Abstract

Title of dissertation: Decision Making Under Uncertainty

Jian Li, Doctor of Philosophy, 2011

Directed by: Professor Amol Deshpande
Department of Computer Science

Almost all important decision problems are inevitably subject to some level of uncertainty either about data measurements, the parameters, or predictions describing future evolution. The significance of handling uncertainty is further amplified by the large volume of uncertain data generated by modern data gathering or integration systems. Various types of problems of decision making under uncertainty have been subject to extensive research in computer science, economics and social science. In this dissertation, I study three major problems in this context, *ranking*, *utility maximization*, and *matching*, all involving uncertain datasets.

First, we consider the problem of ranking and top-k query processing over probabilistic datasets. By illustrating the diverse and conflicting behaviors of the prior proposals, we contend that a single, specific ranking function may not suffice for probabilistic datasets. Instead we propose the notion of *parameterized ranking functions*, that generalize or can approximate many of the previously proposed ranking functions. We present novel exact or approximate algorithms for efficiently ranking large datasets according to these ranking functions, even if the datasets exhibit complex correlations or the probability distributions are continuous.

The second problem concerns with the stochastic versions of a broad class of combinatorial optimization problems. We observe that the expected value is inadequate in capturing different types of *risk-averse* or *risk-prone* behaviors, and instead we consider a more general objective which is to maximize the *expected utility* of the solution for some given utility function. We present a polynomial time approximation algorithm with *additive error* ϵ for any $\epsilon > 0$, under certain conditions. Our result generalizes and improves several prior results on stochastic shortest path, stochastic spanning tree, and stochastic knapsack.

The third is the stochastic matching problem which finds interesting applications in online dating, kidney exchange and online ad assignment. In this problem, the existence of each edge is uncertain and can be only found out by probing the edge. The goal is to design a probing strategy to maximize the expected weight of the matching. We give linear programming based constant-factor approximation algorithms for weighted stochastic matching, which answer an open question raised in prior work.

Decision Making Under Uncertainty

by

Jian Li

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:

Professor Amol Deshpande, Chair
Professor Samir Khuller
Professor David Mount
Professor MohammadTaghi Hajiaghayi
Professor Hal Daumé III
Professor Gang Qu, Dean's representative

Dedicated to my beloved parents
for always supporting me

Acknowledgments

First, I would like to express my deepest gratitude to my advisors, Professor Amol Deshpande and Professor Samir Khuller, for their continuous guidance and support throughout my entire graduate life at Maryland. It has been a tremendous pleasure to work with them. I have learnt a lot from them, not only knowledge on cutting-edge research topics, but also effective presentation skills, and the customs of the academic life. I am proud to be their academic descendant and hope that one day I can become as wonderful advisors as they are. I would also like to thank other members on my thesis committee board, Professors David Mount, MohammadTaghi Hajiaghayi, Hal Daumé III and Gang Qu, for their willingness to participate in my defense and their helpful comments.

The materials of the thesis come from several joint projects and I would like to take this opportunity to thank all of my collaborators: Barna Saha, Julian Mestre, Bhargav Kanagal, Qin Zhang, Ke Yi, Arvind Arasu, Raghav Kaushik, Danny Z. Chen, Rudolf Fleischer, Mordecai Golin, Nikhil Bansal, Anupam Gupta, Viswanath Nagarajan, Atri Rudra, Chandra Chekuri, Sungjin Im, Benjamin Moseley and many others. Their insightful ideas and useful suggestions helped me to develop my research interest and inspired me in discovering many algorithms presented in the thesis. My special thanks goes to Julian, who hosted me during my visit to MPI in the summer of 2009, and Arvind, who was my mentor during my internship at Microsoft in the summer of 2010. It was a very enjoyable and beneficial experience for me to work with them. I am also indebted to Professor Aravind Srinivasan for his excellent lectures on randomized algorithms and many very helpful discussions on various parts of the thesis.

My study aboard experience in US would not be as enjoyable without my dear friends. I would like to thank my roommates, Liang, Dan, Chen, Shule, Min, He, Shengya, Xinlin, Changyi, Guohong, Ying, Hao and Cong (I have been moving a lot). They are all extremely considerate and easy going and have made my

daily life so easy and pleasant. I would also like to thank my Starcraft-mates at Microsoft: Yaodong, Rui, Hua; my table tennis-mates: Xiaoming, Tom, Hao, Xi, Ke; and especially my soccer and poker-mates: Hao, Cong, Changyi, Zhuliang, Huashuai, Yujie, Kejia. Without them, my leisure time would not be so rich and colorful.

Finally, I want to thank my parents, whose support and love have made this all possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent years have witnessed a dramatic increase in the number of application domains that naturally generate uncertain data and that demand support for executing complex decision-support queries and solving large scale optimization problems over them. Uncertainty can arise due to a variety of reasons, such as noisy measurements, missing or conflicting data or predictions of the future. We list some application domains to exemplify where the uncertainty comes from, what types of uncertainty we may encounter, and how we may represent such uncertain data.

1. **Data integration and cleaning**: Data integration involves combining several databases residing in different sources into a single unified database [63]. The integrated table is typically uncertain if the sources are not consistent. See Figure 1.1 for an example. In the integrated table on the right hand side, the existence of each tuple is uncertain (this is called *tuple uncertainty*). Data cleaning is the process of detecting and correcting corrupt or inaccurate records from a database [11]. Uncertainty may arise from data entry errors or differences in data representation. For example, some records in the table may use abbreviated conference names (e.g., FOCS) while the others may use fully expanded names (e.g., Symposium on Foundation of Computer Science). The data cleaning task should be able to identify such matches and convert the tuples to a uniform format. However, the results of this process could be erroneous and uncertain (e.g., FOCS also stands for Femme Of Color Symposium).

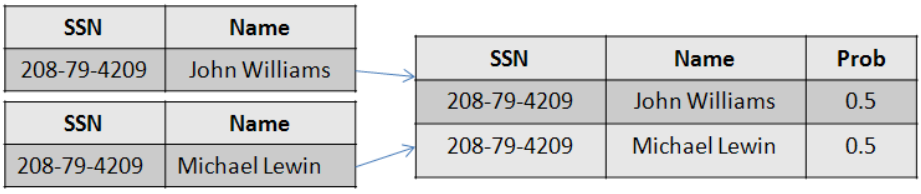


Figure 1.1: A data integration example. The two tuples shown on the left hand side, each coming from a distinct data source, have the same key (i.e., SSN), but different attribute values. In the integrated table, we keep both tuples and associate probability 0.5 to each of them. We also make them mutually exclusive, i.e., at most one tuple is present in any possible realization (possible world).

2. **Information extraction**: The goal of information extraction is to automatically extract structured information (e.g., database tables) from unstructured and/or semi-structured information (e.g., HTML, XML files in the Internet) [71]. The tables are typically constructed by crawling and combining data from multiple sources in the web. In this case, uncertainty may arise because of incomplete data or lack of confidence in the extractions. Figure 1.2(i) is an automatically extracted Car Ads database that may contain many uncertainties on the attribute values (this is called *attribute uncertainty*).

3. **Sensor data**: Large-scale instrumentation of nearly every aspect of our world using sensor monitoring infrastructures has generated an abundance of uncertain data [61,45,62]. In such applications, the presence of uncertainty is largely due to measurement noises or failures. In sensor databases, uncertain sensor readings are often captured by probabilistic models. For example, Deshpande et al. [62] model the sensor data using Gaussian distributions (Figure 1.2(ii)).

