Top-k Queries over Uncertain Scores

Qing Liu[1], Debabrota Basu[2], Talel Abdessalem[3], Stéphane Bressan[4]

[124]School of Computing, National University of Singapore, Singapore
[1]liuqing,[2]debabrota.basu@u.nus.edu, [4]steph@nus.edu.sg
[3]LTCI/IPAL CNRS, Télécom ParisTech, Université Paris-Saclay, France
talel.abdessalem@telecom-paristech.fr

Abstract Modern recommendation systems leverage some forms of collaborative user or crowd sourced collection of information. For instance, services like TripAdvisor, Airbnb and HungyGoWhere rely on user-generated content to describe and classify hotels, vacation rentals and restaurants. By nature of such independent collection of information, the multiplicity, diversity and varying quality of the information collected result in uncertainty. Objects, such as the services offered by hotels, vacation rentals and restaurants, have uncertain scores for their various features. In this context, ranking of uncertain data becomes a crucial issue. Several data models for uncertain data and several semantics for probabilistic top-k queries have been proposed in the literature. We consider here a model of objects with uncertain scores given as probability distributions and the semantics proposed by the state of the art reference work of Soliman, Hyas and Ben-David.
In this paper, we explore the design space of Metropolis-Hastings Markov chain Monte Carlo algorithms for answering probabilistic top-k queries over a database of objects with uncertain scores. We are able to devise several algorithms that yield better performance than the reference algorithm. We empirically and comparatively prove the effectiveness and efficiency of these new algorithms.

1 Introduction

Modern recommendation systems, in the most general sense of the term, rely on some forms of collaborative user or crowd sourced collection of information. For instance, cooperative or crowd sourced information system like TripAdvisor relies on user generated ratings and reviews to recommend travel plans and hotels. Airbnb and HungyGoWhere rely on user-generated content to describe, rank and recommend vacation rentals and restaurants, respectively.

By nature of such independent collection of information, the multiplicity, diversity and varying quality of the information collected result in uncertainty. Objects, such as the services offered by hotels, vacation rentals and restaurants, have uncertain scores quantifying features such as various quality and budget dimensions.

Ranking is one of the building blocks of recommendation. Given a database of objects ranked by their scores for the feature of interest, a top-k query returns

the sequence of the k objects with the highest score or rank, ordered by their score or rank. This sequence is referred to as the *top-k*. With uncertain scores and uncertain ranks, a top-k query can only return an uncertain result.

Several methods [14,11,16,5,4,12,13,8,7,15] have been proposed in the literature that answer probabilistic top-k queries based on different uncertain data model and semantics. Among these works, Soliman, Hyas and Ben-David [12] study top-k queries over a database of objects with uncertain scores given as probability distributions. Here, a probabilistic top-k query returns the sequence of objects that has the highest probability to be the top-k according to the probability distributions of the scores. In this paper, we consider probabilistic top-k queries under this semantics.

The authors of [12] propose several methods to answer such probabilistic top-k queries. The problem is combinatorial. The proposed methods are either inefficient with exponential time complexity or ineffective with poor approximation of the probability of the probabilistic top-k sequence. The most practical results are obtained with an approximate algorithm that searches the space of candidate top-k sequences of objects using a Markov chain Monte Carlo method and that computes the probability of a given sequence to be the top-k using Monte Carlo integration. We think that the approach is the right one but suspect that it can be further improved. Therefore, in this paper, we explore the design space for Metropolis-Hastings Markov chain Monte Carlo algorithms to answer probabilistic top-k queries in a database of objects with uncertain scores. We devise and present several new algorithms.

We first analytically discuss the ideas of the new algorithms. Then, we empirically evaluate the effectiveness and efficiency of the new algorithms in comparison with the reference. The experimental results confirm the superiority of the new algorithms over the reference algorithm. Surprisingly, it is the simplest and less involved algorithm that yields the best performance.

2 Related Work

Uncertainty and the Crowd. Content is increasingly generated by end users and information gathering outsourced. Consequently, the content of modern databases may be erroneous, noisy and, generally, uncertainty [1].

Yet the very problem of resolving the uncertainty can itself be outsourced to the crowd. For instance, the authors of [3] propose to use the crowd to answer top-k and group-by queries. They proposed a variable error model for controlling erroneous answers from the crowd. The authors of [17] provide a detailed survey of crowdsourced approaches the top-k problem and discuss how comparison-based algorithms, rating-based algorithms and hybrid algorithms can tolerate the errors from the crowd. They provide empirical guidelines for selecting appropriate algorithms for various scenarios.

Ranking and Top-k Queries over Uncertain Databases. Probabilistic top-k queries are first proposed in [14]. The uncertain data model consists of

membership probabilities for the objects in the database and possible worlds defined by Boolean constraints.

The authors [14] propose two types of top-k query semantics referred to as U-Top$k$ and U-kRanks queries. Each object (tuple) belongs to the database with a given probability while possible worlds are defined by logical formulae. A U-Top$k$ query returns the sequence of objects with the highest probability to be top-k across all possible worlds or most probable top-k. A U-kRanks query returns a sequence of objects that are individually most probable at their rank over all possible worlds. It must be noted that it is possible this particular sequence of objects may not be the most probable top-k.

The authors of [16] also study these two top-k semantics. They present several algorithms for both types of queries in terms of computation and memory consumption.

The authors of [11] formulate the query answering problem to the problem of evaluating probabilities for constraints in disjunctive normal form (thus avoiding the combinatorial computation of the possible worlds). They propose a Monte Carlo multi-simulation algorithm that repeatedly chooses at random a possible world and rank the top-k answers of an SQL query. The ranks of the top-k answers are guaranteed to be correct and the probability of the answer are approximate. The paper focuses on conjunctive queries and does not handle continuous attribute values.

Several other semantics and algorithms have been proposed for possible worlds data models. For example, the authors of [5] propose top-k query semantics using a threshold. They find the set of objects that have probability at least p to be in the top-k in all possible worlds. The authors propose both an exact algorithm based on pruning and an approximate algorithm based on sampling.

In [4], the authors stress the tradeoffs between reporting high-scoring objects and objects with a high probability of being in the top-k answer. They argue the need to present the score distribution of top-k sequences to allow the user to choose among the results. They thus proposed a new semantics for the top-k query, which they called c-Typical-Top$k$, that provides a number of typical top-k vectors for the user to choose from.

The semantics of top-k query can not only be defined under a possible worlds semantics, but also under parameterized ranking functions. The authors of [12,13] study the problem of ranking objects with uncertain scores. The uncertainty of the result of queries stems from stochastic attribute values. Each attribute of the objects to rank has a score which is a function over a domain inducing a probability distribution. The authors propose a baseline algorithm and a branch-and-bound algorithm to compute the result of top-k queries (U-Top$k$ queries). The complexity of these exact solutions is exponential. The authors propose a sampling-based Markov chain Monte Carlo approach to compute approximate answers.

The authors of [8], [7] and [15] also study the problem of ranking objects with uncertain scores but use different top-k semantics. The authors of [2,6] discuss and compare several existing top-k semantics.

In this paper, we consider a database of objects with uncertain scores given as probability distributions. We consider the semantics of U-Top$k$ queries proposed in [12]: a probabilistic top-k query returns the sequence of objects that has the highest probability to be the top-k according to the probability distributions of the scores. Motivated by the good performance results of [13], we decide to explore the design space for Markov chain Monte Carlo algorithms to answer probabilistic top-k queries in a database of objects with uncertain scores.

