Ranking Continuous Probabilistic Datasets

Jian Li
University of Maryland, College Park
lijian@cs.umd.edu

Amol Deshpande
University of Maryland, College Park
amol@cs.umd.com

ABSTRACT

Ranking is a fundamental operation in data analysis and decision support, and plays an even more crucial role if the dataset being explored exhibits uncertainty. This has led to much work in understanding how to rank uncertain datasets in recent years. In this paper, we address the problem of ranking when the tuple *scores* are uncertain, and the uncertainty is captured using *continuous* probability distributions (e.g. Gaussian distributions). We present a comprehensive solution to compute the values of a *parameterized ranking function* (PRF) [18] for arbitrary continuous probability distributions (and thus rank the uncertain dataset); PRF can be used to simulate or approximate many other ranking functions proposed in prior work. We develop exact polynomial time algorithms for some continuous probability distribution classes, and efficient approximation schemes with provable guarantees for arbitrary probability distributions. Our algorithms can also be used for exact or approximate evaluation of *k-nearest neighbor* queries over uncertain objects, whose *positions* are modeled using continuous probability distributions. Our experimental evaluation over several datasets illustrates the effectiveness of our approach at efficiently ranking uncertain datasets with continuous attribute uncertainty.

1. INTRODUCTION

Ranking and top-k query processing are important tools in decision-making and analysis over large datasets and have been a subject of active research for many years in the database community [14]. In recent years, the rapid increase in the amount of uncertain data in a variety of application domains has led to much work on efficiently ranking uncertain datasets. The need for ranking or top-k processing in presence of uncertainty arises in many application domains. In financial applications, we may want to choose the best stocks in which to invest, given their expected performance in the future (which is uncertain at best). In data integration and information extraction over the web, uncertainty may arise because of incomplete data or lack of confidence in the extractions, and these uncertainties must be taken into account when returning the "best" answers for a user query (Figure 1(i)). In sensor networks or scientific databases, we may not know the "true" values of the physical properties being measured because of measurement noises or failures [9] (Figure 1(ii)). Similarly, in any learning or classification task, we often need to choose the best "k" features to use [30]. All of these applications exhibit resource constraints of some form, and we must somehow rank the entities or tuples under consideration in order to select the most relevant objects to focus our attention on.

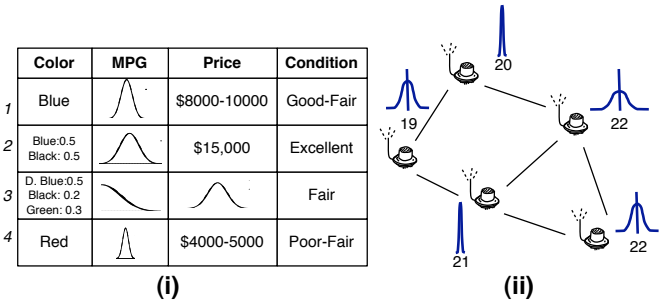


Figure 1: (i) An automatically extracted Car Ads database may contain many uncertainties, which translate into uncertainties over user scores; (ii) Sensor data unavoidably contains complex, continuous uncertainties.

Ranking uncertain datasets is unfortunately much harder than its counterpart in deterministic datasets, mainly because of the complex trade-offs introduced by the score distributions and the tuple uncertainty. This has led to a plethora of ranking functions being proposed in prior literature all of which appear natural at the first glance (we review some of those in the related work section). In prior work [18], we analyzed many of these ranking functions and found that they returned wildly different answers over the same uncertain dataset. To unify the prior ranking functions, we proposed the notion of *parameterized ranking functions ($PRFs$)*, which can simulate or approximate a variety of ranking functions with appropriate parameter choices. We also proposed learning the parameters of the function from user feedback or training data. The algorithms developed there can be used for ranking probabilistic datasets containing tuple uncertainty or discrete attribute uncertainty, but they can not be used to handle continuous distributions directly.

However continuous attribute uncertainty arises naturally in many domains. In most of the applications discussed above (Figure 1), the attributes of interest, and therefore user scores computed using those attributes, are associated with continuous probability distributions. For instance, sensor values, locations (measured using GPS), and feature vectors (in classification applications), are often represented using uniform or Gaussian distributions [9, 3, 4]. Mixtures of Gaussians are widely used in sensing and financial applications [27]. Piecewise polynomial distributions are often used to approximate more complex distributions. Prior work on ranking in probabilistic databases (or more generally, query processing in probabilistic databases with some exceptions) has largely proposed somewhat simplistic solutions to handling continuous distributions.

Proceedings of the VLDB Endowment, Vol. 3, No. 1

Cormode et al. [6] suggested discretizing the continuous distributions to an appropriate level of granularity. Soliman et al. [24] made the first attempt to deal with continuous score distributions directly; however, their main technical tool is the Monte Carlo method which can only obtain an approximate solution in most cases.

In this paper, we address the problem of ranking in presence of continuous attribute uncertainty by developing a suite of exact and approximate polynomial-time algorithms for computing the *rank distribution* for each tuple, i.e., the probability distribution over the rank of the tuple. The rank distributions can be used to order the tuples according to any PRF function, but may be of interest by themselves. For example, Taylor et al. [26] and Guiver et al. [12] treat document scores in an Information Retrieval context as Gaussian random variables, and explicitly compute the rank distributions, which they use to smooth the ranked results. Our main technical contributions in this paper can be summarized as follows:

- We present polynomial-time exact algorithms for computing rank distributions for uniform and piecewise polynomial distributions using the *generating functions* technique (Sections 3 and 4).
- We develop a numerical approximation framework to deal with arbitrary density functions based on our polynomial-time algorithm for piecewise polynomial distributions, by utilizing the *spline* technique. We also present theoretical analysis comparing the spline technique to the discretization method and the *Monte Carlo* method (Section 5).
- We present polynomial-time algorithms for computing *expected ranks* [6] for several important distributions including uniform, Gaussian and exponential distributions. We also propose an efficient approximation scheme, based on Legendre-Gauss Quadrature, for computing the values of a PRF^e function (an important special case of PRF) for arbitrary densities.
- As an application of PRF, we show how to translate a k-nearest neighbor (k-*NN*) query over uncertain objects into a PRF query. This simple observation yields exact or approximation schemes for answering k-*NN* queries.
- We conduct a comprehensive set of experiments over several synthetic datasets to demonstrate the effectiveness and efficiency of our algorithms, and compare them with prior proposed methods (Section 6 and Appendix C).

Due to space constraints, we focus on the first two contributions (and the experimental study) in the body of the paper. The algorithms for PRF^e and expected ranks are presented in the appendix.

2. PRELIMINARIES

We begin with introducing some background and notation, and by formally defining the problem.

2.1 Probabilistic Database Model; Notation

We assume that we are given a probabilistic dataset consisting of n tuples $T = \{t_1, \ldots, t_n\}$, and our goal is to rank the tuples according to a *score function* s, specified as a function of the tuple attribute values. In a deterministic database, each tuple is associated with a single score value, and tuples with higher scores are ranked higher. However, we assume that the database contains significant uncertainty, both tuple existence uncertainty (a tuple may or may not exist in the database) and attribute value uncertainty (wherein we are provided with probability distributions over the values of the attributes). We mainly focus on *continuous* attribute uncertainty in this paper. We assume that the uncertain tuples and attribute values are independent of each other.

For each tuple t_i, we denote its existence probability by $p(t_i)$ or p_i for short. We assume that the attribute value uncertainties are transformed into a single probability distribution over the score of the tuple. If an attribute does not contribute to the score, its uncertainty can be ignored for ranking purposes. For tuple t_i, we denote by s_i the (random) variable corresponding to its score. s_i may be distributed according to a variety of probability distributions, e.g., *uniform, piecewise polynomial*, *Gaussian (Normal)* etc.

s_i	Random variable denoting the score of t_i
μ_i	Probability density function (pdf) of s_i
$\mathsf{supp}(\mu_i)$	Support of μ_i (i.e. $\{x \mid \mu_i(x) \neq 0, x \in \mathbb{R}\}$)
$\rho_i,\ \bar{\rho}_i$	Cumulative density function (cdf) of s_i $\rho_i(\ell) = \int_{-\infty}^{l} \mu_i \mathrm{d}\ell,\ \bar{\rho}_i = 1 - \rho_i$
$\mathsf{Pr}(r(t_i) = j)$	Positional prob. of t_i being ranked at position j
PRF	Parameterized ranking function $\Upsilon_\omega(t) = \sum_{i>0} \omega(t,i)\mathsf{Pr}(r(t) = i)$
$PRF^\omega(h)$	Special case of PRF: $\omega(t,i) = w_i, w_i = 0, \forall i > h$
$PRF^e(\alpha)$	Special case of PRF^ω: $w_i = \alpha^i, \alpha \in \mathbb{C}$
PRF^ℓ	Special case of PRF^ω: $w_i = n - i$
$I = [l_I, u_I]$	A small interval and its range

Table 1: Notation

We denote by μ_i the probability density function (pdf) of s_i. The *support* of μ_i is defined to be the set of reals where μ_i is nonzero, i.e., $\mathsf{supp}(\mu_i) = \{x \mid \mu_i(x) \neq 0, x \in \mathbb{R}\}$. The cumulative density function (cdf) of s_i is defined to be: $\rho_i(\ell) = \mathsf{Pr}(s_i \leq \ell) = \int_{-\infty}^{\ell} \mu_i(x)\mathrm{d}x$. Let $\bar{\rho}_i(\ell) = 1 - \rho_i(\ell)$.

The set of all possible worlds corresponding to the probabilistic dataset is denoted by PW. Note that we have an uncountable number of possible worlds if there is any continuous attribute uncertainty. We use $r_{pw} : T \to \{1, \ldots, n\} \cup \{\infty\}$ to denote the rank of the tuple t in a possible world pw according to s. If t does not appear in the possible world pw, we let $r_{pw}(t) = \infty$. We say t_1 *ranks higher* than t_2 in the possible world pw if $r_{pw}(t_1) < r_{pw}(t_2)$. For each tuple t, we define a random variable $r(t)$ which denotes the rank of t. In other words, $\mathsf{Pr}(r(t) = k)$ is the probability measure of the set of possible worlds where t is ranked at position k.

DEFINITION 1. *The* positional probability *of a tuple t_i and position j is defined to be the probability that t_i is ranked at position j, i.e.,* $\mathsf{Pr}(r(t_i) = j)$.

2.2 Parameterized Ranking Function

In this paper, we focus on ranking the tuples in an uncertain dataset using a *parametrized ranking function* (PRF), introduced in recent work to integrate different approaches to ranking uncertain data [18]. As illustrated in that prior work, PRFs can approximate a variety of different ranking functions through appropriate choices of the parameter values. That prior work also provides algorithms for learning the parameter values from user preferences or training data. In this paper, we assume that the PRF function is already chosen or learned, and we focus on computing the PRF function values for all tuples in presence of continuous attribute uncertainty. We begin with a brief review of PRFs.

DEFINITION 2. *Let $\omega : T \times \mathbb{N} \to \mathbb{C}$ be a* weight function *that maps a tuple-rank pair to a complex number. The* parameterized ranking function (PRF)*, $\Upsilon_\omega : T \to \mathbb{C}$ in its most general form is defined to be:*

$$\Upsilon_\omega(t) = \sum_{i>0} \omega(t,i)\mathsf{Pr}(r(t) = i). \quad (1)$$

A top-k query returns the k *tuples with the highest $|\Upsilon_\omega|$ values.*

We identify several special cases of the PRF function.