4. **Unknown facts**: A fact is, by its literal meaning, certain. But, very often, a fact can only be found out by an experiment at a certain cost. Therefore, before conducting the experiment, the fact is essentially uncertain to the de-

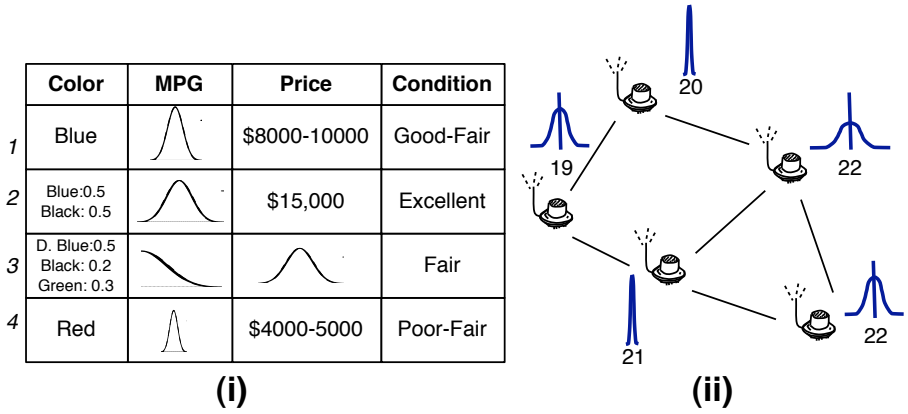


Figure 1.2: (i) An automatically extracted Car Ads database may contain many (attribute) uncertainties; (ii) Sensor data unavoidably contains complex, continuous uncertainties.

cision maker. In many cases, the uncertainty can be captured by probabilistic models which are usually constructed from partial information, known features, historical data and so on. In resource constrained scenarios, we need to make judicious decisions by utilizing such probabilistic information, without conducting experiments to find out all relevant facts, which can be prohibitively expensive. In Section 1.3, we elaborate two examples, kidney exchange and online dating, both involving designing policies to conduct experiments and optimizing the outcomes subject to some resource constraints.

The increasing volume of uncertain data has resulted in a need for efficiently supporting complex queries and decision-making over such data. In fact, various types of problems in decision making under uncertainty have been a subject of extensive research in computer science, economics, finance and social science. In my dissertation research, I concentrate on three important problems in this domain: (1) ranking under uncertainty, (2) maximizing utility under uncertainty, and (3) matching under uncertainty.

Before introducing the problems, we would like to add a few general remarks on decision making under uncertainty. There are many approaches that have been developed to handle uncertainty in a variety of areas. Perhaps the most naïve one is to somehow eliminate the uncertainty. For example, we can replace the uncertain values with the expected or most likely values, and make decisions solely based on these deterministic values. However, such approaches ignore the essential interaction between the uncertainty and the decision, and often lead to undesirable solutions. The most systematic way is to carry the uncertainty through the decision process. In principle, making a decision on an uncertain instance produces a random outcome, i.e., a distribution over possible outcomes. Typically, it is some characteristic of the random outcome that we care about. The characteristic, which is typically an aggregate value of some sort over that distribution of outcomes, is chosen as the decision criterion for optimization. This approach is also called the *possible worlds semantics*. We follow this approach for all of the three problems.

Quite often, choosing the correct decision criterion can be nontrivial. If the outcomes are numerical values, e.g., profits or costs, optimizing the expected value has been the most commonly used decision criterion, (partly) due to its mathematical tractability. In the matching problem, we use the expected value as the objective. However, researchers have discovered certain drawbacks of expected value and proposed more general alternatives. This observation leads the formulation of our second problem, which is based on the *expected utility theory*, a classic generalization of expected value that is known to be powerful in expressing diverse risk-aware behaviors in decision making. If the outcomes are not numerical values, other semantics may be needed. For instance, in the problem of ranking under uncertainty, the random outcome is a ranking (i.e., permutation) of a set of tuples. Choosing a proper decision criterion is a nontrivial problem here.

Now, we introduce and motivate three major problems in this dissertation one by one and also briefly state our contributions.

1.1 Ranking under Uncertainty

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 [100]. The deterministic setting of the problem consists of a universe of elements. Each element is associated with a score. The answer to the ranking problem is just a permutation of the elements in a non-increasing score order, while the answer to the top-k query is the set of k elements with the largest scores. Since we present the problem in the database context, we often use the terms "tuple" and "attribute". A tuple, which is a row in a database table, corresponds to an element and the key of the tuple is the unique id of the element. A tuple may have several attributes, e.g., cost, length, weight, and so on, and one attribute (possibly a derived attribute) is chosen as the score, according to which we rank the tuples.

In the probabilistic setting, the existence of each element or the score of each element may be uncertain. The uncertainty introduces complex trade-offs between scores and probabilities, which make the problem of ranking much harder than its counterpart in deterministic datasets. Let us consider a very simple example with two independent uncertain tuples t_1 (score = 200, $\Pr(t_1) = 1/3$), and t_2 (score = 50, $\Pr(t_2) = 1.0$) where $\Pr(t_i)$ is the probability that t_i exists for $i = 1, 2$. It may appear to some people that t_1 is better since t_1 has a score 4 times of t_2's score but a probability $1/3$ of t_2's. In other words, t_1 has a higher expected score. However, if we look at the top tuple over all outcomes (a.k.a. possible worlds), we can find the probability that t_1 is the top-1 answer is only $1/3$ (as long as t_1 exists, it is the top-1) whereas t_2 is the top-1 with probability $2/3$ (as long as t_1 does not exist, t_2 is the top-1). Noticing this, many risk-averse [1] users might choose t_2. Even in this simple case, it is not clear whether to rank t_1 above t_2 or vice versa. The

[1] Risk aversion is the reluctance of a person to accept a bargain with an uncertain payoff over another bargain with a more certain, but possibly lower payoff.

trade-off would become even more complicated if the scores of the tuples are also uncertain and are correlated with each other. This subtlety has led to a plethora of ranking functions being proposed in prior literature (we discuss most of them in detail in Section 2.4). Indeed, our empirical study demonstrates that the behaviors of several prior ranking functions can be quite diverse and even conflicting. Hence, the foremost challenge in ranking under uncertainty is to define a proper semantics that is capable of capturing different user preferences and, hopefully, unifying the previous ranking functions.

Further, often even semantically clear queries like "rank the tuples in the dataset based on their probabilities to be one of the top-10" (this is the *probabilistic threshold top-k* query [98], which in turn is a special case of our general ranking function), can be computationally nontrivial. Our second challenge is to design efficient algorithms, especially when the probability distributions of the tuples are correlated, in which case, we need to work on a compact correlation model to ensure efficiency, and when the distributions are continuous, in which case we need proper numerical techniques to handle continuous functions.

Our contributions (sketch): (Chapter 4–7) Ranking under uncertainty is the primary focus of the dissertation and and occupies a majority of the space. We conduct systematic studies of both the semantics and the algorithms for ranking and top-k query processing over probabilistic datasets. By illustrating the diverse and conflicting behavior of prior proposals, we contend that a single, specific ranking function may not suffice for probabilistic datasets. Instead we proposed the notion of "parameterized ranking functions"(PRF), which is a broad class of ranking functions. More specifically, the PRF value of tuple t is the expected value of a weight function $\omega(t, r_{pw}(t))$ where $r_{pw}(t)$ is the rank of t in possible world pw. By choosing different weight functions, PRF generalizes or can approximate many of the previously proposed ranking functions. We present novel exact or approximate algorithms for efficiently ranking large datasets according to these ranking

functions, even if the datasets exhibit complex correlations or the probability distributions are continuous. The time complexities of our algorithms match or improve the best known algorithms developed for several prior ranking functions (which are special cases of PRF). Second, we propose the notion of a consensus answer which, roughly speaking, is a deterministic answer that is "closest in expectation" to the possible answers over a probabilistic database. Under this framework, we obtain polynomial time optimal or approximation algorithms for computing the consensus top-k answers. We also show a close relationship between PRF and the consensus top-k answer semantics.

