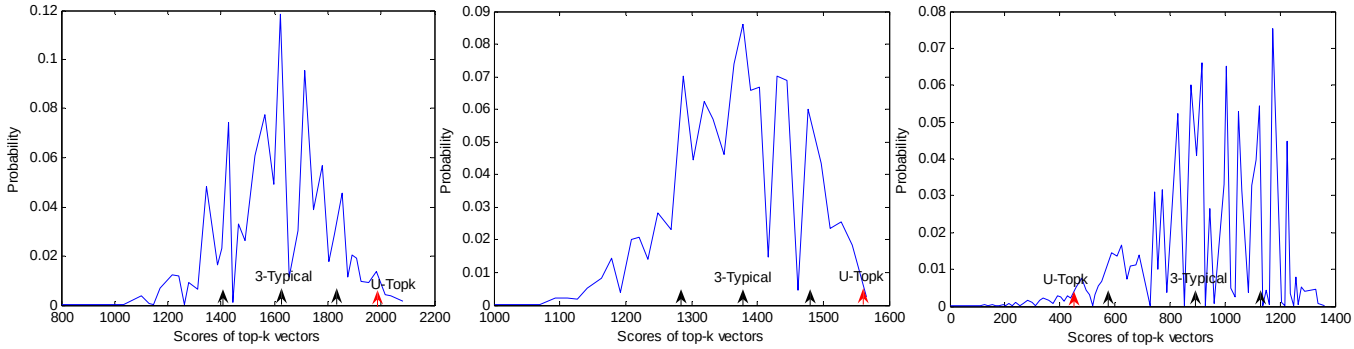


Figure 14. ρ=0, but increase σ to 100. **Figure 15. Increasing gaps between ME tuples.** **Figure 16. Increasing sizes of ME groups**

can see that compared to the independence case (Figure a), a positive correlation between scores and probabilities shifts the score distribution of top-*k* vectors to the right (Figure b) while a negative correlation shifts it to the left (Figure c). This is because if leading tuples (with higher scores) are more likely to exist, they are also more likely to be in top-*k*, thus making the total scores of top-*k* tuples higher. Moreover, we also observe here that in all three cases, the U-Top*k* result is atypical.

We next study how the results change when we alter the range (i.e., variance) of scores in the table. In the previous experiment in Figure 13, we use a bivariate normal distribution with the standard deviation of the scores being 60. With other parameters being the same as in Figure 13a (i.e., ρ = 0), we only increase the standard deviation of the scores σ to be 100. The result is shown in Figure 14. It is clear that the distribution of the total scores of top-*k* vectors now covers a wider range, with the *span* of the significant portion of the distribution increased from around 350 (Figure 13a) to around 1000, making the distance between U-Top*k* score and typical scores farther apart.

Finally, we examine the impact on the results as we vary the mutual exclusion (ME) group settings. With everything else being the same as in Figure 13a (ρ=0, σ=60), we only change the score gaps between two ME tuples. Without changing any scores in the table, we only change the assignment of the tuples to ME groups: we change the distance between two neighboring tuples in an ME group from d_1 tuples to d_2 tuples, where d_1 is a random number from 1 to 8 and d_2 is a random number from 1 to 40. The result is shown in Figure 15. We observe that there is no noticeable change from Figure 13a. However, when we increase the size of ME groups from s_1 to s_2 where s_1 is a random number of either 2 or 3, and s_2 is a random number from 2 to 10, there are some obvious changes in the results, as shown in Figure 16. First of all, we observe that the score distribution of top-*k* vectors covers a much wider range but with smaller values. The bulk of the distribution is at [200, 1350] compared to the original range of [1150, 1550] (Figure 13a), almost three times in width. The reason is that because we can only take at most one tuple from each ME group to include in top-*k*, a larger ME group implies that we end up scanning more tuples and lower scored tuples have a higher chance to be in top-*k*, which in effect increases the variance of the scores of tuples that contribute to the distribution. Secondly we observe that because each ME group now contains a lot more tuples with small probabilities (they must add up to no more than 1), we essentially have an exponential growth in possible top-*k* vectors, all have small probabilities. This makes U-Top*k* (which seeks the highest probability) more unstable or atypical. Figure 16 shows that in this case the U-Top*k* result shifts to the lower end of the score distribution.