- $\underline{PRF^{\omega}(h)}$: One important class of ranking functions is when $\omega(t,i) = w_i$ (i.e., ω is independent of t) and $w_i = 0, \forall i > h$ for some positive integer h. Such ranking functions are often used in domains such as information retrieval and machine learning. For instance, the weight function $\omega(i) = \frac{\ln 2}{\ln(i+1)}$ is often used in the context of ranking documents in information retrieval [15]. Further, the probabilistic threshold top-k (PT-k) [13] can be seen as a special case of $PRF^{\omega}(h)$.

- $\underline{PRF^{e}(\alpha)}$: PRF^{e} is a special case of the PRF^{ω} function, where $w_i = \omega(i) = \alpha^i$, where α is a constant and may be a real or a complex number. As argued in [18], PRF^{e} are highly suitable for ranking in probabilistic databases because of the exponentially decreasing weight function (assuming $|\alpha| < 1$), and they admit very efficient algorithms.

- $\underline{PRF^{\ell}}$: Finally, PRF^{ℓ} (PRF linear) is another special case of the PRF^{ω} function, where $w_i = \omega(i) = n - i$.

The PRF^{ℓ} function bears a close similarity to the notion of *expected ranks* [6], and the expected rank can be easily computed using the PRF^{ℓ} value assuming independence. In fact, if we only have attribute uncertainty, then the two produce an identical ranking (see Appendix B for more details).

2.3 Problem Definition

Given the above, we can formally define the problem that we address in this paper: given (1) a probabilistic dataset containing tuples $t_i, i = 1, ..., n$, and for each tuple its existence probability p_i, and a (continuous) probability density function μ_i of its score s_i, and (2) a weight function $\omega : T \times \mathbb{N} \rightarrow \mathbb{C}$, find the ranking of the tuples according to the PRF function.

In most cases, we actually solve the problem of evaluating the *positional probabilities*, $\mathsf{Pr}(r(t_i) = j), \forall i, \forall j$, i.e., for each tuple t_i and each rank j, we compute the probability that t_i is ranked at position j across the possible worlds. Given these, we can compute the PRF function values quite easily, after which ranking only takes $O(n \log(n))$ time. As expected, the problem is very hard to solve in its full generality, in large part because we need to perform complex numerical integrations on arbitrary probability densities.

3. EXACT ALGORITHMS

We begin with presenting efficient polynomial-time algorithms for exact computation of the PRF functions when the probability distributions on the scores are either uniform or piecewise polynomial. We begin with showing that the *generating functions* framework proposed in [18] can be extended to handle continuous distributions. We consider only attribute value uncertainty in this section, and discuss tuple uncertainty in the next section.

3.1 Generating Functions Framework

Let us begin with looking at the formula for computing positional probability $\mathsf{Pr}(r(t_i) = j)$ closely. If we were trying to explicitly write the formula for positional probability, it would end up being a high dimensional integral, which is typically expensive to evaluate. However, the following states that we can represent the n positional probabilities compactly using their *generating function*, which is a single 1-dimensional integral. This critical transform makes exact algorithms possible for uniform and piecewise polynomial distributions and also makes the problem more amenable to numerical approximation techniques.

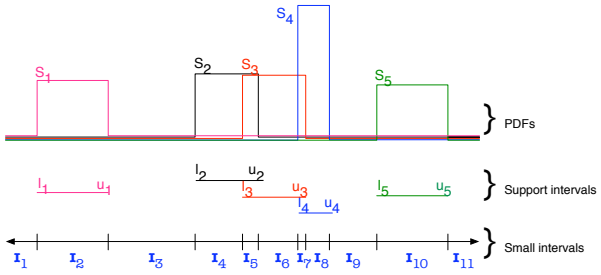


Figure 2: Illustration of support intervals and small intervals for five tuples with uniform probability distributions

THEOREM 1. *For any tuple t_i, define*

$$\mathcal{F}_i(x,\ell) = \prod_{j \neq i} \Big(\rho_j(\ell) + \bar{\rho}_j(\ell) x \Big), \quad \textit{and} \tag{2}$$

$$F_i(x) = x \int_{-\infty}^{\infty} \mu_i(\ell) \mathcal{F}_i(x,\ell) \mathrm{d}\ell \tag{3}$$

Then, $F_i(x)$ is the generating function for the sequence $\{\mathsf{Pr}(r(t_i) = j)\}_{j \geq 1}$, i.e., $F_i(x) = \sum_{j \geq 1} \mathsf{Pr}(r(t_i) = j) x^j$.

In light of Theorem 1 (proof is presented in the appendix), we can see that the task of computing the positional probabilities reduces to expanding the polynomial F_i in terms of x^js and obtaining the coefficients.

3.2 Uniform Distribution

In this section, we consider the case where μ_i is uniform over its *support interval* $[l_i, u_i]$. In that case, the cdf of s_i is a piecewise linear function, i.e., $\rho_i(\ell) = \mathsf{Pr}(s_i \leq \ell) = 0$, for $\ell < l_i$; $\rho_i(\ell) = \frac{\ell - l_i}{u_i - l_i}$, for $l_i \leq \ell \leq u_i$; $\rho_i(\ell) = 1$, for $\ell > u_i$.

3.2.1 *Expanding $F(x)$*

For clarity, we assume that all numbers in $\cup_{j=1}^{n} \{l_j, u_j\}$ are distinct throughout the paper. The general case where not all points are distinct can be handled easily. Those $2n$ points partition the real line into exactly $2n+1$ intervals (see Figure 2 for an example with 5 tuples). For ease of exposition, we call these intervals *small intervals* (in contrast to the *support intervals* $[l_i, u_i]$).

For the small interval I, let l_I and u_I denote its left and right endpoints respectively. Denote the set of small intervals (from left to right) by $\mathcal{I} = \{I_j\}_{j=1}^{2n+1}$ and the subset of those contained in support interval $[l_i, u_i]$ by $\mathcal{I}_i$, i.e., $\mathcal{I}_i = \{I \mid l_I \geq l_i \wedge u_I \leq u_i\}$.

EXAMPLE 1. *In the example shown in Figure 2, $\mathcal{I}_2 = \{I_4, I_5\}$, whereas $\mathcal{I}_3 = \{I_5, I_6, I_7\}$.*

Since $\mathcal{I}$ is a disjoint partition of the real line and since $\mu_i(l)$ is equal to $\frac{1}{u_i - l_i}$ in the interval $[l_i, u_i]$ and 0 otherwise, we have that:

$$F_i(x) = \frac{x}{u_i - l_i} \sum_{I \in \mathcal{I}_i} \int_{l_I}^{u_I} \mathcal{F}_i(x,\ell) \mathrm{d}\ell \tag{4}$$

Thus to be able to expand $F_i(x)$, we just need to be able to expand $\int_{l_I}^{u_I} \mathcal{F}_i(x,\ell) \mathrm{d}\ell$ for all small intervals I.

Now, it is not hard to see that $\bar{\rho}_j(\ell)$ and $\rho_j(\ell)$ are linear functions for all $1 \leq j \leq n$ in each small interval I. Thus, for $\ell \in I$, we can write:

$$\rho_j(\ell) + \bar{\rho}_j(\ell) x = a_{I,j} + b_{I,j} \ell + c_{I,j} x + d_{I,j} x \ell.$$

In particular, we have:

$$(a_{I,j}, b_{I,j}, c_{I,j}, d_{I,j}) = \begin{cases} (1,0,0,0), & u_I \leq l_j; \\ \frac{1}{u_j - l_j} (-l_j, 1, u_j, -1), & I \in \mathcal{I}_j; \\ (0,0,1,0), & l_I \geq u_j. \end{cases}$$

Hence, within each small interval I:

$$\mathcal{F}_i(x,\ell) = \prod_{j\neq i}\Big(a_{I,j} + b_{I,j}\ell + c_{I,j}x + d_{I,j}x\ell\Big)$$

can be easily expanded in the form of $\sum_{j,k}\alpha_{I,i,j,k}x^j\ell^k$ in polynomial time. Therefore, we can write:

$$\int_{l_I}^{u_I}\mathcal{F}_i(x,\ell)\mathrm{d}\ell = \sum_{j,k}\left(\alpha_{I,i,j,k}\int_{l_I}^{u_I}\ell^k\mathrm{d}\ell x^j\right) = \sum_{j,k}\left(\alpha_{I,i,j,k}\frac{1}{k+1}\big(u_I^{k+1}-l_I^{k+1}\big)x^j\right).$$

Summing over all intervals in $\mathcal{I}_i$, and combining with Theorem 1 and equation l(4), we can get

$$\mathsf{Pr}(r(t_i)=j) = \frac{1}{u_i-l_i}\sum_{I\in\mathcal{I}_i}\sum_k\frac{\alpha_{I,i,j,k}}{k+1}\big(u_I^{k+1}-l_I^{k+1}\big) \quad (5)$$

3.2.2 Implementation and Analysis of Running Time

For each small interval $I_j \in \mathcal{I}$, let M_j (M_j' or M_j'') be the set of tuples whose score interval contains (lies to the left or right) I_j. i.e., $\{t_i \mid I_j \subseteq \mathcal{I}_i\}$ ($\{t_i \mid u_{I_j} \leq l_i\}$ or $\{t_i \mid l_{I_j} \geq u_i\}$). Let $m_j = |M_j|$, $m' = |M_j'|$, $m'' = |M_j''|$ and $m = \sum_j m_j$. We call m_j the *overlap number* on I_i

Naively constructing each $\mathcal{F}_i(x,\ell)$ in each small interval and expanding the polynomial from scratch is too expensive (we need to expand at most $O(n^2)$ polynomials and expanding each of them could take up to $O(n^3)$ time). We notice the significant similarity of the polynomials that we can take advantage of to reduce the running time. For example, in a interval I, $\mathcal{F}_i(x,\ell)$ and $\mathcal{F}_j(x,\ell)$ differ in only two multiplicative terms.

Consider a tuple i and a small interval $I_j \in \mathcal{I}_i$. Define

$$\tilde{\mathcal{F}}_{I_j}(x,\ell) = \prod_{j=1}^{n}\Big(\rho_j(\ell)+\bar{\rho}_j(\ell)x\Big) = \prod_{j\in M_j}\Big(\frac{-l_j+\ell+u_jx-x\ell}{u_j-l_j}\Big)x^{m_j''}$$

From (2), we can see that on interval I_j,

$$\mathcal{F}_i(x,\ell) = \tilde{\mathcal{F}}_{I_j}(x,\ell)\frac{u_i-l_i}{-l_i+\ell+u_ix-x\ell} \quad (6)$$

Our algorithm first constructs and expands $\tilde{\mathcal{F}}_j(x,\ell)$ for each small interval $I_j \in \mathcal{I}$ in a straightforward manner. This can be done in $O(m_j^3)$ time. Then, we compute the expansion for $\mathcal{F}_i(x,\ell)$ for each $i \in M_j$, for small interval I_j, based on 6, which needs $O(m_j^2)$ time. We summarize the overall steps in Algorithm **PRF-UNIFORM**. It is not hard to see that this process takes $O(\sum_j m_j^3)$ time, provided the intervals $\mathcal{I}$ are already computed and sorted.