The results in Chapter 5 appear in [127, 128] while those in Chapter 6 appear in [125]. Chapter 7 is mainly based on [124].

1.2 Maximizing Utility under Uncertainty

The field of decision making under uncertainty is also known as *stochastic optimization* if the deterministic version of the problem under consideration is an optimization problem in the usual sense, such as a mathematical program or a combinatorial optimization problem. In this and the next section, we focus on stochastic combinatorial optimization problems.

The most common approach to deal with optimization problems in presence of uncertainty is to optimize the expected value of the solution. However, expected value is inadequate in capturing diverse people's preferences towards decision-making under uncertain scenarios. In particular, it fails at capturing different *risk-averse* or *risk-prone* behaviors that are commonly observed. Consider the following simple example where we have two lotteries L_1 and L_2. In L_1, the player could win 1000 dollars with probability 1.0, while in L_2 the player could win 2000 dollars with probability 0.5 and 0 dollars otherwise. It is easy to see that both have the same expected payoff of 1000 dollars. However, many, if not most, people would treat L_1 and L_2 as two completely different choices. Specifically, a risk-averse player

is likely to choose L_1 and a risk-prone player may prefer L_2 (Consider a gambler who would like to spend 1000 dollars to play double-or-nothing). A more involved but also more surprising example is the *St. Petersburg paradox* (see e.g., [132, 1]) which has been widely used in the economics literature as a criticism of expected value. See Section 2.6 for more details about the St. Petersburg paradox. These observations and criticisms have led researchers, especially in economics, to study the problem from a more fundamental perspective and to directly maximize user satisfaction, often called *utility*. The uncertainty present in the problem instance naturally leads us to optimize the *expected utility*.

Let $\mathcal{F}$ be the set of feasible solutions to an optimization problem. Each solution $S \in \mathcal{F}$ is associated with a random weight $w(S)$. For instance, $\mathcal{F}$ could be a set of lotteries and $w(S)$ is the (random) payoff of lottery S. We model the risk awareness of a user by a utility function $\mu : \mathbb{R} \to \mathbb{R}$: the user obtains $\mu(x)$ units of utility if the outcome is x, i.e., $w(S) = x$. Formally, the *expected utility maximization principle* (EUMP) is simply stated as follows: the most desirable solution S is the one that maximizes the expected utility, i.e.,

$$S = \arg\max_{S' \in \mathcal{F}} \mathbb{E}[\mu(w(S'))]$$

The theory was formally initiated by von Neumann and Morgenstern in 1940s [176, 70] and has been widely used to express diverse risk-averse or risk-prone behaviors. See Section 2.6 for more details.

In this dissertation, we consider the stochastic versions of a broad class of combinatorial problems including shortest paths, minimum weight spanning trees, and minimum weight matchings over probabilistic graphs, and other combinatorial problems like knapsack. Formally, the problem consists of a ground set of elements $U = \{e_i\}_{i=1\ldots n}$. Each element e is associated with a nonnegative random weight w_e. We assume all w_es are independent of each other. Each feasible solution is a subset of the elements satisfying some property. Let $\mathcal{F}$ denote the set of feasible

solutions. For example, $\mathcal{F}$ is the set of all s-t paths in the shortest path problem. We are also given a utility function $\mu : \mathbb{R}^+ \to \mathbb{R}^+$ which maps a weight value to a utility value. By the expected utility maximization principle, our objective is to find a feasible solution $S \in \mathcal{F}$ that maximizes $\mathbb{E}[\mu(w(S))]$. We call this problem the *expected utility maximization* (EUM) problem. Many stochastic optimization problems studied in literature are special cases of EUM. See below for an example. More examples can be found in Section 8.3.

Example 1. *(Stochastic Shortest Path): We are given a probabilistic graph where the length ℓ_e of each edge e is a random variable. The objective is to find an s-t path P connecting s and t such that the probability that the length of P is at most a given threshold T, i.e., $\Pr(\sum_{e\in P} \ell_e \leq T)$, is maximized. The problem has been studied in [141, 139]. To see the problem is a special case of* EUM*, just consider the utility function: $\mu(x) = 1$ for $x \leq T$ and $\mu(x) = 0$ for $x > T$. We study this problem in Section 8.3.*

Our contributions(sketch): (Chapter 8) We show that we can obtain a polynomial time approximation algorithm with *additive error* ϵ for any $\epsilon > 0$, if the utility function satisfies certain continuity or smoothness condition and there is a pseudopolynomial time algorithm for the *exact* version of the problem.[2] Our result generalizes several prior results on stochastic shortest path [141, 139], stochastic spanning tree [102], and stochastic knapsack [119, 79, 29]. For example, using our result, we can show there is a polynomial time algorithm that computes an s-t path P such that $\Pr(\sum_{e\in P} \ell_e \leq (1+\delta)T) \geq OPT - \epsilon$ for any fixed $\delta > 0$ and $\epsilon > 0$, where OPT is the optimal solution for the above stochastic shortest path problem. Moreover, our techniques can be generalized to handle multiple utility functions and multi-dimensional weight distributions. Our algorithm for utility

[2]Following the literature [145], we differentiate between *exact* version and *deterministic* version of a problem; in the exact version of the problem, we are given a target value and asked to find a solution (e.g., a path) with exactly that value (i.e., path length).

maximization makes use of the separability of exponential utility and a technique to decompose a general utility function into exponential utility functions, which may be useful in other stochastic optimization problems.

The results in this chapter appear in [126].

1.3 Matching under Uncertainty

We study the problem of finding a matching of maximum weight in an uncertain graph. Maximum weight matching is a fundamental graph optimization problem and has found numerous applications. It is well-known that the problem can be solved exactly in polynomial time in the deterministic setting. However, in practical applications, the pairwise relations between vertices are often uncertain in the decision making stage. This motivates us to study its stochastic variants. In particular, we consider the following *stochastic matching* problem. We are given a probabilistic graph where each possible edge e is present independently with some probability p_e. Given these probabilities, we want to build a large/heavy matching in the randomly generated graph. However, the only way we can find out whether an edge is present or not is to probe it, and if the edge is indeed present in the graph, we are forced to add it to our matching. Further, each vertex i is associated with a patience level t_i which means at most t_i edges incident on i can be probed. The question is how should we *adaptively* query the edges to maximize the expected weight of the matching.

The problem is motivated by the recent kidney exchange program and the popular online dating application which we now briefly describe.

- *Kidney Exchange*: It happens very often that a friend or a family member of the patient would like to donate a kidney to the patient but the kidney is incompatible with the patient's body. To resolve this problem, the United Network for Organ Sharing (UNOS) launched in year 2000 the kidney exchange program in which two incompatible patient/donor pairs are identified

such that each donor is compatible with the other pair's patient [154, 155]. Then, exchanging the kidneys between the pairs can be performed in order to have two successful transplants. Clearly, the goal is to match the maximum number of pairs. This problem can be modeled as a maximum matching instance in which each node represents an incompatible pair and each edge represents a possible exchange. To decide compatibility, three main tests which indicate the likelihood of successful transplants should be performed. The first two tests, the blood-type test and the antibody screen, compare the blood of the recipient and donor. The third test, called crossmatching, is the most critical one and the feasibility of a transplant can only be determined after this test. However, this test is time-consuming and must be performed close to the surgery date. Therefore, as soon as a pair passes the crossmatch test, the transplant should be performed immediately. Thus, we can model the probability that the exchange between two nodes (incompatible pairs) will succeed based on the initial two tests by a probabilistic edge. The crossmatch tests performed between two nodes correspond to a probe on that edge. Upon a successful probe, the exchange should be performed which means we include this edge in our matching. The patience level for each vertex models the fact that a patient will eventually die without a successful match.