3 Problem Statement

3.1 Top-k Sequence

Let us assume a set $\mathcal{O}$ of n objects and a *scoring function* s from the set of objects, $\mathcal{O}$, to a totally ordered domain, $\mathcal{D}$. The function $s : \mathcal{O} \mapsto \mathcal{D}$, where $(\mathcal{D}, \geq)$ is a total order and $(\mathcal{D}, >)$ is the corresponding strict total order. The image, $s(o)$, of an object $o \in \mathcal{O}$ by the function s is called the score of the object.

Along with the scoring function s, a total order $(\mathcal{D}, \geq)$ induces a ranking on $\mathcal{O}$. For any two objects o_i and $o_j \in \mathcal{O}$, if score of o_i is greater than or equal to that of o_j, i.e., $s(o_i) \geq s(o_j)$, we say that o_i has equal rank to or higher rank than o_j. For the sake of simplicity, we abuse the notation $o_i \geq o_j$ to represent the fact that o_i has equal rank to or higher rank than o_j.

The *top-k sequence* of $\mathcal{O}$ for s with $(\mathcal{D}, \geq)$ is the sequence k objects of $\mathcal{O}$ with the highest ranks in order of their rank.

Definition 1. *The **top-k sequence** of objects in $\mathcal{O}$ with s and $(\mathcal{D}, \geq)$ is the sequence of k objects $\pi^{(k)} = [o_1, \cdots, o_k]$ such that $\forall o \in \bar{\mathcal{O}}(\pi^{(k)})$ $(o_1 \geq \ldots \geq o_k \geq o)$, where $\bar{\mathcal{O}}(\pi^{(k)}) = \mathcal{O} \setminus \pi^{(k)}$.*

3.2 Probability of a Sequence over Uncertain Scores

However, in many applications, typically when the score is collected from a multitude of independent sources, objects do not come with a deterministic score. In order to model this scenario, we consider that the score $s(o_i)$ of an object $o_i \in \mathcal{O}$ is a random variable X_i with a probability density function f_i. Namely, $s : \mathcal{O} \mapsto \{f : \mathcal{D} \mapsto [0, 1]\}$.

Each sequence of k objects in $\mathcal{O}$, $\pi^{(k)} = [o_1, \cdots, o_k]$, has a probability to be the top-k sequence with the realizations of the random variables in the image of s and $(\mathcal{D}, \geq)$.

Definition 2. *The **probability of a sequence** of objects in $\mathcal{O}$, $\pi^{(k)} = [o_1, \cdots, o_k]$, to be the top-$k$ with s and $(\mathcal{D}, \geq)$ is the joint probability function $\mathbf{P}(\forall o \in \bar{\mathcal{O}}(\pi^{(k)})$ $(o_1 \geq \ldots \geq o_k \geq o))$.*

For the sake of simplicity, we refer this joint probability function $\mathbf{P}(\forall o \in \bar{\mathcal{O}}(\pi^{(k)})\ (o_1 \geq \ldots \geq o_k \geq o))$ as $Pr(\pi^{(k)})$ for a sequence $\pi^{(k)} = [o_i, \cdots, o_k]$ for the remainder of this paper.

In most applications the features of objects are independent. We therefore consider that the random variables $\{X_i\}_{o_i \in \mathcal{O}}$, i.e., the individual scores of the objects, are independent.

The probability, $Pr(\pi^{(k)})$, of the sequence $\pi^{(k)}$ to be the top-k is given by the n-dimensional integral with dependent limits of Equation 1 as given in [12,13]. The upper limits of the variables $\{x_1, \cdots, x_k\}$ are $\{\infty, x_1, \cdots, x_{k-1}\}$ and the upper limits of the variables $\{x_{k+1}, \cdots, x_n\}$ are x_k.

$$Pr(\pi^{(k)}) = \int_{-\infty}^{\infty} \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_{k-1}} \int_{-\infty}^{x_k} \cdots \int_{-\infty}^{x_k} f_1(x_1) \cdots f_n(x_n)\ dx_n \cdots dx_1 \quad (1)$$

3.3 Calculating the Probability of a Sequence over Uncertain Scores

In practical cases, it is meaningful to consider that the random variables X_i have bounded continuous density functions. They have a closed interval as support. That is the scores of objects have a minimum and a maximum value. We use $[l_i, u_i]$ to denote the lower and upper bounds of the support of the continuous functions f_i, respectively.

For bounded continuous density functions $f_1, \cdots, f_n$, equation 1 can be estimated by Monte Carlo Integration [10] if $f_1(x_1), \cdots, f_n(x_n)$ can be calculated given $x_1, \cdots, x_n$[12,13].

Monte Carlo Integration is useful for estimating high dimensional integrals, which are often expensive to evaluate, by adopting sampling techniques which take smaller computational effort [9]. Suppose we want to compute $\int_{\Omega'} f(x)dx$, where Ω' is a bounded subspace with volume $\int_{\Omega'} dx$. It is circumscribed by another subspace Ω, i.e., $\Omega' \subseteq \Omega$, whose volume v can be easily computed. Now, we can sample S points uniformly in Ω. Suppose m of those S samples are in Ω', i.e., sample $x^1, \cdots, x^m$ are in Ω'. Then the integral $\int_{\Omega'} f(x)dx$ can be estimated by Equation 2.

$$\int_{\Omega'} f(x)dx \approx \frac{m}{S} \cdot v \cdot \frac{1}{m} \sum_{i=1}^{m} f(x^i) \quad (2)$$

The approximation error of Equation 2 is $O(\frac{1}{\sqrt{S}})$ [9].

For all practical purposes, one can sample the score x_i for each object o_i uniformly from the corresponding support $[l_i, u_i]$. One sample x^i consists of n scores, i.e., $x^i = \{x_1^i, \cdots, x_n^i\}$. After sampling S times, we get samples $x^1, \cdots, x^S$. Suppose that the scores in sample $x^1, \cdots x^m$ satisfy $\forall o \in \bar{\mathcal{O}}(\pi^{(k)})(o_1 \geq \cdots \geq o_k \geq o)$, while other samples do not satisfy this condition. We can estimate $Pr(\pi^{(k)})$ by Monte Carlo Integration according to Equation 3 where $v = \prod_{i=1}^{n}(u_i - l_i)$ is the volume of the space sampled.

$$Pr(\pi^{(k)}) \approx \frac{m}{S} \cdot v \cdot \frac{1}{m} \sum_{i=1}^{m} \prod_{j=1}^{n} f_j(x_j^i) \quad (3)$$

Thus, for general bounded continuous density functions $f_1, \cdots, f_n$, given a sequence of k objects $\pi^{(k)}$, we are able to estimate the probability $Pr(\pi^{(k)})$ via Monte Carlo integration.