Algorithm 1: **PRF-UNIFORM**

```
Sort ∪_{j=1}^{n}{l_j, u_j} in an increasing order and construct
  intervals I_j, 0 ≤ j ≤ 2n;
F̃_0(x, ℓ) = 1;
for t = 1 to 2n do
    Expand F̃_t(x, ℓ) ;
    for each t_i ∈ M_t do
        Expand F_i(x, ℓ) according to (6);  (Note that we can
        obtain coefficients α_{I_t,i,j,k}s in this step);
Compute ϒ(t_i) according to (5) and (1);
Return k tuples with largest |ϒ| values;
```

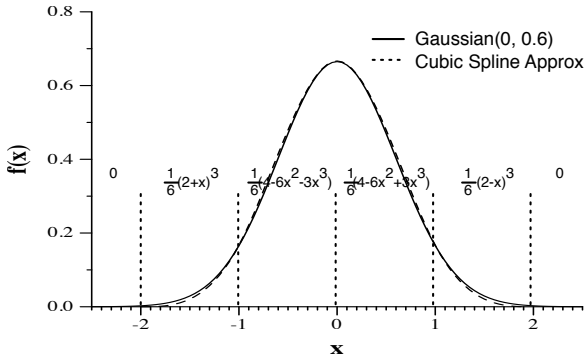


Figure 3: Approximating a Gaussian distribution using a Cubic Spline with 6 pieces (e.g. in the interval $[-2,-1]$, the approximation is done using $\frac{1}{6}(2+x)^3$).

4. EXTENSIONS

Due to space constraints, we briefly list several significant extensions and generalizations of the basic algorithm above in this section. The details can be found in the appendix. Table 2 summarizes the running time complexity of these algorithms.

Computing $PRF^\omega(h)$: Since $\omega(j) = 0$ for all $j > h$, we only need the probability values $\mathsf{Pr}(r(t_i) = j)$ for $j \leq h$. Therefore, we only need to expand $F_i(x)$ up to the x^h term. Since h is typically much smaller than the number of tuples n, the improvement can be significant.

Computing PRF^e: As in [18], we have the same relationship between the generating function and the $PRF^e(\alpha)$ value:

$$\Upsilon(t_i) = \sum_{j\geq 1}\mathsf{Pr}(r(t_i)=j)\alpha^j = F_i(\alpha).$$

Hence, instead of expanding $\mathcal{F}_i(x,\ell)$ as a polynomial with two variables x and ℓ, we can substitute the variable x with the numerical value α and expand $\mathcal{F}_i(\alpha,\ell)$ instead by treating it as a polynomial with a single variable ℓ.

Computing Expecting Ranks and PRF^ℓ: Recall that PRF^ℓ is a special case of the PRF function where the weight function is linear, i.e., $w_i = \omega(i) = n - i$. Aside from being a natural weight function, another key reason to study PRF^ℓ is its close relationship to *expected ranks*. In Appendix B, we describe a series of algorithms for efficiently ranking according to PRF^ℓ or expected ranks for a variety of continuous distributions including Gaussian, exponential, piecewise polynomial, and uniform.

Combining with Tuple Uncertainty: The results described so far can be easily extended to handle tuple uncertainty. Let p_i denote the existence probability associated with tuple t_i. All we need to do is to replace the definition of $\rho_i(\ell)$ with: $\rho_i(\ell) = \mathsf{Pr}(t_i \text{ does not exist or } s_i \leq \ell) = (1-p_i) + p_i\int_{-\infty}\mu_i(x)\mathrm{d}x$ and still let $\bar{\rho}_i(\ell) = 1 - \rho_i'(\ell)$. It can be seen that Theorem 1 still holds. Hence, all algorithms developed can be applied with the new definitions. The running time is reported in Table 2.

Piecewise Polynomial Distributions: Finally, and perhaps most importantly, this is the last polynomially solvable case that we have been able to identify for computing general PRF functions. However, this class of distributions allows us to connect to the rich literature of *approximation theory* from which we can borrow powerful techniques to approximate arbitrary density functions. We elaborate on that in the next section.

For a piecewise polynomial pdf, the density function is expressed using different (typically low-degree) polynomials in different intervals. Figure 3 shows an example (it is also a very good approximation to a Gaussian distribution).

	PRF	$PRF^{\omega}(h)$	PRF^{e}	PRF^{ℓ}
Uniform	$O(\sum_j m_j^3)$	$O(\sum_j (m_j^2 \min(m_j, h)))$	$O(\sum_j m_j^2)$	$O(\sum_j m_j)$
Uniform+TU	$O(\sum_j m_j^2(m_j + m_j''))$	$O(\sum_j (m_j^2 \min(m_j + m_j'', h))$	$O(\sum_j m_j(m_j + m_j''))$	$O(\sum_j m_j)$
P-Poly(γ)	$O(\gamma^2 \sum_j m_j^3)$	$O(\gamma^2 \sum_j (m_j^2 \min(m_j, h)))$	$O(\gamma^2 \sum_j m_j^2)$	$O(\gamma^2 \sum_j m_j)$
P-Poly(γ)+TU	$O(\gamma^2 \sum_j m_j^2(m_j + m_j''))$	$O(\gamma^2 \sum_j m_j^2 \min(m_j + m_j'', h))$	$O(\gamma^2 \sum_j m_j(m_j + m_j''))$	$O(\gamma^2 \sum_j m_j)$

Table 2: Running Times. TU means tuple uncertainty and P-Poly(γ) indicates piecewise polynomial distributions with maximum degree γ. We assume that all small intervals are already sorted. Otherwise, we have another additive factor of $|\mathcal{I}| \log(|\mathcal{I}|)$ for each entry. The summation is over all small intervals. Recall m_j is the overlap number for small interval I_j.

The algorithm for computing the PRF values given that all tuples have piecewise polynomial pdfs, is quite similar to the one for uniform distribution. We partition the real line into small intervals such that the density function of each tuple can be represented as a single polynomial in each small interval. Consider the small interval $I = [l_I, u_I]$. Assume the pdf of s_i is $\mu_i(x) = \sum_{j=0}^{h_i} a_{i,j} x^j$ for all $x \in I$. By indefinite integration, the cdf of s_i over I is $\rho_i(x) = \sum_{j=0}^{h_i} \frac{a_{i,j}}{j+1} x^{j+1} + C_{i,I}$ where $C_{i,I}$ is a constant which can be determined by the equation $\rho_i(l_I) = \int_{-\infty}^{l_I} \mu_i(x)\mathrm{d}x$. Thus, we know that every term inside the integral in (3) is a polynomial of x and ℓ, and their product can be easily expanded in polynomial time. The rest is the same as in the uniform distribution case and we can use a similar technique to (6) to improve the running time. See Table 2 for the exact running time.

We finally note that discrete distributions (or a mixture of above cases) can be handled in the same way since their cdfs are also piecewise polynomials (in fact piecewise constants).

5. ARBITRARY PROBABILITY DENSITIES

For arbitrary probability density functions, the term inside the integral of (3) is not a polynomial any more, and in fact may not even have a closed form expression. This is true for one of the most widely used probability distributions, namely the Gaussian distribution. For most such distributions, the best we can hope for is an efficient approximation. In this section, we first present a general framework for approximate ranking in presence of arbitrary density functions through use of piecewise polynomial approximations (specifically, *cubic spline* approximation). We then analyze the approximation quality of our cubic spline technique and compare it with the discretization method and the Monte Carlo method.

5.1 A Generic Approximation Framework

The class of piecewise polynomials, also called *splines*, is known to be very powerful at approximating other functions. There are many different types of splines and the study of them has a long history with a huge body of literature. In this paper, we focus on the perhaps most widely used one, cubic spline, in which each piece of polynomial is of degree at most 3.

The high level idea of our approximation framework is very simple: For each tuple, we use one cubic spline to approximate the probability density function of its score, then we apply the exact polynomial-time algorithm developed in the previous section to compute the PRF values.

Now, we briefly discuss how to use cubic spline to approximate an arbitrary function $\mu(x)$. We assume $\mu(x)$ is defined over a closed interval $[l, u]$ and we can evaluate the value of $\mu(x)$ and the first derivative $\frac{\mathrm{d}\mu}{\mathrm{d}x}(x)$ at any $l \le x \le u$. We choose k equally-spaced breaking points τ_i such that $l = \tau_1 < \tau_2 < \ldots < \tau_k = u$ and $\tau_{i+1} - \tau_i = \frac{u-l}{k-1}$ for all i. For each interval $[\tau_i, \tau_{i+1}]$, we construct a polynomial $P_i(x)$ of degree at most 3 such that the value and the first derivative of $P_i(x)$ agree with $\mu(x)$ at τ_i and τ_{i+1}, i.e., $P_i(\tau_i) = \mu(\tau_i)$, $\frac{\mathrm{d}P_i}{\mathrm{d}x}(\tau_i) = \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i)$, $P_i(\tau_{i+1}) = \mu(\tau_{i+1})$, $\frac{\mathrm{d}P_i}{\mathrm{d}x}(\tau_{i+1}) = \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_{i+1})$.

It can be shown that (see e.g., [8, pp. 40] for the derivation)

$$P_i(x) = c_{i,1} + c_{i,2}(x - \tau_i) + c_{i,3}(x - \tau_i)^2 + c_{i,4}(x - \tau_i)^3$$

where the coefficients can be computed as: $c_{i,1} = \mu(\tau_i)$, $c_{i,2} = \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i)$, $c_{i,4} = \frac{1}{(\tau_{i+1}-\tau_i)^2}\big(\frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i) + \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_{i+1}) - 2\frac{\mu(\tau_i)-\mu(\tau_{i+1})}{\tau_i - \tau_{i+1}}\big)$, $c_{i,3} = c_{i,4}(\tau_{i+1} - \tau_i) + \frac{1}{\tau_{i+1}-\tau_i}\big(\frac{\mu(\tau_i)-\mu(\tau_{i+1})}{\tau_i - \tau_{i+1}} - \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i)\big)$

We can easily see that the running time to construct a spline approximation for one tuple is only linear in the number of breaking points. In general, more breaking points implies better approximation quality, however, this will also increase the running time for both constructing the splines and in particular, of the exact algorithm for computing PRF. We empirically evaluate this trade-off in Section 6. It is possible to use higher order splines or unequal length partitions which, sometimes, are better choices for approximation. Exploring these opportunities is left for future work.

5.2 Theoretical Comparisons

Here we compare the asymptotic behavior of convergence of the spline approximation with other two methods that have been considered in prior work, the *Monte Carlo method* and the *discretization method*. Our analysis reveals interesting precision-complexity trade-offs among various methods and suggest that spline approximation is more advantageous when a high precision is required, while the Monte Carlo method is more efficient otherwise.