- *Online Dating*: In an online dating system, e.g., eHarmony, users submit their profiles to the central server. The server then estimates the compatibilities of men and women based on their profiles and then sends plausibly compatible couples on blind dates. To see how this may be modeled as our stochastic matching problem, we just think each person in the system as a node and the edge probability between a pair of nodes as the probability that the pair is compatible, which is estimated by the server based on the profiles of the pair. Probing an edge corresponds to sending the pair on a date in this case. The

patience level of a node indicates each person is only willing to participate in at most a given number of unsuccessful dates.

Our contributions(sketch): (Chapter 9) We first consider a more general problem, called *stochastic k-set-packing*. In this problem, we try to pack hyperedges of size k with random sizes and profits into a graph with d vertices, each having a capacity constraint. The size of each hyperedge is a k-dimensional random 0/1 vector. The stochastic k-set-packing problem directly generalizes the stochastic matching problem (for $k = 4$; see the reduction in Section 9.2). Our goal is to design a probing strategy such that the expected profit is maximized. We show that there is a $2k$-approximation algorithm for this problem. When the column outcomes are monotone (see the definition in Section 9.2.1), we can use the *FKG inequality* to strengthen the probability bound and show that the approximation ratio is at most $k + 1$. This implies a 5-approximation for weighted stochastic matching, which answers an open question from [40]. Then, we design improved probing strategies for stochastic matching by making use of the graph structure and the *dependent rounding* scheme [73]. In particular, we give a 4-approximation for weighted stochastic matching on general graphs, and a 3-approximation on bipartite graphs. The probing strategy returned by the algorithm can in fact be made *matching-probing*. In the more restrictive matching-probing model, we can probe a set of vertex disjoint edges (i.e., a matching) in each round and there are k rounds, where k is a given parameter. We introduce a generalization of the stochastic *online* matching problem [68] that also models preference-uncertainty and timeouts of buyers, and give a constant factor approximation algorithm.

The results in this chapter are mainly based on [16, 17].

Chapter 2

Preliminaries

In this chapter, we review some preliminary knowledge that is necessary for latter chapters. We first briefly review the prevalent possible worlds semantics, and then discuss how decision making and query processing over probabilistic datasets are typically done under possible worlds semantics. Next, we introduce some probabilistic data models that we use throughout the article, including the probabilistic and/xor tree and the Markov network. At last, we briefly describe the St. Petersburg paradox and the expected utility theory. The later semantically motivates and underpins many problem formulations in this thesis.

2.1 Possible Worlds Semantics

The common semantics in decision making under uncertainty are the *possible worlds semantics*, where an uncertain instance is considered to correspond to a probability distribution over a set PW of deterministic instances $\{pw_1, pw_2, ..., pw_N\}$ called possible worlds. We use $p(pw)$ to denote the probability of possible world pw. Because of the typically exponential size of PW, an explicit possible worlds representation is not feasible, and hence the semantics are usually captured implicitly by probabilistic models with polynomial size specification. In Section 2.3, we introduce several probabilistic data models that we use in the thesis.

Under possible worlds semantics, making a decision in the uncertain instance corresponds to making the decision over the possible worlds. Thus, we obtain a distribution of possible outcomes, each resulting from the decision on some world. The decision maker chooses the decision which optimizes the decision criterion,

which is typically defined as some aggregate value of the outcome distribution. We discuss some commonly used decision criterions in the next section.

Our ranking problem is studied and presented in the probabilistic database context. Conceptually, a *probabilistic database* is just a probability distribution over deterministic databases, despite the actual probabilistic model and physical implementation being used. Under possible worlds semantics, posing queries over such a probabilistic database generates a probability distribution over a set of deterministic results which we call "possible answers". However, a full list of possible answers together with their probabilities is not desirable in most cases since the size of the list could be exponentially large, and the probability associated with each single answer is extremely small. One approach to addressing this issue is to "combine" the possible answers somehow to obtain a more compact representation of the result. For simple SQL queries that return a list of tuples (often called *select-project-join* queries), one proposed approach is to union all the possible answers, and compute the probability of each result tuple by adding the probabilities of all the possible answers it belongs to [54]. This approach, however, cannot be easily extended to other types of queries like ranking or aggregate queries.

2.2 Stochastic Optimization

If a deterministic problem is an optimization problem in the usual sense, e.g., a mathematical optimization (e.g., linear programming) or a combinatorial optimization problem, the corresponding optimization problem under uncertainty is often known as a *stochastic optimization* problem. Suppose the objective function for the deterministic problem is $w : \mathcal{A} \times \mathcal{F} \to \mathbb{R}$, where $\mathcal{A}$ is the set of problem instances and $\mathcal{F}$ is the solution space. Typically, the decision criterion (also called *the objective*) of the stochastic optimization problem is some aggregate value of the distribution of $w(pw, S)$, where pw is a possible world drawn from the probabilistic instance and S is the decision. For different applications, we may choose

different aggregate values as the objectives. For example, the following objectives are considered in this dissertation:

- **Expected value**: We would like to find the solution S that optimizes

$$\mathbb{E}_{pw}[w(pw, S)] = \sum_{pw \in PW} w(pw, S)p(pw).$$

  This is the most commonly used objective for stochastic optimization.

- **Overflow probability**: For a given value γ, we optimize the probability

$$\Pr(w(pw, S) \geq \gamma) = \sum_{pw : w(pw,S) \geq \gamma} p(pw).$$

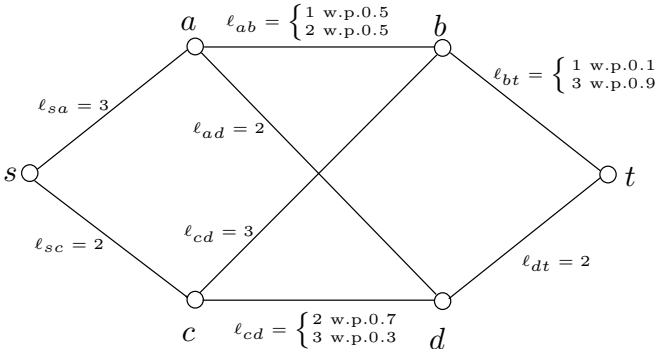
- **Expected utility**: Given a utility function $\mu : \mathbb{R} \to \mathbb{R}$, we optimize the expected utility

$$\mathbb{E}_{pw}[w(pw, S)] = \sum_{pw \in PW} \mu(w(pw, S))p(pw).$$

  This generalizes expected value (where $\mu(x) = x$) and overflow probability (where $\mu(x) = 0$ for $x \leq \gamma$ and $\mu(x) = 1$ otherwise). The motivation for using other utility functions is discussed in Section 2.6.

Now, we use the stochastic shortest path problem as an illustrative example.

Example 2. *(Stochastic Shortest Path) We illustrate this problem through an example. Consider the probabilistic graph shown in Figure 2.1. The length ℓ_e of each edge e is an independent random variable. If the objective is the expected length, the optimal path is $\{s, c, d, t\}$ whose expected length is $(2+2+2) \times .7 + (2+3+2) \times .3 = 6.3$. In fact, due to the linearity of expectation, minimizing the expected path length can be reduced to the deterministic shortest path problem by using $\mathbb{E}[\ell_e]$ as the length of edge e. If the user wants to maximize the probability that the length of*



PW	ℓ_{ab}	ℓ_{cd}	ℓ_{bt}	Prob
pw_1	1	2	1	.035
pw_2	1	2	3	.315
pw_3	1	3	1	.015
pw_4	1	3	3	.135
pw_5	2	2	1	.035
pw_6	2	2	3	.315
pw_7	2	3	1	.015
pw_8	2	3	3	.135

Figure 2.1: A probabilistic graph with three uncertain edges. There are $2^3 = 8$ possible worlds.

the path is at most 5, the optimal path is $\{s, a, b, t\}$*. The optimal probability is* $p(pw_1) + p(pw_3) = 0.05$*. Suppose the objective is to maximize the expected utility with utility function* $\mu(x) = 0.9^x$*. The optimal path is* $\{s, c, d, t\}$ *with expected utility value* $\mu(2 + 2 + 2) \times .7 + \mu(2 + 3 + 2) \times .3 \approx 0.515$*. This problem is studied in Chapter 8.*

Instead of replacing the deterministic objective with a stochastic one, we could also use stochastic constraints (with or without a stochastic objective) to form a stochastic optimization problem. The most common stochastic constraint is the *chance constraint* that asserts the probability that a random event happens should be at least (or at most) a given threshold. The following stochastic knapsack problem is a typical chance-constrained stochastic optimization problem.