If the cumulative distribution function (cdf) of each f_i can easily be computed, i.e., $F_i(x) = \int_{l_i}^{x} f_i(y)dy$ can easily be computed given x, then we can compute $Pr(\pi^{(k)})$ as the k-dimensional integral with dependent limits of Equation 4.

$$Pr(\pi^{(k)}) = \int_{-\infty}^{\infty} \int_{-\infty}^{x_1} ... \int_{-\infty}^{x_{k-1}} f_1(x_1)...f_k(x_k) \cdot (\prod_{o_i \in \bar{O}(\pi^{(k)})} F_i(x_k))dx_k...dx_1 \quad (4)$$

We only need to sample the scores for k objects with equation 4 to estimate $Pr(\pi^{(k)})$ via Monte Carlo Integration. One sample consists of k scores $x_1, \cdots, x_k$, i.e., $x^1 = \{x_1^1, \cdots, x_k^1\}$. After sampling S times, we get samples $x^1, \cdots, x^S$. Suppose scores in samples $x^1, \cdots x^m$ satisfy $o_1 \geq \cdots \geq o_k$, while other samples do not satisfy this condition. Equation 4 can be estimated using Equation 5.

$$Pr(\pi^{(k)}) \approx \frac{m}{S} \cdot v \cdot \frac{1}{m} \sum_{i=1}^{m} (\prod_{j=1}^{k} f_j(x_j^i) \prod_{o_j \in \bar{\mathcal{O}}(\pi^{(k)})} F_j(x_k^i)) \quad (5)$$

3.4 Probabilistic Top-k Sequence

In this paper, we study the top-k query that returns the probabilistic top-k sequence (Definition 3) of objects in $\mathcal{O}$ given the probability density functions f_i of the random variables $s(o_i) = X_i$ for each of the objects $o_i \in \mathcal{O}$. In order to find the probabilistic top-k sequence, we calculate and maximize the probability of sequences of k objects in $\mathcal{O}$, $\pi^{(k)}$, to be a top-k sequence.

Definition 3. *The **probabilistic top-k sequence** of objects in $\mathcal{O}$ with s and $(\mathcal{D}, \geq)$ is the sequence of k objects $\pi^{(k)} = [o_1, \cdots, o_k]$ that maximizes the joint probability function* $\mathbf{P}(\forall o \in \bar{\mathcal{O}}(\pi^{(k)}) \ (o_1 \geq \ldots \geq o_k \geq o))$.

For answering the top-k query and calculating the probabilistic top-k sequence, a naive optimal approach consists in calculating $Pr(\pi^{(k)})$ for every possible sequence $\pi^{(k)}$ and returning the $\pi^{(k)}$ with the highest $Pr(\pi^{(k)})$. This approach is combinatorial in n and k, since there are $\frac{n!}{(n-k)!}$ possible sequences to examine. In [13], the authors proposed a Branch-and-Bound alternative to find the sequence $\pi^{(k)}$ with the highest $Pr(\pi^{(k)})$ optimally. The idea of this approach is based on the following property. Given a sequence $\pi^{(k)}$ with length k, any sequence $\pi^{(x)}$ with length $x \leq k$ and $Pr(\pi^{(x)}) < Pr(\pi^{(k)})$ can be safely pruned from the candidate results since $Pr(\pi^{(x)})$ upper-bounds the probability of any top-k sequence extended from $\pi^{(x)}$. The complexity of this approach is still combinatorial in n an k.

Soliman's Algorithm In order to calculate the probabilistic top-k sequence efficiently, Soliman, Hyas and Ben-David [12,13] propose an approximate algorithm that searches the space of candidate probabilistic top-k sequences using a Markov chain Monte Carlo algorithm. The idea is to sample the state, i.e., a ranking over n objects, from a target distribution and hope that the states visit the probabilistic top-k sequence we seek.

Figure 1 is an example of the Markov chain in Soliman's algorithm. A state in the Markov Chain is a ranking over all the n objects, e.g., the ranking at step t is π_t. To generate the next state, this approach shuffles the rank of the objects by swapping two objects randomly according to the pairwise probabilities. More specifically, to generate a candidate state π_{t+1}, they randomly pick a rank r in current state, move the object $o(r)$ at rank r downward if $r \in [1, k]$, otherwise move the object upward. The movement is done by swapping the object $o(r)$ with $o(r+1)$ with probability $Pr(o(r+1) > o(r))$ if $r \in [1, k]$ or swapping $o(r)$ with $o(r-1)$ with probability $Pr(o(r) > o(r-1))$ if $r \notin [1, k]$. Swapping is conducted one by one and stops at the first uncommitted swap. Do the above swapping for multiple number of randomly picked ranks, i.e., multiple r-s. Then the proposal distribution $Pr(\pi_{t+1}|\pi_t)$ is the product of the probabilities of all the committed swaps since each swap in committed independently. Finally, the candidate state π_{t+1} is accepted with probability $\alpha = \min(\frac{Pr(\pi_{t+1}^{(k)}) \cdot Pr(\pi_t|\pi_{t+1})}{Pr(\pi_t^{(k)}) \cdot Pr(\pi_{t+1}|\pi_t)}, 1)$.

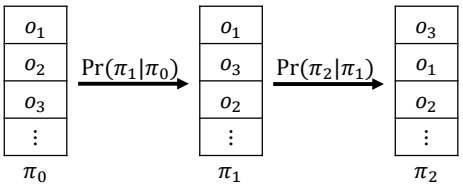


Figure 1. An Example of Markov chain

In this paper, we explore the design space and devise several Metropolis-Hastings Markov chain Monte Carlo algorithms which answer the top-k query efficiently. We will elaborate our algorithms in next section.

4 Markov chain Monte Carlo Algorithms

In this section, we propose several variants of a Markov chain Monte Carlo algorithm based on different ways of generating the candidate state: by swapping objects in the sequence or by sampling or re-sampling scores.

First of all, we present the general framework of the Markov chain Monte Carlo approaches in Algorithm 1. We use variable "BestSeq" to remember the most probable top-k sequence we have seen so far and "Pr(BestSeq)" as its probability (Line 1-2). The initial state of a Markov Chain is generated by sampling

the score for each object from the corresponding score distribution. Sorting the scores gives us a ranking over n objects. This ranking is used as the initial state (Line 4). A bad initial state may trap the random walk in a region for many steps. The authors of [12,13] proposed to run multiple independent Markov Chains with independent initial states (Line 3). The candidate states in the Markov Chain are generated in different ways as we will elaborate in this section (Line 7). Each candidate state is accepted with the acceptance probability α which guarantees that the chain converges to the target distribution with sufficient number of steps (Line 8). Update the most probable top-k sequence seen so far (Line 10-11). Finally, return the most probable sequence of length k as the probabilistic top-k sequence (Line 12).

Algorithm 1: Framework of Markov chain Monte Carlo Algorithms

Input: Score distributions $f_1, \cdots, f_n$
Output: The probabilistic top-k sequence $\pi^{(k)}$

```
1  BestSeq ← null;
2  Pr(BestSeq) ← 0;
3  for Chain ID = 1 to C do
4      Generate the initial state π_0;
5      for Step t = 1 to L do
6          Set state π_t = π_{t-1};
7          Generate a candidate state π'_t(with Different Algorithms);
8          Set π_t = π'_t with probability α;
9          if Pr(π_t^(k)) > Pr(BestSeq) then
10             BestSeq ← π_t^(k);
11             Pr(BestSeq) ← Pr(π_t^(k)) ;
12 Return BestSeq;
```

The First Variant (Swap) In this algorithm, a state is a ranking over n objects. To generate a candidate state π_{t+1} from current state π_t, we pick two ranks r_1 and r_2 randomly such that $r_1 \in [1, k]$ and $r_2 \in [1, n]$, swap the object $o(r_1)$ at rank r_1 with the object $o(r_2)$ at rank r_2. This simple process generates a candidate state π_{t+1}. The proposal distribution $Pr(\pi_{t+1}|\pi_t) = \frac{1}{k} \cdot \frac{1}{n}$. The candidate state π_{t+1} is accepted with probability $\alpha = \min(\frac{Pr(\pi_{t+1}^{(k)}) \cdot \frac{1}{kn}}{Pr(\pi_t^{(k)}) \cdot \frac{1}{kn}} = \frac{Pr(\pi_{t+1}^{(k)})}{Pr(\pi_t^{(k)})}, 1)$. This Markov Chain generates samples from the target distribution $Pr(\pi^{(k)})$.