For completeness of the paper, we briefly describe the Monte Carlo method and the discretization method. Monte Carlo simulation is a widely used and much simpler method to approximate a variety of quantities such as probability, expectation etc., and it can be used to approximate PRF functions as well. To approximately rank a dataset using Monte Carlo simulation, we draw N independent random samples from the probabilistic database D (each sample being a possible world), and sort every sample. Let $r_i(t)$ be the rank of tuple t in the i^{th} sample. Our estimate of $\Upsilon_\omega(t)$ is simply:

$$\tilde{\Upsilon}_\omega(t) = \frac{1}{N} \sum_{i=1}^{N} \omega(t, r_i(t)).$$

The method of discretizing continuous distributions has been suggested in [6], however, no further detail and analysis is provided. We consider the following natural discretization: We partition the $\mathsf{supp}(\mu_i)$ into N equal-length intervals $I_{i,1}, I_{i,2}, \ldots, I_{i,N}$. The number N depends on the granularity we decide to use. Then, t_i is replaced by a set of x-tuples (the set of tuples are mutually exclusive) $t'_{i,1}, \ldots, t'_{i,N}$ such that $t'_{i,j}$ has a fixed score $s_{i,j} =$ midpoint of $I_{i,j}$ and existence probability $\mathsf{Pr}(t_{i,j}) = \int_{I_j} \mu_i(x)\mathrm{d}x$. Another popular choice to approximate general continuous distributions is to use histograms, where we partition the domain into intervals and use a uniform distribution to approximate the true distribution in each interval. It can be seen as another way of discretization and also a degenerate case of piecewise polynomial method where each piece is just a constant.

In order to prove anything interesting, we have to make some assumptions; we discuss their generality and applicability later.

Assume that for each i, $\mathsf{supp}(\mu_i)$ is an interval of length $O(1)$ and $\mu_i(x)$ and its first four derivatives are bounded for all $x \in \mathsf{supp}(\mu_i)$. We stick with the cubic spline approximations.

THEOREM 2. *We partition each* $\mathsf{supp}(\mu)$ *into small intervals such that the maximum length* Δ *of any small interval is* $O(n^{-\beta})$ *for some* $\beta > \frac{3}{8}$ *where* n *is the number of tuples. If we use cubic spline to approximate* μ_i *based on the partition and compute the approximation* $\widehat{\Upsilon}_\omega(t)$ *by the algorithm in Section 3, then:*

$$|\widehat{\Upsilon}_\omega(t) - \Upsilon_\omega(t)| \leq O(n^{3/2-4\beta}).$$

The proof can be found in Appendix E.

Assuming bounded length of the support and continuity of the first derivative of μ_i for each i, we can prove the following asymptotic convergence for the discretization method. The proof is similar to Theorem 2 and can be found in the full version of the paper. We note that the same bound also holds for the histogram method.

THEOREM 3. *If we replace the continuous distribution* μ_i *with a discrete distribution over* $O(n^\beta)$ *points (in the way described above) for some* $\beta > 3/2$*, and we compute the* PRF *value* $\widehat{\Upsilon}_\omega(t)$ *for the discrete distribution. Then, we have:*

$$|\widehat{\Upsilon}_\omega(t) - \Upsilon_\omega(t)| \leq O(n^{3/2-\beta}).$$

On the other hand, the following fact about the Monte Carlo method is a well known folklore (see [20, Ch.11]) : With $N = \Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ samples, we can get a approximated $\Upsilon_\omega(t)$ value within an additive error ϵ with probability at least $1-\delta$. To better compare it with the other two methods, we rephrase this fact as follows:

THEOREM 4. *With* $N = \Omega(n^\beta \log\frac{1}{\delta})$ *samples, the Monte Carlo method yields an approximation* $\tilde{\Upsilon}_\omega(t)$ *of* $\Upsilon_\omega(t)$ *such that*

$$\mathsf{Pr}\left(|\tilde{\Upsilon}_\omega(t) - \Upsilon_\omega(t)| \leq O(n^{-\beta/2})\right) \geq 1-\delta$$

For ease of comparison, we use N to denote (1) the number of small intervals into which we partition the support of one tuple for the spline technique, (2) the number of discrete points which we use to approximate a continuous pdf for discretization method, and (3) the number of samples we take for Monte Carlo method. Doubling N is roughly equivalent to doubling the execution time for each method. With Theorems 2, 3 and 4, the precision-complexity trade-offs of the three methods become clear: spline method has a relatively high overhead $O(\sum_j m_j^3)$, the discretization method has an $O((nN)^2)$ implementation [18], while Monte Carlo method only needs $O(n \log n)$ time for each sample. However, roughly speaking, doubling N increases the precision by $2^4 = 16$ times for the spline method, 2 times for discretization, but only by $\sqrt{2}$ times for Monte Carlo method. Therefore, the spline method starts to outperform the other two when higher precision is required. See Figure 4 for a clearer illustration of the trade-off.

In many applications, very high precision is often required. We give a contrived but still simple example here. Consider the problem of ranking a subset of 10 tuples $\{t_i\}_{i=1}^{10}$, in a database which has 20 tuples $\{t_i\}_{i=1}^{20}$, by their probability of being the top answer, i.e., $\mathsf{Pr}(r(t) = 1)$ (this is a special case of PRF^ω). Assume the score s_i of t_i is certain and around 6 for $11 \leq i \leq 20$. The other 10 tuples are the ones we want to rank and their scores follow Gaussian distribution with mean around 0 and standard deviation around 1. By a rough analytic estimation, we can show that $\Upsilon(t_i) = \mathsf{Pr}(t_i = 1)$ for all $1 \leq i \leq 10$ are in an order of magnitude 10^{-11}, and it is likely that the Monte Carlo estimates for them are all zero with even 10^9 samples, thus requiring an astronomical number of samples for accurate estimates. However, by partitioning $[-10, 10]$ into 10^5 small intervals ($\beta = 4$), the spline approximation can give us an estimate with a 10^{-14} error by Theorem 2, which should be a fairly good estimate of $\Upsilon(t_i)$.

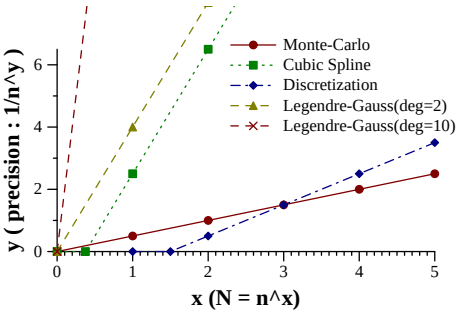


Figure 4: Asymptotic precision-complexity trade-offs for various methods. Note the meaning of the axes: $N = n^x$, & precision is of order $1/n^y$ (thus, a steeply increasing curve indicates that approximation quality increases rapidly with higher N). Legendre-Gauss method for approximating PRF^e is discussed in the appendix. Constants hidden in $O()$ are ignored.

Now, we discuss the assumptions we made for Theorem 2 and Theorem 3. For some distributions, for example, the Gaussian distribution, the support is not bounded. However, we can truncate the distribution and ignore the tail with minuscule probability. For example, for a random variable x following the standard Gaussian distribution $\mathcal{N}(0,1)$, the probability that $x > 6$ is less than 2×10^{-9}. Note that the truncation needs to be done by taking the precision requirement into consideration. In fact, we can easily show that the total estimation error is at most the sum of the approximation error and the truncation error. The assumption that $|\mathsf{supp}(\mu_i)| = O(1)$ captures the fact that (most of) the probability mass of a distribution concentrates within a bounded range and does not scale with the size of the database. For instance, the variance of the temperature reported by a sensor does not scale with the number of sensors deployed and the number of readings that are stored. Assuming certain continuity of the density function and its derivatives is necessary for most approximation techniques with provable bounds, and is usually met in practice.

Finally, we would like to remark that all analyses done here are worst case analyses; better bounds may be obtained if more information about the dataset is provided. For example, if the variances of the PRF values are small, less samples are needed to obtain an approximation with the prescribed error bound (see e.g. [7] [1]).

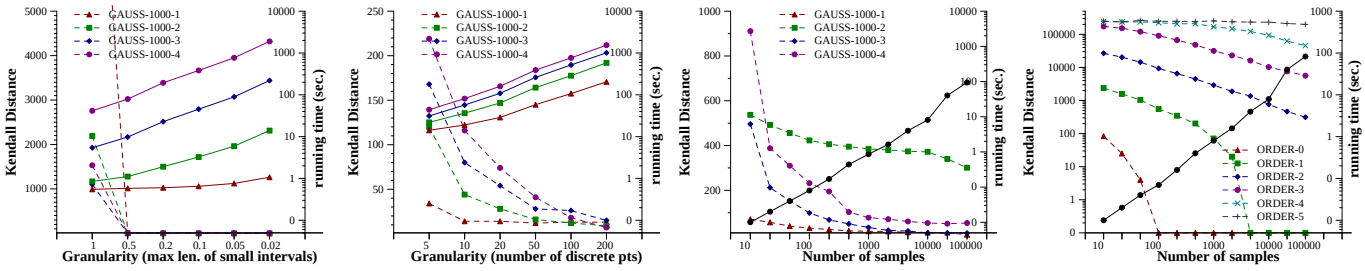
6. EMPIRICAL EVALUATION

In this section, we present results from an extensive empirical study over several datasets to illustrate the effectiveness and efficiency of our algorithms and to compare them with the Monte Carlo method and other heuristics proposed in prior work.

Datasets: We mainly use several synthetic datasets with various distributions and deviations to study our algorithms.

- UNIFM-n-d: We have 40 datasets, each of which contains a mixture of *certain* tuples and *uncertain* tuples with uniformly distributed scores. All scores are between $[0, 10000]$. $n(= 10000, \ldots, 100000)$ is the number of tuples and $d(= 1, 2, 3, 4)$ indicates the degree of "variance" of the data. Specifically, for $d = 1$ ($2, 3, 4$ resp.), we have 10% (%30, %50, %90 resp.) uncertain tuples and the average length of the support intervals is 2 (5, 10, 20 resp.).
- GAUSS-n-d: We have 40 datasets which is a mixture of cer-

[1]Actually, the main result of [7] is an estimation with a relative error bound. It is straightforward to translate the result in terms of additive error. However, the worst case is the same as in Theorem 4.



(a) Spline (b) Discretization (c) Monte Carlo (d) Monte Carlo

Figure 5: The comparison of various methods for computing general PRF (weight function $\omega(t, j) = 1/j$). Solid lines indicate the running times (with axes drawn on the right hand side), whereas dashed lines indicate the Kendall distance (an error measure).

tain tuples and uncertain tuples with normally distributed scores. All scores and the means of Gaussians are uniformly chosen between $[0, 1000]$. $n(= 1000, \ldots, 10000)$ and $d(= 1, 2, 3, 4)$ have the same meaning as in the uniform case. Specifically, for $d = 1$ (2, 3, 4 resp.), we have 10% (%30, %50, %90 resp.) uncertain tuples and the average standard deviation of the uncertain scores is 2 (5, 10,20 resp.).

- ORDER-d: There are 5 datasets that are specially designed to test the convergence of various methods. Each of them has 1000 tuples $\{t_1, \ldots, t_{1000}\}$. All scores are normally distributed with the same standard deviation 1. In ORDER-d (where $d = 0, 1, 2, 3, 4, 5$), the mean of the score of t_i is $i \cdot 10^{-d}$. Note that as d increases, the Gaussian distributions have means very close to each other, and become harder to separate from each other.

Setup: All the algorithms were implemented in C++, and the experiments were run on a 2GHz Linux PC with 2GB memory. We compare the following algorithms with varying parameters:

- SPLINE: The exact algorithm for uniform and spline distributions (developed in Section 3.2) and the spline approximation. For spline approximation, we run the algorithms on various granularities, i.e., the maximum length of the small intervals.
- DISC: The discretization method (outlined in Section 5.2). The parameter is the number of discrete points that we use to replace a continuous distribution. After discretizing the continuous distributions, we use the algorithm developed in [18] to compute PRF value for x-tuples.
- MC: We run the Monte Carlo method (outlined in Section 5.2) with different number of samples.