Example 3. *(Stochastic knapsack) We are given a set* U *of* n *items and a positive constant* $0 \leq \gamma \leq 1$*. Each item* i *has a random size* w_i *and a deterministic profit* v_i*. The goal is to find a subset* $S \subseteq U$ *such that the chance constraint* $\Pr(w(S) \leq 1) \geq \gamma$ *holds and the total profit* $v(S) = \sum_{i \in S} v_i$ *is maximized. This problem is studied in Chapter 8.*

Stochastic optimization problems can be further classified by the number of stages in the problem, where in each stage a partial decision is made and the

random effect will be carried over to later stages. The ranking and utility maximization problems we study involve only one stage, i.e., the decision must be made before the random experiment, while the matching problem is a multi-stage decision problem.

Many optimization problems are computationally hard, e.g., NP-hard or #P-hard, which implies that it is very unlikely that there is a polynomial time algorithm that can solve the problem exactly. A common way to tackle such a problem is from the approximation algorithms perspective, that is to design efficient (typically polynomial time) algorithms which find solutions with objective values close to the optimum. For a maximization (resp. minimization) problem, an α-approximation algorithm is one that computes a solution with objective value at least $1/\alpha$ (resp. at most α) times the value of the optimal solution. A *polynomial time approximation scheme (PTAS)* is an algorithm which takes an instance of a maximization (resp. minimization) problem and a parameter $\epsilon > 0$ and produces a solution whose cost is at least a factor $1 - \epsilon$ (resp. at most a factor of $1 + \epsilon$) of the optimum, and the running time, for any fixed ϵ, is polynomial in the size of the input. Even more restrictive is the *fully polynomial-time approximation scheme (FPTAS)*, which requires the algorithm to be polynomial in both the size of the input and $\frac{1}{\epsilon}$. Sometimes, we can obtain *additive approximations*. We say an algorithm is a β-additive approximation, if the absolute difference between the value of the solution found by the algorithm and the optimum is at most β.

2.3 Probabilistic Data Models

Our general data model consists of a set of ground elements. Each element is typically associated with a weight (also interchangeably called score). We may have two types of uncertainties, *existence uncertainty* (the existence of each element is uncertain) and *value uncertainty* (the weight of each element is uncertain).

The ranking problem is presented in the probabilistic database context. We

assume that the elements are stored in a probabilistic table (a.k.a. probabilistic relation). A tuple, which is a row in the table, corresponds to an element. The key of the tuple is the unique id of the element. The key is also called the *possible worlds key*, which means the key must be the unique identifier of a tuple in any possible world. We note that in a probabilistic relation, there may be several tuples having the same possible world key. However, they must be mutual exclusive, i.e., at most one of them is present in a possible world. A tuple may have several attributes, but only one attribute is chosen as the ranking criterion, which we call the score of the tuple. Note that the score can be a value derived from the values of other attributes. Existence uncertainty and value uncertainty are also called *tuple-level uncertainty* (or *tuple uncertainty*) and *attribute-level uncertainty* [1] (or *attribute uncertainty*), respectively, in probabilistic database terminology.

If all elements (tuples) are independent of each other, the model is *element-independent* (*tuple-independent*). In our utility maximization and matching problems, we only consider the element-independent model. For the ranking problem, we consider two correlation models, the probabilistic and/xor tree model and the Markov network model. We can handle arbitrarily correlated relations with correlations modeled using Markov networks. However, in many parts of this work, we focus on the *probabilistic and/xor tree model*, that can capture only a more restricted set of correlations, but admits highly efficient ranking algorithms.

2.3.1 Probabilistic And/Xor Trees

A probabilistic and/xor tree captures two types of correlations: (1) *mutual exclusivity* (denoted $\veebar$ (*xor*)) and (2) *mutual co-existence* ($\wedge$ (*and*)). Two events satisfy the mutual co-existence correlation if, in any possible world, either both events occur or neither occurs. Similarly two events are mutually exclusive if there

[1] In general, attribute-level uncertainty allows for each tuple to have multiple uncertain attributes. Since only the score attribute is relevant in the ranking problem, attribute-level uncertainty means that the score of each tuple is uncertain.

Time	Car Loc.	Plate No.	Speed	.....	Prob	Tuple Id
11:40	L1	X-123	120	...	0.4	t_1
11:55	L2	Y-245	130	...	0.7	t_2
11:35	L3	Y-245	80	...	0.3	t_3
12:10	L4	Z-541	95	...	0.4	t_4
12:25	L5	Z-541	110	...	0.6	t_5
12:15	L6	L-110	105	...	1.0	t_6

Possible Worlds	Prob
$pw_1 = \{t_2, t_1, t_6, t_4\}$	.112
$pw_2 = \{t_2, t_1, t_5, t_6\}$	.168
$pw_3 = \{t_1, t_6, t_4, t_3\}$	.048
$pw_4 = \{t_1, t_5, t_6, t_3\}$	.072
$pw_5 = \{t_2, t_6, t_4\}$	.168
$pw_6 = \{t_2, t_5, t_6\}$	.252
$pw_7 = \{t_6, t_4, t_3\}$	.072
$pw_8 = \{t_5, t_6, t_3\}$	.108

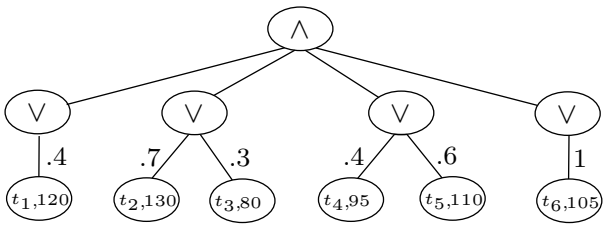


Figure 2.2: Example of a probabilistic database which contains automatically captured information about speeding cars. Tuple t_2 and t_3 (similarly, t_4 and t_5) are mutually exclusive. The corresponding and/xor tree compactly encodes these correlations.

is no possible world where both happen.

Now, let us formally define a probabilistic and/xor tree. In tree $\mathcal{T}$, we denote the set of children of node v by $Ch_{\mathcal{T}}(v)$ and the least common ancestor of two leaves l_1 and l_2 by $LCA_{\mathcal{T}}(l_1, l_2)$. We omit the subscript if the context is clear.

Definition 1. *A probabilistic and/xor tree $\mathcal{T}$ represents the mutual exclusion and co-existence correlations in a probabilistic dataset. In $\mathcal{T}$, each leaf is an id-value pair, representing a specific element with a specific weight value. Each inner node has a mark, $\circledvee$ or $\circledwedge$. For each $\circledvee$ node u and each of its children $v \in Ch(u)$, there is a nonnegative value $p_{(u,v)}$ associated with the edge (u, v). Moreover, we require*

- *(Probability Constraint)* $\sum_{v:v\in Ch(u)} p_{(u,v)} \leq 1$.
- *(Id Constraint) For any two different leaves l_1, l_2 holding the same id, $LCA(l_1, l_2)$ is a $\circledvee$ node*[2].