The Second Variant (SwapEXP) Actually, we are more interested in the sequences of length k that have higher probabilities of being the probabilistic top-k sequence. We refer "good states" to the states that contain the top-k sequence of objects, $\pi^{(k)}$, of high probability $Pr(\pi^{(k)})$. We want to sample from

the target distribution where good states have higher probabilities. Thus, we design the following weighted target distribution for the Markov chain:

$$\widehat{Pr}(\pi^{(k)}) = C_\beta^{-1} \exp(\beta Pr(\pi^{(k)})) \tag{6}$$

where C_β^{-1} is a normalizing constant. β is a parameter, where a larger β means the target distribution $\widehat{Pr}(\pi^{(k)})$ puts heavier weight on the good states.

With this weighted target distribution, we can generate the candidate state by swapping two objects as described in `Swap` algorithm and accept the candidate state with probability $\alpha = \min(\frac{\widehat{Pr}(\pi_{t+1}^{(k)})}{\widehat{Pr}(\pi_t^{(k)})} = \exp(\beta(Pr(\pi_{t+1}^{(k)}) - Pr(\pi_t^{(k)}))), 1)$. From this formula $a' = \exp(\beta(Pr(\pi_{t+1}^{(k)}) - Pr(\pi_t^{(k)})))$, we can see that when $Pr(\pi_{t+1}^{(k)}) > Pr(\pi_t^{(k)})$, a' is very likely to exceed 1 with a large β. Thus, the candidate state π_{t+1} is more likely to be accepted. On the contrary, when $Pr(\pi_{t+1}^{(k)}) < Pr(\pi_t^{(k)})$, a' is approaching 0. Thus, the candidate state π_{t+1} is more likely to be rejected.

The Third Variant (ReSample) In this algorithm, a state is a ranking over n objects. Once we have the initial states, a re-sampling technique can be adopted to generate a candidate state. Specifically, to generate a candidate state, we randomly pick a rank r from $[1, n]$ and re-sample the score of object $o(r)$ at rank r from its score distribution $f_{o(r)}$. Then redo the sorting over the n objects according to the new scores. This process generates a new ranking over n objects, which is regarded as a candidate state π_{t+1}. Suppose the rank of $o(r)$ in the candidate state is r'. If $r' > r$, it means the object $o(r)$ jumps to a lower rank, then the proposal distribution $Pr(\pi_{t+1}|\pi_t) = \frac{1}{n}\frac{\min(u_{o(r)}, s(o(r'))) - \max(l_{o(r)}, s(o(r'+1)))}{u_{o(r)} - l_{o(r)}}$, where $o(r')$ is the object at rank r', $s(o(r'))$ is its score at time step t. Else if $r' < r$, the object $o(r)$ jumps to a higher rank, the proposal distribution $Pr(\pi_{t+1}|\pi_t) = \frac{1}{n}\frac{\min(u_{o(r)}, s(o(r'-1))) - \max(l_{o(r)}, s(o(r')))}{u_{o(r)} - l_{o(r)}}$. Else (i.e., $r = r'$), $Pr(\pi_{t+1}|\pi_t) = \frac{1}{n}\frac{\min(u_{o(r)}, s(o(r'-1))) - \max(l_{o(r)}, s(o(r'+1)))}{u_{o(r)} - l_{o(r)}}$. The proposal distribution $Pr(\pi_{t+1}|\pi_t)$ reflects the probability that the object $o(r)$ is picked at step t and it jumps to the new rank r' via re-sampling. $Pr(\pi_t|\pi_{t+1}) = \frac{1}{n}\frac{\min(u_{o(r)}, s(o(r-1))) - \max(l_{o(r)}, s(o(r+1)))}{u_{o(r)} - l_{o(r)}}$. The candidate state π_{t+1} is accepted with probability $\alpha = \min(\frac{Pr(\pi_{t+1}^{(k)}) \cdot Pr(\pi_t|\pi_{t+1})}{Pr(\pi_t^{(k)}) \cdot Pr(\pi_{t+1}|\pi_t)}, 1)$.

The Forth Variant (ReSampleEXP) In this algorithm, we generate the candidate state as described in `ReSample` algorithm and adopt the weighted target distribution as in Equation 6. Thus, the candidate state π_{t+1} is accepted with probability $\alpha = \min(\frac{Pr(\pi_t|\pi_{t+1})}{Pr(\pi_{t+1}|\pi_t)} \cdot \exp(\beta(Pr(\pi_{t+1}^{(k)}) - Pr(\pi_t^{(k)}))) , 1)$.

The Fifth Variant (ReSampleAll) The initial state is generated in the same way as in previous Markov chain Monte Carlo Algorithms. To generate a candi-

date state π_{t+1}, this algorithm re-samples the score for each of the n objects and sort them according to the newly sampled scores. This process generates a new ranking over n objects, which is regarded as the candidate state. The proposal distribution $Pr(\pi_{t+1}|\pi_t) = Pr(\pi_{t+1}^{(k)})$. The candidate state π_{t+1} is accepted with probability $\alpha = \min(\frac{Pr(\pi_{t+1}^{(k)}) \cdot Pr(\pi_t|\pi_{t+1})}{Pr(\pi_t^{(k)}) \cdot Pr(\pi_{t+1}|\pi_t)} = 1, 1) = 1$. In this algorithm, the candidate state is generated from the space of the ranking and is independent from current state.

For simplicity, in the rest of this paper, we refer `Soliman` to the Soliman's algorithm which is the baseline algorithm, `Swap`, `SwapEXP`, `ReSample`, `ReSampleEXP` and `ReSampleAll` to our five variants of Markov chain Monte Carlo algorithms.

5 Performance Evaluation

In this section, we evaluate the performance of our proposed algorithms in comparison with the baseline Soliman's algorithm. We explain the settings of our experiments in Section 5.1. Then, in Section 5.2 we evaluate the effectiveness in terms of the probability of the probabilistic top-k sequence found by different algorithms, the higher probability means the better performance of the algorithm. In Section 5.3, we discuss the efficiency of different algorithms analytically and empirically.

5.1 Experimental Set-up

Synthetic Datasets We evaluate the Markov chain Monte Carlo algorithms based on a series of synthetic datasets. We prefer to use synthetic data in this paper, since it lets us study a wide range of patterns of the score intervals, i.e., each synthetic dataset represents one pattern of how the score intervals overlap. Specifically, in each dataset, the score interval $[l_i, u_i]$ of each object o_i is generated by drawing a median score, i.e., $\frac{l_i+u_i}{2}$, from a predefined *median score* distribution. The width of the interval is drawn from a predefined *width* distribution. The predefined median score distribution and the width distribution are summarized in Table 1, where $G(\mu, \sigma)$ represents Gaussian distribution with mean μ and standard deviation σ defined on [0,1] and $U[l, r]$ represents uniform distribution on the support $[l, r]$. Uniform distributions are used as score density functions in the experiments.