To measure the approximation quality of an algorithm, we use the *Kendall's tau distance* between the true ranking and the ranking obtained by the algorithm. Kendall's tau distance between two rankings is defined to be the number of reversals, i.e., tuple pairs that are in different order in the two rankings [16].

6.1 Spline vs. Monte Carlo vs. Discretization

We begin with considering the speed of convergence of various approximation methods by varying the granularity or the number of samples. Our first set of experiments is to approximate an arbitrary PRF function for the GAUSS datasets using SPLINE, DISC and MC. The weight function we use is $\omega(t_i, j) = 1/j$. Since no polynomial-time algorithm is known to compute PRF values with such a weight function for general distributions, there is no easy way to know the true (ground) ranking. We however take the presumed truth to be the ranking obtained by SPLINE with a very fine granularity (the length of each small interval is 0.005). As we can see from Figure 5(a), when the granularity is finer than 0.5, SPLINE converges to a fixed ranking. We also check the changes of the actual PRF value for the tuples – when the granularity is finer than 0.1, is less than 10^{-10}. Therefore, we can be confident that the presumed true ranking is actually the true ranking. We note that the running time of SPLINE depends heavily on the overlap numbers.

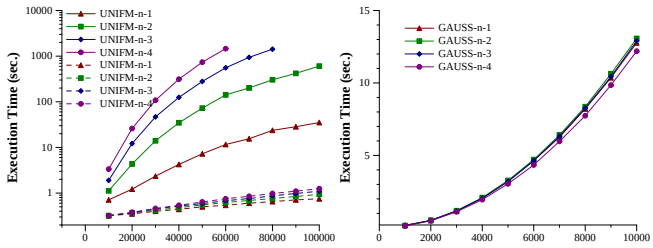
Figure 5(c) shows the convergence rate and running time of MC. We can see that MC converges slower than SPLINE in all cases, especially when the average standard deviation becomes larger. This is not quite surprising since the convergence rate of MC highly depends on the variance of the random variable – a higher variance in general implies a slower convergence rate. A closer look at the actual approximated PRF values reveals that the changes are in a order of magnitude of $10^{-3} \sim 10^{-5}$ even when more than 10000 samples are used. The running time of MC is roughly linear in the number of samples, and does not depend on the overlap number as oppose to SPLINE. So, the running time curves for all GAUSS-1000-d datasets are roughly the same and we only plot one of them. From Figure 5(b), we can see that the convergence of DISC is slower than SPLINE, but much faster than MC. By replacing a Gaussian with a distribution over only $k = 5$ discrete points, we can get an approximate ranking with less than 200 reversals w.r.t. the true ranking.

Next we compare the behaviors of three algorithms on ORDER-datasets. Since all Gaussian distributions have the same standard deviation, a Gaussian distribution with a higher mean *stochastically dominates* one with a lower mean, thus having a higher Υ_ω value for any positive decreasing weight function ω. So we know the true ranking is $\{t_{1000}, t_{999}, \ldots, t_1\}$. Both SPLINE and DISC can find the exact ranking, with even the coarsest granularity, so we omit their curves. This phenomena may be due to the regularity in the datasets and the approximation algorithms, which result in homogeneous errors in the estimation of PRF values, thus the correct order is preserved. On the other hand, MC behaves drastically differently from other datasets (Figure 5(d)). MC can find the exact ranking with a reasonable number of samples, for ORDER-0 and ORDER-1, where the means of the tuples are well separated. However, when the means of tuples become closer, so do their PRF values, which makes it really hard for the randomized strategy MC to separate and rank them. We can see the convergence rate of MC on ORDER-5 is particular slow: with 100000 samples, the approximate ranking is not much better than a random permutation. We also tested several other weight functions and observed similar behavior. We omit those curves due to space constraints.

Summary: Overall, SPLINE typically converges faster than MC, but has a higher overhead compared to MC. DISC lies somewhere between SPLINE and MC. Therefore, if we face a situation where a high precision is needed to rank the tuples and MC needs too many iterations to achieve that precision (the number of iterations can be roughly estimated from Theorem 4), we can recommend using the Spline technique.

6.2 Execution Times for Exact Algorithms

Figure 6(a) shows the execution time of SPLINE, for the UNIFM-datasets, for different dataset sizes and variances. Recall in all UNIFM-datasets, the scores are in $[0, 10000]$. So generally speak-



(a) Exec. Times for Spline (solid) and PRF^l (dashed) (b) Exec. Times for PRF-l

Figure 6: Execution times for (a) SPLINE on UNIFM-datesets, PRF^ℓ on UNIFM-datasets and (b)PRF^ℓ on GAUSS-datasets.

ing, the higher the variance d is, the larger the overlap numbers are. The execution time is directly related to the overlap number, thus increases with d. We can also see the execution time does not scale linearly with the number of tuples. Again, the reason is that an increasing number of tuples results in larger overlap numbers. The execution time of SPLINE for piecewise polynomial was shown in the experiment of approximating PRF for GAUSS-datasets (the time for constructing splines for each Gaussian distribution is much smaller compared to computing PRF values for spline distribution). For PRF^ℓ function (see Appendix B for the details of the algorithm), the execution time is faster than the general SPLINE method. Figures 6(b) and (c) show the execution time of PRF^ℓ on UNIFM- and GAUSS- datasets. As we can see, the running time increases with d on UNIFM-datasets ($O(\sum_j m_j)$) and is independent of d on GAUSS-datasets ($O(n^2)$).

7. RELATED WORK

There is large body of literature on the area of ranking and top-k queries (see, for example, Ilyas et al.'s survey [14]). Recently, ranking over probabilistic databases has also drawn many researchers' attentions, and there have been several different proposals for ranking functions, including *U-Top* and *U-Rank* (Soliman et al. [23], Yi et al. [28]), probabilistic threshold top-k (PT-k) (Ming Hua et al. [13]), *Global-Topk* (Zhang and Chomicki [29]), and expected rank (Cormode et al. [6]). Ge et al. [10] study the score distributions and propose the notion of *typical top-k* queries. We refer the reader to our prior work [18] for a thorough discussion of the differences between these semantics, and how PRF is a unifying framework for reasoning about them.

However, the aforementioned work has focused mainly on tuple uncertainty and discrete attribute uncertainty. Soliman and Ilyas [24] were the first to consider the problem of handling continuous distributions in ranking probabilistic datasets. In particular, they consider uniformly distributed scores and their main algorithm is based on Monte Carlo integration to compute the positional probabilities. Therefore, their algorithm is randomized and only able to get an approximate answer. There is also much work on nearest neighbor-style queries over uncertain datasets [17, 4, 2, 5]. We discuss this work in more detail in Appendix A.

8. CONCLUSION AND FUTURE WORK

We studied the problem of ranking in presence of continuous attribute uncertainty, and developed several exact and approximate polynomial-time algorithms for efficiently ranking large probabilistic datasets. Our techniques significantly generalize and extend the prior work on ranking in presence of uncertainty, and enable us to efficiently handle continuous probability distributions, common in many applications that generate uncertain data. Our work has also opened up many new avenues for future work. In particular, we would like to explore the possibility of *progressive* approximation – in most cases, we don't need a large number of samples, or fine-granularity spline approximations, to rank the tuples, and it would be more efficient to spend the extra computational effort only where needed. Finally, incorporating correlations into our ranking framework is an important problem that we plan to study in future work.

Acknowledgements: This work was supported in part by NSF under Grants IIS-0546136 and IIS-0916736.

9. REFERENCES

[1] M. Abramowitz and I. Stegun, editors. *Handbook of Mathematical Functions*. Natl. Bureau of Standards, Appl. Math. Series - 55, 1972.
[2] G. Beskales, M. Soliman, and I. IIyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *VLDB*, 2008.
[3] C. Bohm, A. Pryakhin, and M. Schubert. The Gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *ICDE*, 2006.
[4] R. Cheng, J. Chen, M. Mokbel, and C. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, 2008.
[5] R. Cheng, L. Chen, J. Chen, X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *EDBT*, 2009.
[6] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, 2009.
[7] P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for Monte Carlo estimation. In *FOCS*, 1995.
[8] Carl de Boor. *A Practical Guide to Spline*. Springer, 2001.
[9] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.
[10] T. Ge, S. Zdonik, and S. Madden. Top-k queries on uncertain data: on score distribution and typical answers. In *SIGMOD*, 2009.
[11] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 2 edition, 1994.
[12] J. Guiver and E. Snelson. Learning to rank with softrank and gaussian processes. In *SIGIR*, 2008.
[13] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *SIGMOD*, 2008.
[14] I. Ilyas, G. Beskales, and M. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 2008.
[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4), 2002.
[16] M. Kendall. A new measure of rank correlation. *Biometrica*, 1938.
[17] H.P. Kriegel, P. Kunath, M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, 2007.
[18] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. In *VLDB*, 2009.
[19] J. Li and A. Deshpande. Consensus answers for queries over probabilistic databases. In *PODS*, 2009.
[20] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
[21] E. Parzen. On estimation of a probability density function and mode. *Ann. Math. Stat.*, 33:1065–1076, 1962.
[22] A. Ralston and P. Rabinowitz. *A First Course in Numerical Analysis*. 2001.
[23] M. Soliman, I. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.
[24] M. Soliman and I. Ilyas. Ranking with uncertain scores. In *ICDE*, pages 317–328, 2009.
[25] A. H. Stroud and D. Secrest. *Gaussian Quadrature Formulas*. Prentice-Hall Inc., 1966.
[26] M. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: optimizing non-smooth rank metrics. In *WSDM*, 2008.
[27] T. Tran, L. Peng, B. Li, Y. Diao, and A. Liu. PODS: a new model and processing algorithms for uncertain data streams. In *SIGMOD*, 2010.
[28] K. Yi, F. Li, D. Srivastava, and G. Kollios. Efficient processing of top-k queries in uncertain databases. In *ICDE*, 2008.
[29] X. Zhang and J. Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *DBRank*, 2008.
[30] O. Zuk, L. Ein-Dor, and E. Domany. Ranking under uncertainty. In *UAI*, pages 466–473, 2007.

APPENDIX

A. NEAREST NEIGHBOR QUERIES OVER UNCERTAIN OBJECTS

In this section, we briefly sketch how to apply our algorithms to *nearest neighbor* (*NN*) and k-*nearest neighbor* (k-*NN*) queries over uncertain objects. For generality, we only consider k-*NN* since *NN* is just special case of k-*NN* with $k = 1$. Suppose we are given a set of uncertain objects $\{t_i\}_{i=1}^n$ in d-dimensional Euclidean space $\mathbb{R}^d$. The position of each object t_i is captured by a *pdf* $p_i : \mathbb{R}^d \to \mathbb{R}^+$ and is independent of other objects. For a set of deterministic points, define $kNN(q)$ to be the set of k points that have the smallest Euclidean distances from q.

DEFINITION 3. *Given a query point $q \in \mathbb{R}^d$, a* k-NN *query retrieves k objects that have highest P_{knn} values where $P_{knn}(t_i, q) = \mathsf{Pr}(t_i \in kNN(q))$.*

In other words, we are looking for the objects that have the highest probability of being one of the k nearest neighbors of the query point. Kriegel et al. [17] and Cheng et al. [4] considered the threshold version of the query with $k = 1$, i.e., all objects with P_{1nn} values above a given threshold are returned. Beskales et al. [2] studied exactly the above query with $k = 1$. Cheng et al. [5] also considered kNN queries in a probabilistic setting. However, their semantics focus on the probability that a set of vertices is (as a whole) the set of k nearest neighbors (a semantics similar to *U-Top*-k [23]). This is not captured by the above definition (and cannot be captured using a PRF function either [18]).