Let $\mathcal{T}_v$ be the subtree rooted at v and $Ch(v) = \{v_1, \ldots, v_\ell\}$. The subtree $\mathcal{T}_v$ inductively defines a random subset S_v of its leaves by the following independent

[2]The id constraint is imposed to avoid two elements with the same id but different attribute values coexisting in a possible world.

Possible Worlds	Prob
$pw_1 = \{(t_3, 6), (t_2, 5), (t_1 1)\}$	.3
$pw_2 = \{(t_3, 9), (t_1, 7)\}$	.3
$pw_3 = \{(t_2, 8), (t_4, 4), (t_5, 3)\}$	.4

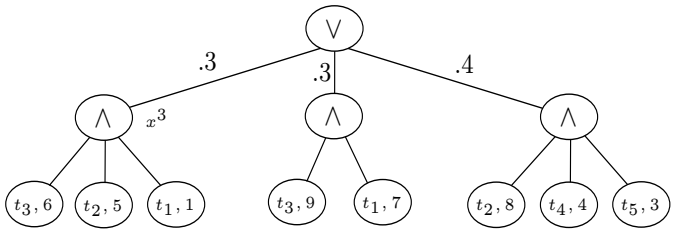


Figure 2.3: Example of a highly correlated probabilistic database with 3 possible worlds and the and/xor tree that captures the correlation.

process:

- *If v is a leaf, $S_v = \{v\}$.*
- *If $\mathcal{T}_v$ roots at a $\veebar$ node, then*
$$S_v = \begin{cases} S_{v_i} & \text{with probability } p_{(v,v_i)} \\ \emptyset & \text{with probability } 1 - \sum_{i=1}^{\ell} p_{(v,v_i)} \end{cases}$$
- *If $\mathcal{T}_v$ roots at a $\wedge$ node, then $S_v = \cup_{i=1}^{\ell} S_{v_i}$*

This model subsumes a popular probabilistic data model, called x-tuples [157, 182] which can be used to specify mutual exclusivity correlations between tuples. Specifically, x-tuples correspond to the special case where we have a tree of *height* 2, with a $\wedge$ node as the root and only $\veebar$ nodes in the second level. Figure 2.2 shows an example of an and/xor tree that models the data from a traffic monitoring application [167], where the tuples represent automatically captured traffic data. The inherent uncertainty in the monitoring infrastructure is captured using an and/xor tree, that encodes the tuple existence probabilities as well as the correlations between the tuples. For example, the leftmost $\veebar$ node indicates t_1 is present with probability .4 and the second $\veebar$ node dictates that exactly one of t_2 and t_3 should appear. The topmost $\wedge$ node tells us the random sets derived from

these $\veebar$ nodes coexist.

We note that and/xor trees are able to represent any finite set of possible worlds. This can be done by listing all possible worlds, creating one $\wedge$ node for each world, and using a $\veebar$ node as the root to capture that these worlds are mutual exclusive. Figure 2.3 shows how we can capture arbitrary possible worlds using an and/xor tree.

Probabilistic and/xor trees significantly generalize *x-tuples* [157, 182], block-independent disjoint tuples model, and p-or-sets [55]. The correlations captured by such a tree can be represented by probabilistic c-tables [86] and provenance semirings [85]. However, that does not directly imply an efficient algorithm for ranking. We remark that Markov or Bayesian network models are able to capture more general correlations in a compact way [159], however, the structure of the model is more complex and probability computations on them (inference) is typically exponential in the treewidth of the model. The treewidth of an and/xor tree (viewing it as a Markov network) is not bounded, and hence the techniques developed for those models cannot be used to obtain a polynomial time algorithms for and/xor trees. *We note that no prior work on ranking in probabilistic databases has considered more complex correlations than x-tuples.*

2.3.2 Markov Networks

Among many models for capturing the correlations in a probabilistic database, graphical models (Markov or Bayesian networks) perhaps represent the most systematic approach [161]. The appeal of graphical models stems both from the pictorial representation of the dependencies, and a rich literature on doing inference over them. In this section, we briefly review some notations and definitions related to Markov networks and junction trees. We only consider tuple-level uncertainty. Let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of tuples. For each tuple t in T, we associate an indicator random variable X_t, which is 1 if t is present, and 0 otherwise. Let $\mathcal{X} = \{X_{t_1}, \ldots, X_{t_n}\}$. For a set of variables S, we use $\Pr(S)$ to denote

the joint probability distribution over those variables. So $\Pr(\mathcal{X})$ denotes the joint probability distribution that we are trying to reason about. This joint distribution captures all the correlations in the dataset. However, directly trying to represent it would take $O(2^n)$ space, and hence is clearly infeasible.

Probabilistic graphical models allow us to represent this joint distribution compactly by exploiting the conditional independences present among the variables. Given three disjoint sets of random variables A, B, C, we say that A is conditionally independent of B *given* C if and only if:

$$\Pr(A, B|C) = \Pr(A|C) \Pr(B|C)$$

We assume that we are provided with a *junction tree* over the variables $\mathcal{X}$ that captures the correlations among them. A junction tree can be constructed from a graphical model using standard algorithms [69]. Recently junction trees have also been used as a internal representation for probabilistic databases, and have been shown to be quite effective at handling lightly correlated probabilistic databases [111]. We describe the key properties of junction trees next.

Junction tree Let $\mathcal{T}$ be a tree with each node v associated with a subset $C_v \subseteq \mathcal{X}$. We say $\mathcal{T}$ is a *junction tree* if any intersection $C_u \cap C_v$ for any $u, v \in \mathcal{T}$ is contained in C_w for every node w on the unique path between u and v in $\mathcal{T}$ (this is called the *running intersection property*). The treewidth tw of a junction tree is defined to be $\max_{v \in \mathcal{T}} |C_v| - 1$.

Denote $S_{u,v} = C_v \cap C_u$ for each edge $(u, v) \in \mathcal{T}$. We call $S_{u,v}$ a *separator* since removal of $S_{u,v}$ disconnects the graphical model. The set of conditional independences embodied by a junction tree can be found using the Markov property:

(Markov Property) Given variable sets A, B, C, if C separates A and B (i.e., removal of variables in C disconnects the variables in A from variables in B in the junction tree), then A is conditionally independent of B given C.

Example 4. *Let $T = \{t_1, t_2, t_3, t_4, t_5\}$. Figure 2.4 (i) and (ii) show the (undirected)*

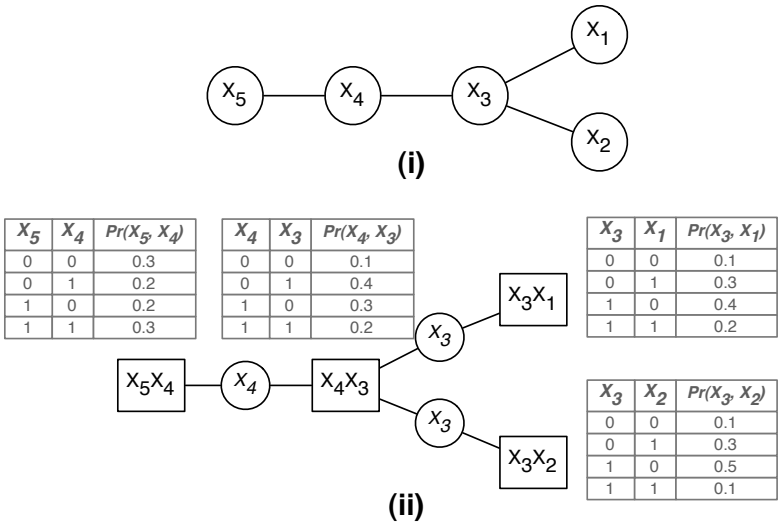


Figure 2.4: (i) A graphical model; (ii) A junction tree for the model along with the (calibrated) potentials.

graphical model and the corresponding junction tree $\mathcal{T}$. $\mathcal{T}$ *has four nodes:* $C_1 = \{X_{t_4}, X_{t_5}\}$, $C_2 = \{X_{t_4}, X_{t_3}\}$, $C_3 = \{X_{t_3}, X_{t_1}\}$ *and* $C_4 = \{X_{t_3}, X_{t_2}\}$. *The treewidth of* $\mathcal{T}$ *is* 1. *We have,* $S_{1,2} = \{X_4\}$, $S_{2,3} = \{X_3\}$ *and* $S_{2,4} = \{X_3\}$. *Using the Markov property, we observe that* X_5 *is independent of* X_1, X_2, X_3 *given* X_4.