Table 1. Distributions

	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5
median score	$G(0.5, 0.05)$	$G(0.5, 0.2)$	$G(0.5, 0.8)$	$G(0.5, 3.2)$	$U[0, 1]$
width	$G(0.5, 0.05)$	$G(0.5, 0.2)$	$G(0.5, 0.8)$	$G(0.5, 3.2)$	$U[0, 1]$

We generate $5 \times 5 = 25$ datasets in total. In each dataset, the median scores and the widths of the score intervals are drawn from one of the settings in

Table 1, i.e., *Median score* with setting i and *width* with setting j generate the $(i-1)*5+j_{th}$ dataset. For example, the dataset where the median scores are drawn from $G(0.5, 0.05)$ and the widths are drawn from $G(0.5, 0.05)$ represents the set of objects that have a high degree of overlaps. In this case, the objects have very similar score intervals. For the dataset that the median scores are drawn from $U[0, 1]$, the widths are drawn from $G(0.5, 0.05)$, it represents the pattern that the score intervals are scattered and there are fewer overlaps of the score intervals. Thus, these datasets present different patterns of overlapping of the score intervals and reflect different levels of uncertainty of finding the probabilistic top-k sequence.

Parameters We run 10 Markov chains for each algorithm, the length of each chain grows from 1 to 100000. In the experiments, we watch how different algorithms behave as more states are generated (i.e., as the length of the chain increases) in terms of the probability of the probabilistic top-k sequence. Gelman-Rubin statistic is used to diagnose the convergence of the chains. For each of the algorithm, we use the same set of initial states such that the comparison is fair. The sample size for Monte Carlo Integration is 1000000. In the target distribution $\widehat{Pr}(\pi^{(k)})$, we set $\beta = 10^{10}$.

We set $k = 5$, $n = 1000$ unless otherwise is specified. That is, we are seeking for 5 objects from 1000 objects. The space of the sequences of length k is large, i.e., there are around 9.9E14 (990 trillion), sequences of length k in total. Thus, it is inefficient to traverse all the sequences of length k to find the probabilistic top-k sequence.

5.2 Effectiveness

Effect of Weighted Target Distribution Each data point in each sub-figure of Figure 2 and Figure 3 is the probability of the probabilistic top-k sequence found by the corresponding algorithm under certain number of steps in the Markov chain. These two figures show how a different target distribution for the Markov chain affects the performance.

Figure 2 compares the `Swap` algorithm and the `SwapEXP` algorithm in terms of the probability of the probabilistic top-k sequence they found. We can see that `Swap` and `SwapEXP` do not dominate each other, the performance of these two algorithms are similar over the 25 datasets in general. To be more specific, `SwapEXP` reaches a local optimal state earlier than `Swap`. However, there is a minor trend that `Swap` will outperform `SwapEXP` with sufficient number of steps.

Figure 3 compares the performance of the `ReSample` algorithm and the `ReSampleEXP` algorithm. In Figure 3, we can observe that the `ReSampleEXP` algorithm outperforms the `ReSample` algorithm in all the 25 datasets.

The weighted target distribution $\widehat{Pr}(\pi^{(k)})$ has different effects on the `Swap` and the `ReSample` algorithm. We explain this results as follows. With the weighted target distribution that has a large β, the `SwapEXP` and the `ReSampleEXP` algorithm are more likely to accept the candidate states that have a higher probability than that of current state (i.e., the "better" candidate states) and reject

the candidate states that have a lower probability than the current state (i.e., the "worse" candidate states), while `Swap` and `ReSample` will accept the "worse" candidate states with a certain probability. `SwapEXP` (`ReSampleEXP`) spends more steps searching the neighbor states since it only jumps to the "better" candidate states. Thus, it finds a local optimal state faster than `Swap` (`ReSample`).

The effect of the weighted target distribution is also related to the sampling strategy. Under the "swap" strategy, it is hard to jump to a better local optimal state. While "re-sample" is more effective in generating a better candidate state. `ReSample` is worse than `ReSampleEXP` since it probably rejects too many "better" candidate states.

Effectiveness Figure 4 presents probabilities of the probabilistic top-k sequence found by our proposed Markov chain Monte Carlo algorithms and by the Soliman's algorithm. The high the probability, the better the algorithm.

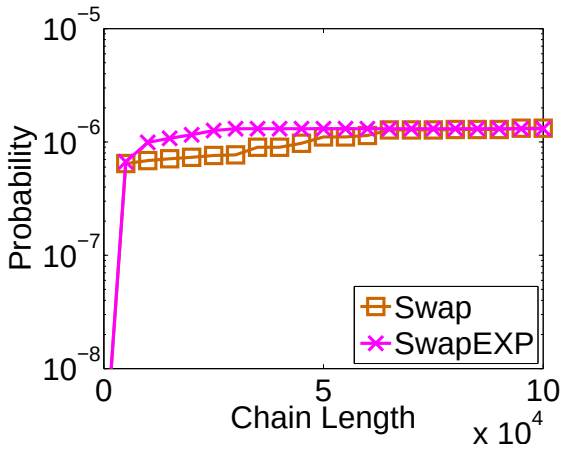
We can observe from figure 4 that `ReSampleAll` outperforms all the other algorithms in general. The superiority of the `ReSampleAll` algorithm is especially obvious in Dataset 1-5 where the score intervals have high degree of overlapping. The second best algorithm is the `ReSampleEXP` algorithm. The `Soliman` algorithm is the worst among the six algorithms.

Different ways of generating the candidate states results in the different performance of the six algorithm. The way of generating a candidate state determines whether an algorithm can reach an optimal state. The `ReSampleAll` algorithm generates the candidate state by sampling directly from the probability distribution, i.e., $Pr(\pi^{(k)})$, of sequences of length k. The probabilistic top-k sequence will be sampled with the highest probability, which is probably the reason why `ReSampleAll` performs well.

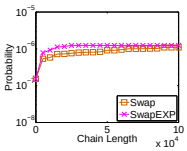
The idea of `ReSampleAll` and `ReSample` are similar except that `ReSample` generates states in a lighter-weight fashion of re-sampling the score for only one object at each step. `ReSampleAll` accepts all the candidate states while `ReSample` may reject a "better" candidate state, which is probably the reason why `ReSample` performs badly. `ReSampleEXP` fixes this problem of `ReSample` by accepting the "better" candidates with high probability. However, `ReSampleEXP` tends to stick to current local optimal state and not jumps to a better state efficiently.

The `Soliman` algorithm performs badly for the reason that it rarely generates a candidate state that has a different top-k sequence with the top-k sequence of current state. This is an intrinsic drawback of its process of generating candidate states where the top-k objects would be changed by the swap with very low probability. As a result, the top-k sequence in the candidate state is the same to the top-k sequence in current state in most steps. Thus, `Soliman` is not able to search a sufficient number of sequence in the top-k sequence space.

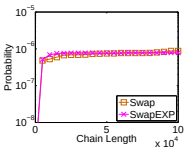
The `Swap` and `SwapEXP` algorithms try to fix `Soliman`'s drawback by involving one of the current top-k objects in a swap. However, `Swap` and `SwapEXP` are not smart enough because of their randomness in selecting the two objects for swapping.