6. RELATED WORK

There has been significant work on uncertain data management lately due to the importance of emerging applications (e.g., [5, 20, 3, 17, 1, 12]). Re et al. [15] studied top-*k* queries on uncertain data where the ranking is based on the probability that a result tuple appears in the result. The semantics of top-*k* queries on uncertain data with arbitrary ranking functions was first studied by Soliman et al. [18]. The authors in [18] gave two kinds of semantics (U-Top*k* and U-*k*Ranks) and devised optimal algorithms in terms of the number of accessed tuples and search

states. Yi et al. [21] improved the time and space efficiency of the algorithms that compute U-Top*k* and U-*k*Ranks results. Hua et al. [9] proposed a new semantics called probabilistic threshold top-*k* (PT-*k*). More recently, Jin et al. [13] studied top-*k* queries in the uncertain data stream setting.

As discussed in Section 1, we can classify the proposed semantics into two categories, both of which are useful for their own application scenarios. In this paper, we extend the work in the first category and propose new semantics which shifts the emphasis more toward ranking scores. As we have discussed, our new semantics is useful for many applications that are not sufficiently addressed before.

Zhang and Chomicki [22] proposed the Global-Top*k* semantics which falls into the second category. Interestingly, in the future work section of [22], two of the open problems that the authors listed are: (1) integrating the *strength of preference* expressed by score into the semantics framework (i.e., existing semantics are not as sensitive to *score* as to *probability*) and (2) considering non-injective scoring functions (ties). Our work happens to address both of these open problems.

7. CONCLUSIONS

In this work, we observe the need to shift the emphasis a little more on *ranking scores*, as opposed to the *probabilities* for many applications. We propose to provide the score distribution of top-*k* vectors and *c*-Typical-Top*k* answers to applications and devise efficient algorithms to cope with the computational challenges. We also extend the work to score ties. Experimental results verify our motivation and our approaches.

8. ACKNOWLEDGMENTS

We wish to thank the anonymous referees for several comments and suggestions that have improved the paper. This work was supported by the NSF, under the grants IIS-0086057, IIS-0325838, and IIS-0448124.

9. REFERENCES

[1] L. Antova, T. Jansen, C. Koch, D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. In *ICDE*, 2008.

[2] R. Bellman. *Dynamic Programming*. Princeton Univ. Press, 1957.

[3] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[4] T. Cover and J. Thomas. *Elements of Information Theory*. A Wiley-Interscience Publication, 1991.

[5] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*, 2004.

[6] N. Dalvi and D. Suciu. Management of Probabilistic Data: Foundations and Challenges. In *PODS*, 2007.

[7] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and Aggregating Rankings with Ties. In *PODS*, 2004.

[8] R. Hassin and A. Tamir. Improved complexity bounds for location problems on the real line. In *Operations Research Letters*, 1991.

[9] M. Hua, J. Pei, W. Zhang, X. Lin. Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach. In *SIGMOD*, 2008.

[10] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *SenSys*, 2006.

[11] I. Ilyas, G. Beskales, and M. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. In *ACM Computing Surveys*, Vol. 40, 2008.

[12] R. Jampani, F. Xu, M. Wu, L. Perez, C. Jermaine, and P. Haas. MCDB: A Monte Carlo Approach to Managing Uncertain Data. In *SIGMOD*, 2008.

[13] C. Jin, K. Yi, L. Chen, J. Yu, X. Lin. Sliding-Window Top-k Queries on Uncertain Streams. In *VLDB*, 2008.

[14] S. Lim, H. Balakrishnan, D. Gifford, S. Madden, D. Rus. Stochastic Motion Planning and Applications to Traffic. In *WAFR*, 2008.

[15] C. Re, N. Dalvi and D. Suciu. Efficient Top-k Query Evaluation on Probabilistic Data. In *ICDE*, 2007.

[16] K. Rosen. *Discrete Mathematics and Its Applications*. 1995.

[17] P. Sen and A. Deshpande. Representing and Querying Correlated Tuples in Probabilistic Databases. In *ICDE*, 2007.

[18] M. Soliman, I. Ilyas, and K. Chang. Top-k Query Processing in Uncertain Databases. In *ICDE*, 2007.

[19] N. Tatbul, M. Buller, R. Hoyt, S. Mullen, S. Zdonik. Confidence-based Data Management for Personal Area Sensor Networks. In *DMSN*, 2004.

[20] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.

[21] K. Yi, F. Li, G. Kollios, D. Srivastava. Efficient Processing of Top-k Queries in Uncertain Databases with x-Relations. In *TKDE*, 2008.

[22] X. Zhang and J. Chomicki. On the Semantics and Evaluation of Top-k Queries in Probabilistic Databases. In *DBRank*'08.

[23] The R Project for Statistical Computing: www.r-project.org.

[24] http://en.wikipedia.org/wiki/Edit_distance.