Clique and Separator Potentials With each clique C_v in the junction tree, we associate a *potential* $\pi_v(C_v)$, which is a function over all variables $X_{t_i} \in C_v$ and captures the correlations among those variables. Similarly, with each separator $S_{u,v}$, we associate a *potential* $\mu_{u,v}(S_{u,v})$. Without loss of generality, we assume that the potentials are *calibrated*, i.e., the potential corresponding to a clique (or a separator) is exactly the joint probability distribution over the variables in that clique (separator). Given a junction tree with arbitrary potentials, calibrated potentials can be computed using a standard *message passing algorithm* [69]. The complexity of this algorithm is $O(n2^{tw})$. Then the joint probability distribution of $\mathcal{X}$, whose correlations can be captured using a calibrated junction tree $\mathcal{T}$, can be

written as:

$$\Pr(\mathcal{X}) = \frac{\prod_{v \in \mathcal{T}} \pi_v(C_v)}{\prod_{(u,v) \in \mathcal{T}} \mu_{u,v}(S_{u,v})} = \frac{\prod_{v \in \mathcal{T}} \Pr(C_v)}{\prod_{(u,v) \in \mathcal{T}} \Pr(S_{u,v})}$$

2.4 Prior Semantics on Ranking over Probabilistic Data

The interplay between probabilities and scores complicates the semantics of ranking on probabilistic datasets. This was observed by Soliman et al. [167], who first considered this problem and presented two definitions of top-k queries. Several other definitions of ranking have been proposed since then. We briefly review the ranking functions [3] we consider in this work. We use the dataset shown in Figure 2.2 to illustrate the ranking functions.

- **Uncertain Top-k (U-Top)** [167]: Here the query returns the k-tuple set that appears as the top-k answer in most possible worlds (weighted by the probabilities of the worlds).

  As an example, for the dataset shown in Figure 2.2, the top-2 answer under this semantics is $\{t_2, t_1\}$, since $\Pr(\{t_2, t_1\}$ is the top-2 answer$) = p(pw_1) + p(pw_2) = 0.112 + 0.168 = 0.28$.

- **Uncertain Rank-k (U-Rank)** [167]: At rank i, we return the tuple with the maximum probability of being at the i'th rank in all possible worlds. In other words, U-Rank returns: $\{t_i^*, i = 1, .., k\}$, where $t_i^* = \arg\max_t(\Pr(r(t) = i))$.

  For example, the top-2 answer for our example dataset is $\{t_2, t_5\}$. This is because $\Pr(r(t_2) = 1) = p(pw_1) + p(pw_2) + p(pw_5) + p(pw_6) = 0.7$ and $\Pr(r(t_5) = 2) = p(pw_4) + p(pw_6) = 0.324$.

[3] Unlike in deterministic settings, the answer to a top-k query on a set of uncertain tuples may not be the length-k prefix of the ranking of the tuples computed according to some ranking function. In Section 4.5.2, we draw a clear distinction between the two terms. However, we do not distinguish the terms "ranking" and "top-k" and use them interchangeably in the rest of the dissertation.

- **Probabilistic Threshold Top-k (PT(h))** [98]: The original definition of a probabilistic threshold query asks for all tuples with probability of being in top-h answer larger than a pre-specified threshold, i.e., all tuples t such that $\Pr(r(t) \leq h) > \mathit{threshold}$. For consistency with other ranking definitions, we slightly modify the definition and instead ask for the k tuples with the largest $\Pr(r(t) \leq h)$ values. Zhang et al. [183] use the choice $k = h$ and call the resulting special case *global top-k queries*.

  For example, we can see that $\Pr(r(t_1) \leq 2) = p(pw_1) + p(pw_2) + p(pw_3) + p(pw_4) = 0.4$. Similarly, $\Pr(r(t_2) \leq 2) = 0.7$, $\Pr(r(t_3) \leq 2) = 0$, $\Pr(r(t_4) \leq 2) = 0.072$, $\Pr(r(t_5) \leq 2) = 0.432$ and $\Pr(r(t_6) \leq 2) = 0.396$. Hence, the top-2 answer is $\{t_2, t_5\}$ under PT(2) semantics.

- **Expected Rank (E-Rank)** [48]: The tuples are ranked in the increasing order by:

$$\mathbb{E}_{pw}[r_{pw}(t)] = \sum_{pw \in PW} p(pw) r_{pw}(t),$$

  where $r_{pw}(t) = |pw| + 1$ if $t \notin pw$.

  For example, $\mathbb{E}_{pw}[r_{pw}(t_1)] = 2 \times (p(pw_1) + p(pw_2)) + 1 \times (p(pw_3) + p(pw_4)) + 4 \times (p(pw_5) + p(pw_6) + p(pw_7) + p(pw_8)) = 2.92$.

- **Expected Score (E-Score)**: Another natural ranking function, also considered by [48], is simply to rank the tuples by their expected score, $\Pr(t)s(t)$.

- **c-typical top-k** (TYP-Top) [76]: Let S_{pw} (a random variable) be the total score of the top-k answer in possible world pw. Let $s_1, \ldots, s_c$ be c values such that $\mathbb{E}_{pw}[\min_i |S_{pw} - s_i|]$ is minimized where each s_i is some possible value of S_{pw}. The answer to the query is a set of c top-k lists $l_1, \ldots, l_c$ such that l_i is the most probable top-k list that has a total score s_i.

  Suppose $c = 2$ and $k = 1$. The distribution of the total score of the top-1 answer is $\{130, \text{w.p.}0.7;\ 120, \text{w.p.}0.12;\ 110, \text{w.p.}0.108;\ 105, \text{w.p.}0.072\}$. It is not hard to see that $s_1 = 130$ and $s_2 = 110$. Hence, $l_1 = \{t_2\}$ and $l_2 = \{t_5\}$.

2.5 Distance between Two Top-k Answers

In this section, we review some popular distance functions between two permutations or two top-k lists. The distance function is a measurement of (dis)similarity and a higher value indicates a larger disagreement. Fagin et al. [66] provide a comprehensive analysis of the problem of comparing two top-k lists. They present extensions of the Kendall's tau and Spearman footrule metrics (defined on full rankings) to top-k lists and propose several other natural metrics, such as the intersection metric and Goodman and Kruskal's gamma function. We consider four metrics discussed in that paper: the symmetric difference metric, the intersection metric and one particular extension to Spearman's footrule distance and an extension to Kendall's tau distance. We briefly recall some definitions here. For more details and the relation between different definitions, please refer to [66].

We use the symbol τ to denote a top-k ranked list, and τ^i to denote the restriction of τ to the first i items. We use $\tau(i)$ to denote the i^{th} item in the list τ for positive integer i, and $\tau(t)$ to denote the position of $t \in T$ in τ.

Symmetric Difference Given two top-k lists, τ_1 and τ_2, the symmetric difference metric is defined as:

$$\mathsf{dis}_\Delta(\tau_1, \tau_2) = \frac{1}{2k}|\tau_1 \Delta \tau_2| = \frac{1}{2k}|(\tau_1 \backslash \tau_2) \cup (\tau_2 \backslash \tau_1)|.$$

Intersection Metric: While dis_Δ focuses only on the membership, the intersection metric dis_I also takes the order of tuples into consideration. It is defined to be:

$$\mathsf{dis}_I(\tau_1, \tau_2) = \frac{1}{k}\sum_{i=1}^{k} \mathsf{dis}_\Delta(\tau_1^i, \tau_2^i)$$

We note that both $\mathsf{dis}_\Delta()$ and $\mathsf{dis}_I()$ values are always between 0 and 1.