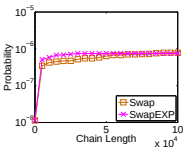
(a) Dataset1



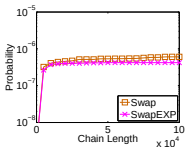
(b) Dataset2



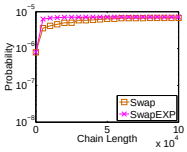
(c) Dataset3



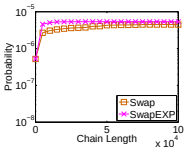
(d) Dataset4



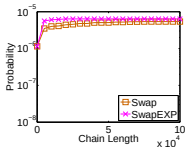
(e) Dataset5



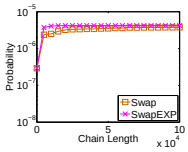
(f) Dataset6



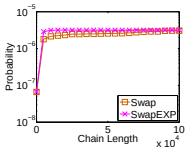
(g) Dataset7



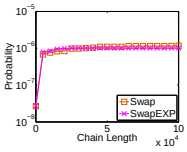
(h) Dataset8



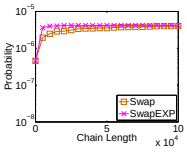
(i) Dataset9



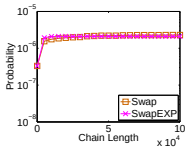
(j) Dataset10



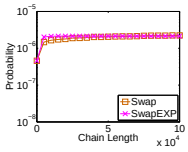
(k) Dataset11



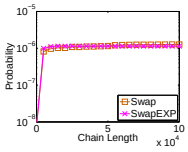
(l) Dataset12



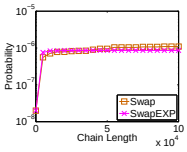
(m) Dataset13



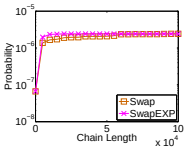
(n) Dataset14



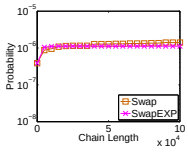
(o) Dataset15



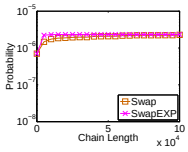
(p) Dataset16



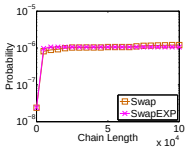
(q) Dataset17



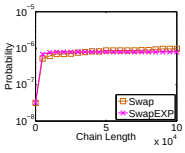
(r) Dataset18



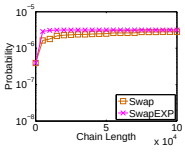
(s) Dataset19



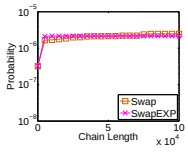
(t) Dataset20



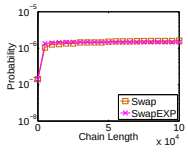
(u) Dataset21



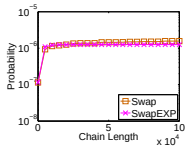
(v) Dataset22



(w) Dataset23

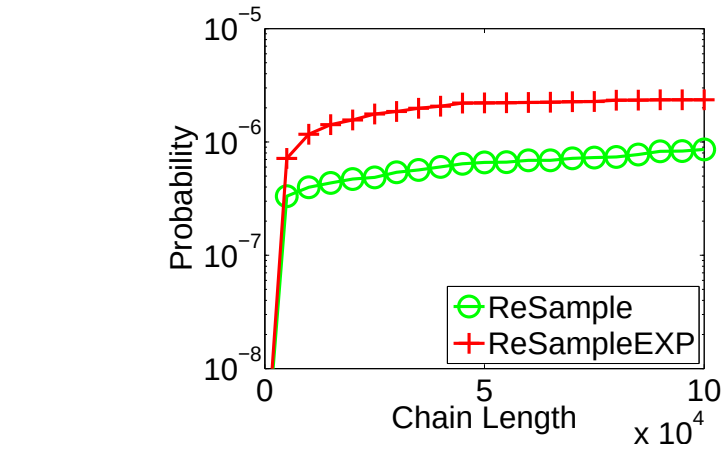


(x) Dataset24

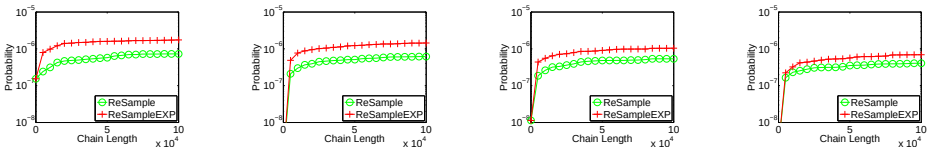


(y) Dataset25

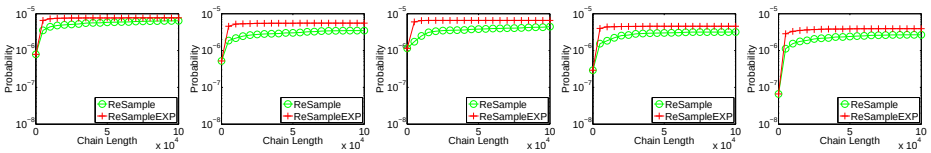
Figure 2. Probabilities of the Probabilistic Top-k Sequence Found by `Swap` and `SwapEXP`



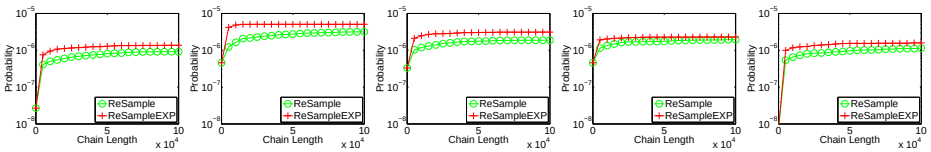
(a) Dataset1



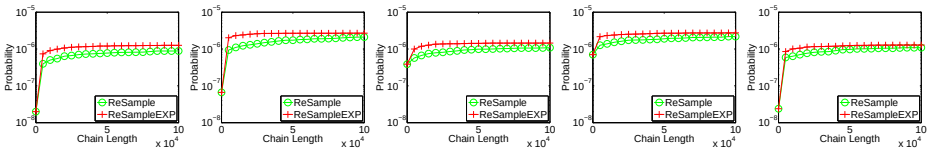
(b) Dataset2 (c) Dataset3 (d) Dataset4 (e) Dataset5



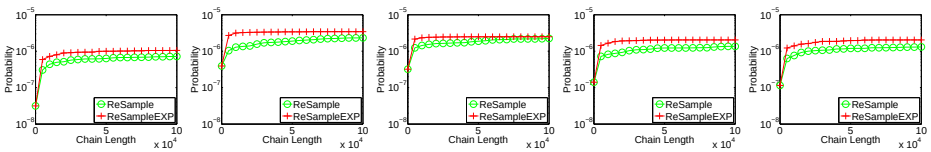
(f) Dataset6 (g) Dataset7 (h) Dataset8 (i) Dataset9 (j) Dataset10