In fact, it is not hard to see that the k-*NN* query can be directly translated into a PRF^ω query with the weight function:

$$\omega(i) = 1 \ \forall i \le k; \ \ \omega(i) = 0 \ \forall i > k$$

and the pdf of t_i's score being $\mu_i(x) = \mathsf{Pr}(\mathsf{dis}(t_i, q) = x)$. If p is a deterministic point, all μ_is are independent and we can apply our exact or approximate algorithms developed in Section 3 and 5 directly, depending on the type of the probability distributions μ_is.

EXAMPLE 2. *If the dimension $d = 1$ and each p_i is a uniform distribution over interval $[u_i, l_i]$, then μ_i is a piecewise constant function with at most 2 pieces. In fact, if $q \ge l_i$ or $q \le u_i$, μ_i is a uniform distribution over $[\min(u_i - q, l_i - q), \max(u_i - q, l_i - q)]$; if $u_i < q < l_i$, $\mu_i(x) = \frac{2}{l_i - u_i}$ for $x \in [0, \min(l_i - q, q - u_i)]$ and $= \frac{1}{l_i - u_i}$ for $x \in [\min(l_i - q, q - u_i), \max(l_i - q, q - u_i)]$. Therefore, we can apply the polynomial-time exact algorithm for piecewise polynomials developed in Section 3.*

Beskales et al. [2] also considered the case where the query point q itself can be uncertain. Although we can still translate it into a PRF query, our algorithms can not be directly applied since the probabilities $\mathsf{Pr}(\mathsf{dis}(t_{i,q}) = x)$ are correlated for different objects t_i. Theoretically, we can generalize the generating function technique we developed in Section 3 to handle this special correlation. However, this may introduce integration in higher dimensional space which can be tricky to implement. The practical implication of this approach is yet to be explored. Due to space constraints, we omit the details here. Finally, we would like to remark that it is possible to explore the spatial properties and design effective pruning rules to speed up the running time as the prior work has done. We leave it as an interesting future direction.

B. EXPECTED RANKS AND PRF$^\ell$

Recall that PRF^ℓ is a special case of the PRF function where the weight function is linear, i.e., $w_i = \omega(i) = n - i$. Aside from being a natural weight function, another key reason to study PRF^ℓ is its close relationship to *expected ranks*.

Expected rank of a tuple t is defined to be:

$$\mathsf{E}[r_{pw}(t)] = \int_{pw \in PW} p(pw) r_{pw}(t)$$

where $r_{pw}(t) = |pw|$ if $t_i \notin pw$. Let C denote the expected size of a possible world. It is easy to see that: $C = \sum_{i=1}^n p_i$ due to linearity of expectation. Then the expected rank of t can be seen to consist of two parts:

(1) $\sum_{i>0} i \times \mathsf{Pr}(r(t) = i)$, which counts the contribution of the possible worlds where t exists, and

(2) $\int_{pw: t \notin pw} \mathsf{Pr}(pw)|pw| = (C - p(t))(1 - p(t))$ which counts the contribution of the worlds where t does not exist[2]. Thus:

$$\begin{aligned}\mathsf{E}[r_{pw}(t)] &= \textstyle\sum_{i>0} i \times \mathsf{Pr}(r(t) = i) + (1 - p(t))(C - p(t)) \\ &= -PRF^\ell(t) + C + (n - C - 1)p(t) + (p(t))^2.\end{aligned}$$

Next, we present algorithms for computing $\sum_{i>0} i \times \mathsf{Pr}(r(t) = i)$, and hence for ranking according to PRF^ℓ or expected ranks.

Since tuple uncertainty is also considered, we let $\bar{\rho}_i(\ell) = \mathsf{Pr}(s_i \ge \ell) = p_i \int_\ell^\infty \mu_i(x)\mathrm{d}x$. We can then see:

$$\begin{aligned}\sum_{i>0} i \, \mathsf{Pr}(r(t) = i) &= p_i \int_{-\infty}^{+\infty} \mathsf{E}\Big[\sum_{j \ne i} \delta(s_j > \ell) \mid s_i = l\Big] \mu_i(\ell)\mathrm{d}\ell \\ &= p_i \sum_{j \ne i} \int_{-\infty}^{+\infty} \mathsf{E}\big[\delta(s_j > \ell)\big] \mu_i(\ell)\mathrm{d}\ell \\ &= p_i \sum_{j \ne i} \int_{-\infty}^{+\infty} \bar{\rho}_j(\ell)\mu_i(\ell)\mathrm{d}\ell \\ &= p_i \sum_{j \ne i} p_j \int_{-\infty}^{+\infty} \int_l^{+\infty} \mu_j(x)\mu_i(\ell)\mathrm{d}x\mathrm{d}\ell\end{aligned}$$

Let $\mathcal{A}$ be a class of functions. Suppose each $\mu_i(\ell)$ is a piecewise function such that each piece can be expressed by a function in $\mathcal{A}$. Similar to Section 3.2, we partition the real line into a set $\mathcal{I}$ of small intervals such that in each small interval, every $\mu_i(\ell)$ can be expressed by a single formula.

In general, given i, j and small interval I, if we can obtain the numerical value of:

$$\int_{l_I}^{u_I} \int_\ell^\infty \mu_j(x)\mu_i(\ell)\mathrm{d}x\mathrm{d}\ell$$

in $O(\gamma)$ time, then we can compute $\sum_{j>0} i \ \mathsf{Pr}(r(t_i) = j)$ in $O(\gamma n |\mathcal{I}_i|)$ time. The expected ranks and the PRF^ℓ values for all tuples can then be computed in $O(\gamma n \sum_i |\mathcal{I}_i|) = O(\gamma n \sum_j m_j)$ time. Next we look at different classes of functions $\mathcal{A}$ in turn.

Gaussian: Suppose s_i is a normally distributed with mean λ_i and variance σ^2, denoted $s_i \sim \mathcal{N}(\lambda_i, \sigma_i^2)$. Since Gaussians are defined on the entire real line, we have a single partition $[-\infty, +\infty]$. By the above discussion, the problem reduces to computing $\int_{-\infty}^{+\infty} \int_\ell^\infty \mu_j(x)\mu_i(\ell)\mathrm{d}x\mathrm{d}\ell$ for any i, j. The key observation here is that the above formula is exactly $\mathsf{Pr}(s_j \ge s_i)$. Also, it is well known that $s_j - s_i \sim \mathcal{N}(\lambda_j - \lambda_i, \sigma_j^2 + \sigma_i^2)$. Therefore,

$$\mathsf{Pr}(s_j \ge s_i) = 1 - \Phi\Big(\tfrac{\lambda_i - \lambda_j}{\sqrt{\sigma_j^2 + \sigma_i^2}}\Big) = \Phi\Big(\tfrac{\lambda_j - \lambda_i}{\sqrt{\sigma_j^2 + \sigma_i^2}}\Big)$$

where $\Phi(x)$ is the cdf of the standard normal distribution $\mathcal{N}(0, 1)$, i.e., $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{x^2}{2}} \mathrm{d}x$. Indeed, the first equality is due to the fact that the cdf of $\mathcal{N}(\lambda, \sigma^2)$ is $\Phi(\frac{x - \lambda}{\sigma})$ and the second holds since $\Phi(x) = 1 - \Phi(-x)$.

$\Phi(x)$ has been widely used in scientific and statistical computing and its numerical value with high precision can be computed extremely efficiently [1]. Therefore, it is very reasonable to assume that it can be computed in $O(1)$ time even though it does

[2]This relationship does not hold in presence of correlations.

not have a closed form expression. The overall running time for computing PRF^ℓ values of all tuples is then $O(n^2)$.

A similar relationship between expected ranks and $\mathsf{Pr}(s_j \geq s_i)$ was also observed by Cormode et al. [6], who use it to to obtain algorithms for discrete distributions.

Convex combination of Gaussians: Convex combinations of Gaussians have been widely used in curve fitting, kernel density estimation and other application domains [21].

Let $\mu_i(x) = \sum_{j=1}^{l_i} \gamma_{ij} f_{ij}(x)$ where $\sum_{j=1}^{l_i} \gamma_{ij} = 1, \gamma_{i,j} > 0 \forall i, j$ and $f_{ij}(x)$ is the pdf for $\mathcal{N}(\lambda_{ij}, \sigma_{ij}^2)$. Let s_{ij} be a real random value that follows the distribution f_{ij}. Again, we need to compute $\mathsf{Pr}(s_j \geq s_i)$. It is convenient to think that s_i is generated by the following two-stage process: we first randomly choose a integer j between 1 and l_i according to distribution γ, i.e., t is chosen with probability γ_{it} $\forall 1 \leq t \leq l_i$. Then, s_i is generated according to distribution f_{ij}. Now, we can easily see

$$\begin{aligned}\mathsf{Pr}(s_j \geq s_i) &= \sum_{j'=1}^{l_j} \sum_{j=1}^{l_i} \gamma_{jj'} \gamma_{ij} \mathsf{Pr}(s_{jj'} \geq s_{ij}) \\ &= \sum_{j'=1}^{l_j} \sum_{j=1}^{l_i} \gamma_{jj'} \gamma_{ij} \Phi\Big(\frac{\lambda_{jj'} - \lambda_{ij}}{\sqrt{\sigma_{jj'}^2 + \sigma_{ij}^2}}\Big)\end{aligned}$$

For each tuple pair (j, i), we need $O(l_j l_i)$ time, thus, the total running time is $O(\sum_{1 \leq i < j \leq n} l_i l_j)$.

Exponential: Suppose s_i follows the exponential distribution with rate parameter λ_i, i.e., $\mu_i(x) = \begin{cases} \lambda_i e^{-\lambda_i x}, & x \geq 0, \\ 0, & x < 0. \end{cases}$

We only need to consider the positive axis. It is easy to see that:

$$\begin{aligned}\int_0^\infty \int_\ell^\infty \mu_j(x)\mu_i(\ell)\mathrm{d}x\mathrm{d}\ell &= \lambda_i\lambda_j \int_0^{+\infty} \int_\ell^\infty e^{-\lambda_j x} e^{-\lambda_i \ell}\mathrm{d}x\mathrm{d}\ell \\ &= \lambda_i \int_0^{+\infty} e^{-(\lambda_i+\lambda_j)\ell}\mathrm{d}\ell = \frac{\lambda_i}{\lambda_i + \lambda_j}.\end{aligned}$$

Hence, the PRF^ℓ values can be computed in $O(n^2)$ time.

Piecewise polynomial of order γ: Directly applying the above framework gives an $O(\gamma^2 n \sum_j m_j)$ time algorithm. We can improve the running time to $O(\gamma^2 \sum_j m_j)$ as follows. For each small interval I_j, we first compute the expansion of the polynomial $\sum_j \bar{\rho}_j(\ell)$ which can be done in $O(\gamma m_j)$ time (m_j additions of polynomials of degree γ). Subsequently, for each i such that $I_j \in \mathcal{I}_i$, the expansion of $\sum_{j \neq i} \bar{\rho}_j(\ell)\mu_i(\ell)$ can be obtained in $O(\gamma^2)$ time (subtract $\bar{\rho}_j(\ell)$ from $\sum_j \bar{\rho}_j(\ell)$ and then multiply with $\mu_i(\ell)$) [3] and computing the numerical value of:

$$A_{I_j,i} = \int_{I_j} \sum_{j \neq i} \bar{\rho}_j(\ell)\mu_i(\ell)\mathrm{d}\ell$$

takes an additional $O(\gamma)$ time (integrating each term of the polynomial takes $O(1)$ time). Therefore, the overall running time is $O(\gamma \sum_j m_j + \sum_j m_j(\gamma^2 + \gamma)) = O(\gamma^2 \sum_j m_j)$.