(k) Dataset11 (l) Dataset12 (m) Dataset13 (n) Dataset14 (o) Dataset15

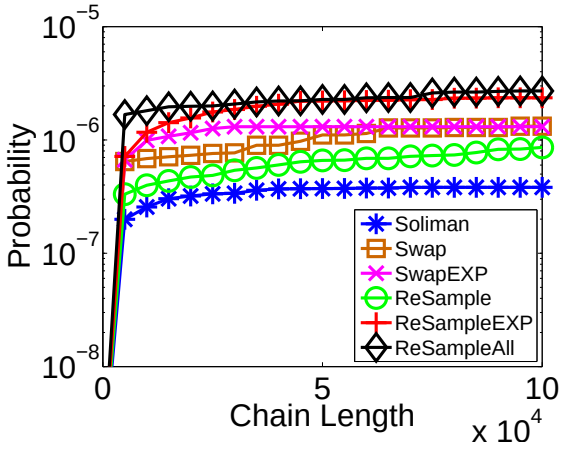


(p) Dataset16 (q) Dataset17 (r) Dataset18 (s) Dataset19 (t) Dataset20

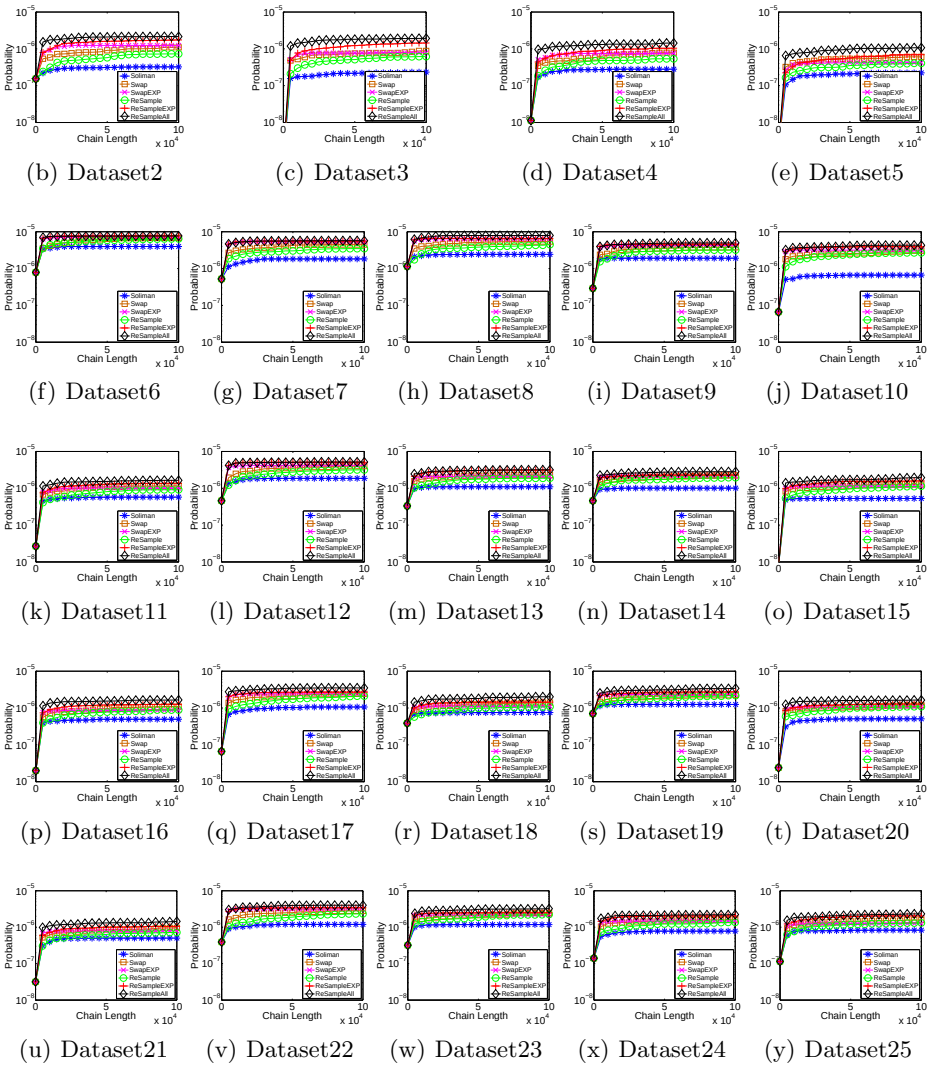


(u) Dataset21 (v) Dataset22 (w) Dataset23 (x) Dataset24 (y) Dataset25

Figure 3. Probabilities of the Probabilistic Top-k Sequence Found by `ReSample` and `ReSampleEXP`



(a) Dataset1



(b) Dataset2 (c) Dataset3 (d) Dataset4 (e) Dataset5

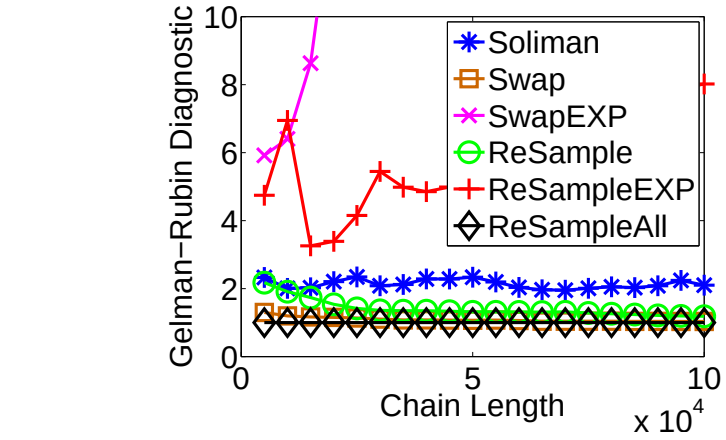
(f) Dataset6 (g) Dataset7 (h) Dataset8 (i) Dataset9 (j) Dataset10

(k) Dataset11 (l) Dataset12 (m) Dataset13 (n) Dataset14 (o) Dataset15

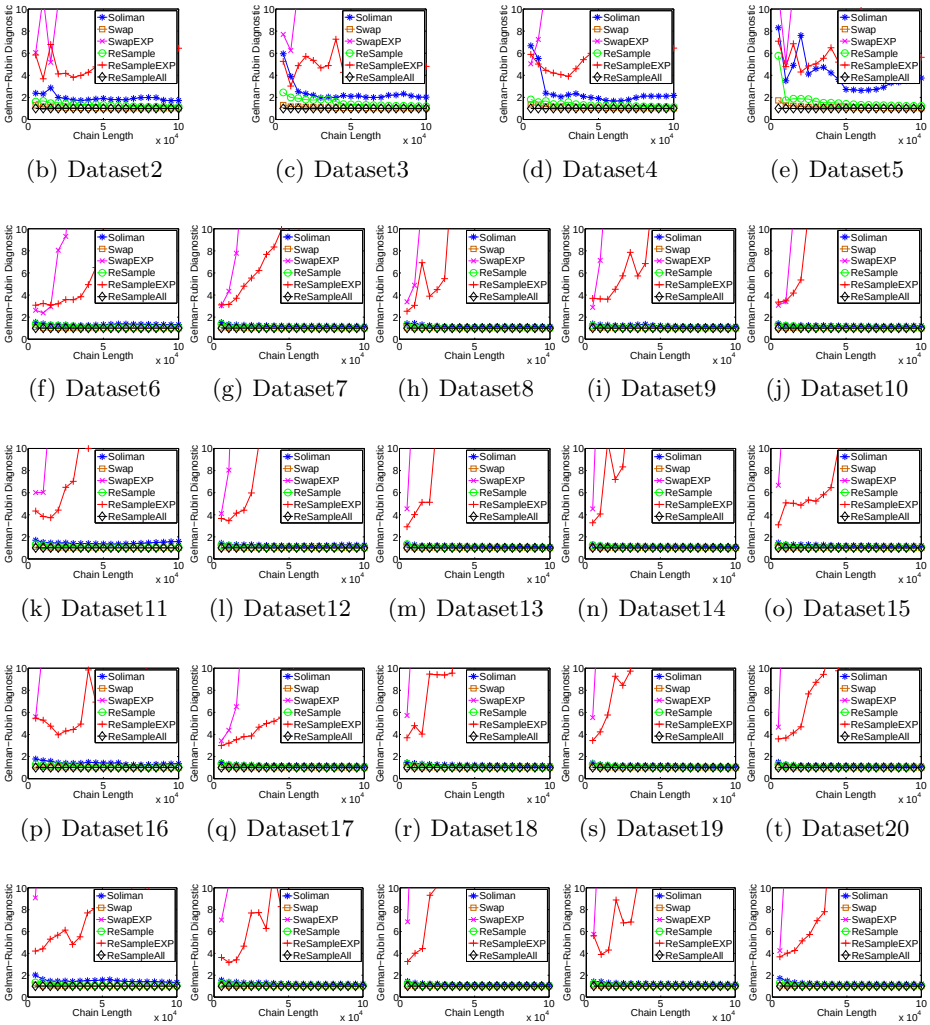
(p) Dataset16 (q) Dataset17 (r) Dataset18 (s) Dataset19 (t) Dataset20

(u) Dataset21 (v) Dataset22 (w) Dataset23 (x) Dataset24 (y) Dataset25

Figure 4. Probabilities of the Probabilistic Top-k Sequence Found by the Six Algorithms: `Soliman`, `Swap`, `SwapEXP`, `ReSample`, `ReSampleEXP`, `ReSampleAll`



(a) Dataset1



(b) Dataset2 (c) Dataset3 (d) Dataset4 (e) Dataset5

(f) Dataset6 (g) Dataset7 (h) Dataset8 (i) Dataset9 (j) Dataset10

(k) Dataset11 (l) Dataset12 (m) Dataset13 (n) Dataset14 (o) Dataset15

(p) Dataset16 (q) Dataset17 (r) Dataset18 (s) Dataset19 (t) Dataset20

(u) Dataset21 (v) Dataset22 (w) Dataset23 (x) Dataset24 (y) Dataset25

Figure 5. Convergence of the Markov chains Generated by the Six Algorithms: `Soliman`, `Swap`, `SwapEXP`, `ReSample`, `ReSampleEXP`, `ReSampleAll`

Generally, all of the six algorithms reach a relatively good state (compared to the initial states) within 5000 steps. After 5000 steps, the probability returned by each algorithm increases slowly.

5.3 Efficiency

Analytical Evaluation In Table 2, we refer the Time Complexity to the worst case time complexity of generating the candidate state in each algorithm.

In `Soliman`'s algorithm, each step consists of multiple swaps. There are nk swaps in maximum for generating the candidate state. So, time complexity of `Soliman` is $O(nk)$. Time complexity of `Swap` or `SwapEXP` is $O(1)$ since each step consists of only one swap. In `ReSample` algorithm, the new rank of the picked object can be determined within n comparisons in maximum. Thus, time complexity of `ReSample` is $O(n)$. In `ReSampleAll` algorithm, re-sampling the score for each object takes $O(n)$. Then, choosing the top-k highest scores takes $O(nlogk)$. So, time complexity of `ReSampleAll` is $O(nlogk)$.

Table 2. Worst Case Time Complexity of Generating Next State

	`Soliman`	`Swap(EXP)`	`ReSample(EXP)`	`ReSampleAll`
Time Complexity	$O(nk)$	$O(1)$	$O(n)$	$O(nlogk)$

Convergence Figure 5 shows the convergence of the six algorithms within 100000 steps. We use Gelman-Rubin Convergence Diagnostic to evaluate the convergence of the multiple Markov chains generated by different algorithms. The test compares the variance within and across chains. When chains mix, the test approaches 1. Values that are far higher than 1 mean poor convergence.

We can see that the algorithms, i.e. `SwapEXP` and `ReSampleEXP`, with the weighted target distributions $\widehat{Pr}(\pi^{(k)})$ do not converge well. Convergence means that the states in the chain are samples following the target distribution. As the target distributions $\widehat{Pr}(\pi^{(k)})$ put very high weights on the good states, only when all the states are the good states, will the chains converge. Thus, it is harder for a Markov chain with a weighted target distribution to converge.

Gelman-Rubin Convergence Diagnostic is a test of relative convergence. Therefore, as shown by the performance of `ReSampleEXP`, it is possible for an algorithm to be generally effective yet not to converge well.

Other algorithms converge well in general except that `Soliman` does not converge well in the Datasets 1-5. That means samples in the chains generated by the `Swap`, `ReSample` and `ReSampleAll` algorithms follow the target distribution $Pr(\pi^{(k)})$ after 50000 steps.

Runtime In practice, all the algorithms can generate the candidate states fast. The main consumption of the runtime lies in the computation of the probability

of the top-k sequence where the Monte Carlo integration is adopted. In our implementation, we cache in memory the probabilities of the top-k sequences seen so far. Thus, if we encounter an already seen sequence, there is no need to recompute its probability. We record the average runtime of one step for each algorithm, the results are shown in Table 3. In our experiments, `Soliman`'s algorithm runs fast. This is due to its inability to discover easily new top-k sequences, it tends to remain on the same top-k sequence when moving from a current state to a candidate state. In contrast, `ReSampleAll` tends to discover a new top-k sequence at most of the states, thus, it is slower than `Soliman`'s algorithm.

Table 3. Runtime Per Step of the Algorithms (seconds)

	`Soliman`	`Swap`	`SwapEXP`	`ReSample`	`ReSampleEXP`	`ReSampleAll`
Runtime Per Step	0.0058	1.9128	0.1163	0.0523	0.0071	0.9056

6 Conclusion

We study top-k queries over uncertain data. We consider probabilistic top-k queries that return the sequence of objects that has the highest probability to be the top-k according to a probability distributions of the scores. Finding such most probable top-k is combinatorial. Efficient algorithms are approximate. Markov chain Monte Carlo approaches are promising. We explore the design space for Metropolis-Hastings Markov chain Monte Carlo algorithms. We are able to devise several algorithms and verify through extensive empirical evaluation that they are more effective than the state of the art approach. Surprisingly, the most effective approach, `ReSampleAll`, is based on a simple sampling.

Acknowledgement This research is funded by research grant R-252-000-622-114 by Singapore Ministry of Education Academic Research Fund (project 251RES1607-"Janus: Effective, Efficient and Fair Algorithms for Spatio-temporal Crowdsourcing") and is a collaboration between the National University of Singapore and Télécom ParisTech.

References

1. A. Amarilli, Y. Amsterdamer, and T. Milo. Uncertainty in crowd data sourcing under structural constraints. In *DASFAA(UnCrowd)*, pages 351–359, 2014.
2. G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316, 2009.
3. S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

4. T. Ge, S. Zdonik, and S. Madden. Top-k queries on uncertain data: on score distribution and typical answers. In *SIGMOD*, pages 375–388. ACM, 2009.
5. M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD*, pages 673–686. ACM, 2008.
6. J. Jestes, G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data. *TKDE*, 23(12):1903–1917, 2011.
7. J. Li and A. Deshpande. Ranking continuous probabilistic datasets. *VLDB*, 3(1-2):638–649, 2010.
8. J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *VLDB*, 2(1):502–513, 2009.
9. M. E. Newman, G. T. Barkema, and M. Newman. *Monte Carlo methods in statistical physics*, volume 13. Clarendon Press Oxford, 1999.
10. D. P. O'Leary. Multidimensional integration: partition and conquer. *Computing in Science and Engineering*, 6(6):58–66, 2004.
11. C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.
12. M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *ICDE*, pages 317–328, 2009.
13. M. A. Soliman, I. F. Ilyas, and S. Ben-David. Supporting ranking queries on uncertain and incomplete data. *The VLDB Journal*, 19(4):477–501, 2010.
14. M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.
15. C. Wang, L. Y. Yuan, J.-H. You, O. R. Zaiane, and J. Pei. On pruning for top-k ranking in uncertain databases. *VLDB*, 4(10):598–609, 2011.
16. K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *TKDE*, 20(12):1669–1682, 2008.
17. X. Zhang, G. Li, and J. Feng. Crowdsourced top-k algorithms: An experimental evaluation. *VLDB*, 9(8):612–623, 2016.