Uniform: This is a special case of the previous one with $\gamma = 1$. Thus, the running time is $O(\sum_j m_j)$.

To summarize, the expected ranks and the PRF^ℓ values for all tuples can be computed very efficiently (in $O(n^2)$ time) for many continuous probability distributions. This significantly generalizes the results on these two functions in the prior work.

[3] Actually, this can be done in $O(\gamma \log \gamma)$ time by using FFT. However, since γ is usually very small, we can just do the polynomial multiplication in the straightforward manner in $O(\gamma^2)$ time.

C. APPROXIMATE PRF^E COMPUTATION

In this section, we present an approximation technique for approximating the PRF^e function using Legendre-Gauss Quadrature method, and demonstrate its effectiveness through an empirical evaluation. For completeness, we begin with a brief overview of Legendre-Gauss Quadrature.

C.1 Legendre-Gauss Quadrature

Suppose we want to approximate $\int_a^b f(x)\mathrm{d}x$ by a linear sum $\sum_{i=1}^k c_i f(x_i)$ for a fixed integer k where c_i and x_i are to be determined but independent of the function f. Actually, if we let $c_1 = \ldots = c_k = c = 1/(k-1)$, $x_i = a + ci$ and k approach to infinity, the linear sum is exactly the Riemann sum which should be equal to the value of the integral. However, in practice, we are only allowed to evaluate the function at a finite number of points which results in an approximation of the integral. Assume that $a = -1$ and $b = 1$. The Legendre-Gauss quadrature (LGQ) of degree k evaluates the function at x_i for $1 \leq i \leq k$ where x_is are the k roots of *Legendre polynomial* of degree k and c_i can be computed by $c_i = \int_{-1}^1 \prod_{j \neq i} \left(\frac{x - x_j}{x_i - x_j}\right) \mathrm{d}x$. For example, the roots of Legendre polynomial of degree 3 are $x_1 = -\sqrt{3/5}, x_2 = 0, x_3 = \sqrt{3/5}$. Then we can get $c_1 = 5/9, c_2 = 8/9$ and $c_3 = 5/9$. Therefore, our approximation is simply:

$$\int_{-1}^1 f(x)\mathrm{d}x = \tfrac{5}{9} \cdot f(-\sqrt{\tfrac{3}{5}}) + \tfrac{8}{9} \cdot f(0) + \tfrac{5}{9} \cdot f(\sqrt{\tfrac{3}{5}}) + error$$

Computing the roots $\{x_i\}_{i=1,\ldots,k}$ for general k is computationally nontrivial. However, due to the practical importance of the method, the values of x_i and c_i have already been tabulated for every k up to a few hundreds [25] and we can use these values directly.

If the integral is not $[-1, +1]$, we can use the following simple transform:

$$\int_a^b f(x)\mathrm{d}x = \tfrac{b-a}{2} \int_{-1}^1 f\big(a + \tfrac{b-a}{2}(y+1)\big)\mathrm{d}y$$

and then approximating $\int_{-1}^1 g(y)\mathrm{d}y$ where $g(y) = f(a + \frac{b-a}{2}(y+1))$. Sometimes, if the length of $[a, b]$ is very large, it is better to partition $[a, b]$ into small intervals, approximate the integral over each small interval such that we do not need to evaluate the function at many points in each small interval, thus can still use the existing x_i and c_i values from the tablet. It is called *composite rule*.

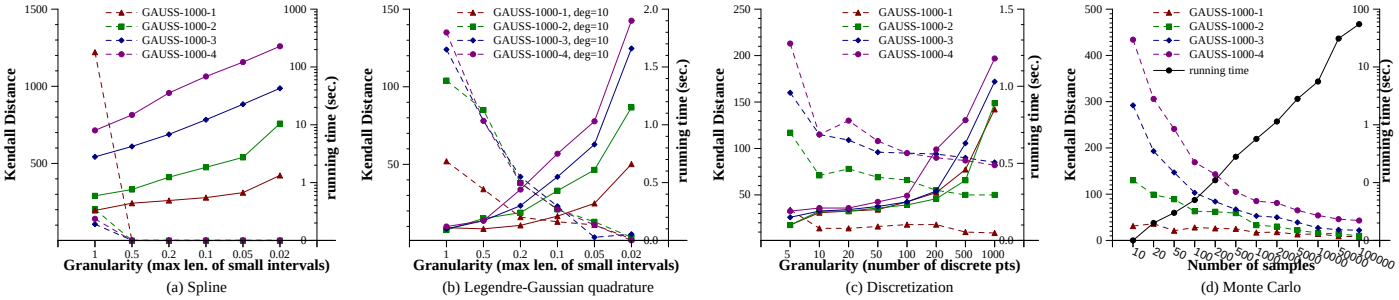
Theoretically, assuming continuity of the $2k$th derivative of $f(x)$, if we partition $[a, b]$ into N small intervals and apply LGQ of degree k on each small interval, the approximation error is

$$Error = \frac{(b-a)^{2k+1}}{N^{2k}} \frac{(k!)^4}{(2k+1)[(2k)!]^3} f^{(2k)}(\xi)$$

where ξ is some points in (a, b) [22, pp.116]. Let $\Delta = \frac{b-a}{N}$. If we treat k, $f(x)$ as fixed, the behavior of the error (in terms of Δ) is $Error(\Delta) = O(\Delta^{2k})$. Although it seems that the error decays exponentially with k (assuming N fixed) and polynomially with N (assuming k fixed), in practice, people usually use Legendre-Gauss quadrature with a bounded degree (typically $k < 20$). This is because (1) it is good enough for most applications, (2) the roots for high order Legendre polynomial are nontrivial to compute and (3) it is hard to analyze and control the behavior of the higher order derivative of $f(x)$, thus the error.

C.2 Approximating $PRF^e(\alpha)$ by Legendre-Gauss Quadrature for Real α

As we discussed in Section 4, the $PRF^e(\alpha)$ value of tuple t_i has a closed form expression, which is the value of the generating function (3) evaluated at α, i.e., $F_i(\alpha)$. For arbitrary distributions, we



(a) Spline (b) Legendre-Gaussian quadrature (c) Discretization (d) Monte Carlo

Figure 7: The comparison of various methods for computing $PRF^e(\alpha = 0.99)$. Solid lines indicate the running times (with axes drawn on the right hand side), whereas dashed lines indicate the Kendall distance (an error measure).

can of course use the approximation technique developed for general PRF functions. However, we observe that $F_i(\alpha)$ is simply the integral of the function $f(\ell) = \prod_{j \neq i}(\rho_j(\ell) + \bar{\rho}_j(\ell)\alpha)\mu(\ell)$; we can evaluate the function $f(\ell)$ itself at any point ℓ in polynomial time[4]. Given this, we can use the Legendre-Gauss quadrature (LGQ) method directly to achieve a much more efficient approximation of $PRF^e(\alpha)$ for real α. We also plot the asymptotic Error-N (precision-complexity) trade-offs for LGQ of degree 2 and 10 (See Figure 4). In our experimental study, we use LGQ of degree 10.

C.3 Empirical Comparison

We compared the four techniques for PRF^e computation: (1) Spline (SPLINE), (2) Legendre-Gauss Quadrature (LGQ), (3) Monte Carlo (MC), and (4) Discretization (DISC). We use the same datasets as described in Section 6. The key parameter for LGQ is the granularity of intervals. For SPLINE and DISC, there are faster implementations, which we will call SPLINE-E and DISC-E respectively, for computing PRF^e.

Figure 7(a),(b),(c) and (d) show the execution times and convergence rates for SPLINE-E, LGQ, DISC-E, and MC, respectively. The "true" ranking is computed by using SPLINE-E with a granularity of 0.005. We can see that SPLINE-E converges very fast, just like the general SPLINE algorithm, but the running time is faster. In Figure 7(b), we can see LGQ (with degree 10) also converges very fast: Exact ranking can be obtained when the granularity is less then 0.05, which is a bit slower than SPLINE-E, but the execution time is much lower. For example, LGQ takes less than 2 seconds to get an exact answer on GAUSS-1000-4 while SPLINE needs more than 10 seconds. Actually, a significant portion of the execution time is for the construction of small intervals, so using a higher degree quadrature does not incur a significant increase in the running time. In Figure 7(c), we observe that the convergence of MC is quite similar to the previous case and the running time is almost the same since MC does not utilize any special property of PRF^e to speed up the execution. For DISC-E the convergence rate is also similar to the general DISC algorithm while the running time is much faster. We also did the experiments on ORDER datasets. The convergence rates for SPLINE-E, MC and DISC-E are quite similar to their counterparts for the general PRF computation: SPLINE-E and DISC-E continue to find exact ranking in all granularities we tested while MC converges rather slowly on ORDER-4 and -5. For LGQ, a granularity of 1 is able to find the exact ranking for all ORDER-datasets and execution time is always less then 1 second. Due to space constraints, we omit those curves. In both sets of the experiments, the quadrature method typically converges faster than MC and DISC-E and has a lower overhead than SPLINE-E.

[4]We assume $\rho_i(x)$ and $\bar{\rho}_i(x)$ can be computed easily.

D. THE PROOF OF THEOREM 1

First, we note that $F_i(x)$ defined in (3) is a polynomial of x. This is because each term in the expansion of the product inside the integral is of the form $f(\ell)x^k$ for some integer k and function $f()$, and taking integral on ℓ eliminates the variable ℓ but has no effect on x.

Let $\delta(p) = \begin{cases} 1, & \text{if } p = true \\ 0, & \text{if } p = false \end{cases}$ be the indicator function for the event p. It is straightforward to see that t_i is ranked at position j in a possible world *iff* there are exactly $(j-1)$ tuples with higher score present in that world. Given this, we get:

$$\begin{aligned}\mathsf{Pr}(r(t_i) = j) &= \mathsf{Pr}\Big(\sum_{j \neq i} \delta(s_j > s_i) = j - 1\Big) \\ &= \int_{-\infty}^{\infty} \mathsf{Pr}\Big(\sum_{j \neq i} \delta(s_j > \ell) = j - 1\Big)\mu_i(\ell)\mathrm{d}\ell.\end{aligned}$$

Let us consider how to compute $\mathsf{Pr}\Big(\sum_{j \neq i} \delta(s_j > \ell) = j\Big)$ for any fixed ℓ, i.e., the probability of the random event that there are exactly j tuples other than t_i that have score larger than ℓ. The key observation here is that computing the probability is equivalent to the following problem: Given a set of tuples $t_j, j = 1, \ldots, n, j \neq i$, with tuple t_j having existence probability $\bar{\rho}_j(\ell) = \mathsf{Pr}(s_j > \ell)$, compute $\mathsf{Pr}(j \text{ tuples exist})$. Consider the following generating function:

$$\mathcal{F}_i(x, \ell) = \prod_{j \neq i}\Big(\rho_j(\ell) + \bar{\rho}_j(\ell)x\Big).$$

If we treat ℓ as a fixed value and $\mathcal{F}_i(x, \ell)$ as a polynomial of x, the coefficient of the term x^j is $\mathsf{Pr}\Big(\sum_{j \neq i} \delta(s_j > \ell) = j\Big)$ (see [19, 18]). Thus, we can write:

$$\mathcal{F}_i(x, \ell) = \sum_{j \geq 0} \mathsf{Pr}\Big(\sum_{j \neq i} \delta(s_j > \ell) = j\Big)x^j.$$

Multiplying by $x\mu_i(\ell)$ and taking integrals on both sides, we get:

$$\begin{aligned}F_i(x) &= x\int_{-\infty}^{\infty} \mathcal{F}_i(x, \ell)\mu_i(\ell)d\ell \\ &= x\int_{-\infty}^{\infty} \sum_{j \geq 0} \mathsf{Pr}\Big(\sum_{j \neq i} \delta(s_j > \ell) = j\Big)\mu_i(\ell)x^j\mathrm{d}\ell \\ &= \sum_{j \geq 1} \int_{-\infty}^{\infty} \mathsf{Pr}\Big(\sum_{j \neq i} \delta(s_j > \ell) = j - 1\Big)\mu_i(\ell)\mathrm{d}\ell x^j \\ &= \sum_{j \geq 1} \mathsf{Pr}(r(t_i) = j)x^j\end{aligned}$$

Therefore, $F_i(x)$ is the generating fn. for $\{\mathsf{Pr}(r(t_i) = j)\}_{j \geq 0}$.

E. THE PROOF OF THEOREM 2

We need a few lemmas before establishing the theorem.

LEMMA 1. *$c_1, \ldots, c_n$ and $e_1, \ldots, e_n$ are complex numbers such that $|c_i| \leq 1$ and $|e_i| \leq n^{-\beta}$ for all i and some $\beta > 1$.*

$$\Big|\prod_{i=1}^{n}(c_i + e_i) - \prod_{i=1}^{n} c_i\Big| \leq O(n^{1-\beta})$$

PROOF.

$$\begin{aligned}\Big|\prod_{i=1}^{n}(c_i + e_i) - \prod_{i=1}^{n} c_i\Big| &= \Big|\sum_{S\subseteq[n], S\neq\emptyset}\prod_{i\in S} c_i \prod_{i\in[n]\setminus S} e_i\Big| \\ &\leq \Big|\sum_{k=1}^{n}\sum_{S\subseteq[n],|S|=k}\prod_{i\in S} c_i \prod_{i\in[n]\setminus S} e_i\Big| \\ &\leq \sum_{k=1}^{n}\binom{n}{k} n^{-k\beta} \leq \sum_{k=1}^{n}\frac{n^{k(1-\beta)}}{k!} \\ &\leq e^{n^{1-\beta}} - 1 = O(n^{1-\beta})\end{aligned}$$

The third inequality holds because $\binom{n}{k} \leq \frac{n^k}{k!}$. The last inequality holds since $e^z = \sum_{i>0}\frac{z^i}{i!}$ and the last equality is due to the fact that $e^{O(f(n))} = 1 + O(f(n))$ if $f(n) = O(1)$ (e.g. [11, p.452]). □

LEMMA 2. *Let μ be a probability density function with $|\mathsf{supp}(\mu)| = O(1)$. $\widehat{\mu}$ is another function such that $\mathsf{supp}(\widehat{\mu}) = \mathsf{supp}(\mu)$ and $|\widehat{\mu}(x) - \mu(x)| \leq \epsilon_1 < 1$. Let $f, \widehat{f} : \mathbb{R} \to \mathbb{C}$ be two functions such that $|f(x)| \leq 1$ and $|f(x) - \widehat{f}(x)| \leq \epsilon_2 < 1$ for all x. Then,*

$$\Big|\int_{-\infty}^{\infty}\mu(x)f(x)\mathrm{d}x - \int_{-\infty}^{\infty}\widehat{\mu}(x)\widehat{f}(x)\mathrm{d}x\Big| \leq O(\epsilon_1 + \epsilon_2)$$

PROOF.

$$\begin{aligned}\text{LHS} =& \Big|\int_{\mathsf{supp}(\mu)}\big(\mu(x)f(x) - \widehat{\mu}(x)\widehat{f}(x)\big)\mathrm{d}x\Big| \\ \leq& \Big|\int_{\mathsf{supp}(\mu)}\big(\mu(x)f(x) - \mu(x)\widehat{f}(x)\big)\mathrm{d}x\Big| + \Big|\int_{\mathsf{supp}(\mu)}\epsilon_1\widehat{f}(x)\mathrm{d}x\Big| \\ \leq& \int_{\mathsf{supp}(\mu)}\mu(x)\big|f(x) - \widehat{f}(x)\big|\mathrm{d}x + \Big|\int_{\mathsf{supp}(\mu)}\epsilon_1\widehat{f}(x)\mathrm{d}x\Big| \\ \leq& \int_{\mathsf{supp}(\mu)}\mu(x)\epsilon_2\mathrm{d}x + \epsilon_1|\mathsf{supp}(\mu)| = O(\epsilon_1 + \epsilon_2)\end{aligned}$$

The first inequality holds since $|a + b| \leq |a| + |b|$ for any complex numbers a, b. □

LEMMA 3. *Suppose $\omega(i) \leq 1$ for all $0 \leq i \leq n-1$. Let $\psi(0), s, \psi(n-1)$ denote the discrete Fourier transform of $\omega(0), s, \omega(n-1)$. Then $\sum_{i=0}^{n-1}|\psi(i)| \leq n^{3/2}$.*

PROOF.

$$\Big(\sum_{i=0}^{n-1}|\psi(i)|\Big)^2 \leq \sum_{i=0}^{n-1}1\sum_{i=0}^{n-1}|\psi(i)|^2 = n^2\sum_{i=1}^{n-1}\omega(i)^2 \leq n^3$$

The first inequality is the the Cauchy-Schwarz inequality which states $|\langle x, y\rangle|^2 \leq \langle x, x\rangle\langle y, y\rangle$ for any vectors x and y where $\langle,\rangle$ is the inner product. The second equality follows from Parseval's equality $\sum_{i=0}^{n}|\omega(i)|^2 = \frac{1}{n}\sum_{i=0}^{n}|\psi(i)|^2$. □

Now we can prove our main theorem.

PROOF OF THEOREM 2: Let $\widehat{\mu}_i$ be the approximated distribution of s_i for each i. Let $\bar{\rho}_i(\ell) = \mathsf{Pr}(s_i > \ell) = \int_\ell^\infty \mu_i(x)\mathrm{d}x$, $\rho_i(\ell) = 1 - \bar{\rho}_i(\ell)$, $\widehat{\rho}_i(\ell) = \int_\ell^\infty \widehat{\mu}_i(x)\mathrm{d}x$ and $\widehat{\varrho}_i(\ell) = 1 - \widehat{\rho}_i(\ell)$. It is known that (e.g. [8, p.40]), for each small interval I,

$$|\mu_i(x) - \widehat{\mu}_i(x)| \leq \left(\frac{|I|}{2}\right)^4 \frac{\max_{y\in I}|\mu^{(4)}(y)|}{4!}.$$

Since $|\mathsf{supp}(\mu)| = O(1)$ and $\max_{y\in\mathsf{supp}(\mu)}\mu^{(4)}(y) = O(1)$, we can see $|\mu_i(x) - \widehat{\mu}_i(x)| = O(n^{-4\beta})$. From Lemma 2, it follows that $|\bar{\rho}_i(\ell) - \widehat{\rho}_i(\ell)| \leq O(|I|^4) = O(n^{-4\beta})$ for all ℓ.

For ease of description, we assume that the rank start from 0. Let us focus on the estimation of $\Upsilon_\omega(t)$ for a particular tuple t. Let $\psi(0), s, \psi(n-1)$ denote the discrete Fourier transform of $\omega(t, 0), s, \omega(t, n-1)$. Hence, we have

$$\omega(t, i) = \frac{1}{n}\sum_{k=0}^{n-1}\psi(k)e^{\frac{2\pi\jmath}{n}ki} \qquad i = 0, \ldots, n-1.$$

where $\jmath$ is the imaginary unit. Denote the PRF^e value of t with parameter $e^{\frac{2\pi\jmath}{n}k}$ by $\Upsilon_k(t)$. Therefore, we have

$$\Upsilon_\omega(t) = \frac{1}{n}\sum_{k=0}^{n-1}\psi(k)\Upsilon_k(t). \tag{7}$$

Now, we analyze the approximation error for the approximated PRF^e value with any parameter α such that $|\alpha| = 1$. Since the PRF^e value with parameter α equals the value of the generating function evaluated at α, it suffices to bound $|F(\alpha) - \widehat{F}(\alpha)|$ where F is the generating function for t (see Eq. 3) and $\widehat{F}$ is its approximation (replace $\bar{\rho}_i$s and ρ_is with $\widehat{\rho}_i$s and $\widehat{\varrho}_i$s respectively).

We observe that, for any $\alpha \in \mathbb{C}$ with $\alpha = 1$ and any $\ell \in \mathbb{R}$,

$$|\rho_j(\ell) + \bar{\rho}_j(\ell)\alpha| \leq |\rho_j| + |\bar{\rho}_j(\ell)\alpha| = 1.$$

Hence, we have that

$$\begin{aligned}&|\widehat{\varrho}_i(\ell) + \widehat{\rho}_i(\ell)\alpha - (\rho_i(\ell) + \bar{\rho}_i(\ell)\alpha)| \\ \leq& |\widehat{\varrho}_i(\ell) - (\rho_i(\ell)| + \alpha|\widehat{\rho}_i(\ell) - \bar{\rho}_i(\ell)| \leq O(n^{-4\beta})\end{aligned}$$

Therefore, by Lemma 1, we have

$$\Big|\prod_{j\neq i}\Big(\widehat{\varrho}_j(\ell) + \widehat{\rho}_j(\ell)\alpha\Big) - \prod_{j\neq i}\Big(\rho_j(\ell) + \bar{\rho}_j(\ell)\alpha\Big)\Big| \leq O(n^{1-4\beta})$$

Recall that $F_i(x) = x\int_{-\infty}^{\infty}\mu_i(\ell)\prod_{j\neq i}\Big(\rho_j(\ell) + \bar{\rho}_j(\ell)x\Big)d\ell$. Applying Lemma 2, we can get

$$|\widehat{F}_i(\alpha) - F_i(\alpha)| \leq O(n^{1-4\beta} + n^{-4\beta}) = O(n^{1-4\beta})$$

Hence, from (7) and Lemma 3, we obtain that

$$\begin{aligned}|\widehat{\Upsilon}_\omega(t) - \Upsilon_\omega(t)| &= \Big|\frac{1}{n}\sum_{k=0}^{n-1}\psi(k)\big(\widehat{\Upsilon}_k(t) - \Upsilon_k(t)\big)]\Big| \\ &= \Big|\frac{1}{n}\sum_{k=0}^{n-1}\psi(k)\big(\widehat{F}_k(e^{\frac{2\pi\jmath}{n}k}) - F_k(e^{\frac{2\pi\jmath}{n}k})\big)\Big| \\ &\leq \frac{1}{n}O(n^{1-4\beta})\Big|\sum_{k=0}^{n-1}\psi(k)\Big| = O(n^{3/2-4\beta})\end{aligned}$$

□