Spearman's Footrule: The original Spearman's Footrule metric is defined as the L_1 distance between two permutations σ_1 and σ_2. Formally, $F(\sigma_1, \sigma_2) =$

$\sum_{t\in T} |\sigma_1(t) - \sigma_2(t)|$. Let ℓ be a integer greater than k. The *footrule distance with location parameter* ℓ, denoted $F^{(\ell)}$ generalizes the original footrule metric. It is obtained by placing all missing elements in each list at position ℓ and then computing the usual footrule distance between them. A natural choice of ℓ is $k+1$ and we denote $F^{(k+1)}$ by dis_F. It is also proven that dis_F is a real metric and a member of a big and important equivalence class [4] [66].

It is shown in [66] that:

$$\mathsf{dis}_F(\tau_1, \tau_2) = (k+1)|\tau_1 \Delta \tau_2| + \sum_{t\in\tau_1\cap\tau_2} |\tau_1(t) - \tau_2(t)| - \sum_{t\in\tau_1\setminus\tau_2} \tau_1(t) - \sum_{t\in\tau_2\setminus\tau_1} \tau_2(t).$$

Kendall's tau: Another prevalent distance function is *Kendall's tau* distance defined for comparing top-k answers [67]. It is also called *Kemeny distance* in the literature and is considered to have many advantages over other distance metrics [64]. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ denote two full ranked lists, and let $\mathcal{K}_1$ and $\mathcal{K}_2$ denote the top-k ranked tuples in $\mathcal{R}_1$ and $\mathcal{R}_2$ respectively. Then *Kendall tau distance* between $\mathcal{K}_1$ and $\mathcal{K}_2$ is defined to be:

$$\mathsf{dis}(\mathcal{K}_1, \mathcal{K}_2) = \sum_{(i,j)\in P(\mathcal{K}_1,\mathcal{K}_2)} \hat{K}(i,j),$$

where $P(\mathcal{K}_1, \mathcal{K}_2)$ is the set of all unordered pairs of $\mathcal{K}_1 \cup \mathcal{K}_2$; $\hat{K}(i,j) = 1$ if it can be inferred from $\mathcal{K}_1$ and $\mathcal{K}_2$ that i and j appear in opposite order in the two full ranked lists $\mathcal{R}_1$ and $\mathcal{R}_2$, otherwise $\hat{K}(i,j) = 0$. Intuitively the Kendall distance measures the number of inversions or flips between the two rankings. Sometimes, for ease of comparison, we divide the Kendall distance by k^2 to obtain *normalized Kendall distance*, which always lies in $[0, 1]$. We adopt Kendall distance (or the normalized version) for our experiments. To get some intuition, it is easy to see that if the Kendall distance between two top-k answers is δ, then the two answers

[4] All distance functions in one equivalence class are bounded by each other within a constant factor. This class includes several extensions of Spearman's footrule and Kendall's tau metrics.

must share at least $1 - \sqrt{\delta}$ fraction of tuples [5] (so if the distance is 0.09, then the top-k answers share at least 70%, and typically 90% or more tuples). The distance is 1 only if two top-k answers are completely disjoint.

2.6 St. Petersburg Paradox and Expected Utility Theory

The St. Petersburg paradox is a paradox related to decision making under uncertainty. It is a classic example where the expected value criterion, i.e., making decisions solely based on the expected value of the objective, suggests a course of action that no rational person would be willing to take. It is therefore often used as a criticism of the expected value criterion. The paradox is named from Daniel Bernoulli's presentation of the problem, published in 1738 in the *Commentaries of the Imperial Academy of Science of Saint Petersburg*.

The paradox is as follows. Consider the following game: you pay a fixed fee X to enter the game. In the game, a fair coin is tossed repeatedly until a tail appears ending the game. The payoff of the game is 2^k where k is the number of heads that appears., i.e., you win 1 dollar if a tail appears on the first toss, 2 dollars if a head appears on the first toss and a tail on the second, 4 dollars if a head appears on the first two tosses and a tail on the third and so on. The question is what would be a fair fee X to enter the game? First, it is easy to see that

$$\begin{aligned}\mathbb{E}[\text{payoff}] &= \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 4 + \frac{1}{16} \cdot 8 + \cdots \\ &= \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \cdots = \sum_{k=1}^{\infty} \frac{1}{2} = \infty\end{aligned}$$

If we use the expected payoff as a criterion for decision making, we should therefore play the game at any finite price X (no matter how large X is) since the expected

[5] To see this, consider the worst case where two top-k lists which share the prefix of length $k - x$. The remaining parts of the two lists are disjoint. The Kendall distance is $(\frac{x}{k})^2$ in this case.

payoff is always larger. However, researchers have done extensive survey and found that not many people would pay even 25 dollars to play the game [132], which significantly deviates from what the expected value criterion predicts. We refer interested reader to [132, 1] for more information.

In fact, the paradox can be resolved by the classic expected utility theory with a logarithmic utility function, suggested by Bernoulli himself. The expected utility theory is a branch of the utility theory that studies "betting preferences" of people with regard to uncertain outcomes (gambles). The theory was formally initiated by von Neumann and Morgenstern in 1940s [176, 70][6] who gave an axiomatization of the theory (known as *von Neumann-Morgenstern expected utility theorem*). Since then, the expected utility theory is widely used in economics and psychology to explain diverse risk-aware behaviors under uncertainty.

Roughly speaking, the theory suggests to use a mathematical function to correct the expected value depending on probability, to account for the risk-averse or risk-prone behaviors. The mathematical function is called the utility function that indicates the level of the user satisfaction associated with different objective values. Instead of optimizing the expected value, the decision maker should optimize the user satisfaction, i.e., the expected utility value. Formally, let $\mathcal{F}$ be the set of feasible solutions to an optimization problem. Each solution $S \in \mathcal{F}$ is associated with a random weight $w(S)$. For instance, $\mathcal{F}$ could be a set of lotteries and $w(S)$ is the (random) payoff of lottery S. Assume the risk awareness of a user can be captured by a utility function $\mu : \mathbb{R} \to \mathbb{R}$: the user obtains $\mu(x)$ units of utility if the outcome is x, i.e., $w(S) = x$. The *expected utility maximization principle* (EUMP) simply suggests that the most desirable solution S is the one that maximizes the expected utility, i.e.,

$$S = \arg\max_{S' \in \mathcal{F}} \mathbb{E}[\mu(w(S'))]$$

[6]Daniel Bernoulli also developed many ideas, such as risk aversion and utility, in his work *Specimen theoriae novae de mensura sortis (Exposition of a New Theory on the Measurement of Risk)* in 1738 [22].

The theory is well known to be versatile in expressing diverse risk-averse or risk-prone behaviors. The theory plays a key role in this dissertation, both in explaining existing semantics and in formulating new problems. Let us see a simple example explaining how the expected utility theory captures diverse risk aware behaviors.

Example 5. *We recall the lottery example in Section 1.2. In* L_1*, the player could win* 1000 *dollars with probability* 1.0*, while in* L_2 *the player could win* 2000 *dollars with probability* 0.5 *and* 0 *dollars otherwise. Both lotteries have the same expected payoff and therefore cannot be distinguished by the expected value criterion. Now, we pick a concave utility function, e.g.,* $\mu(x) = \ln x$*, which is typically used to capture the risk-averse behavior. We can easily see that* $\mathbb{E}[\mu(L_1)] \geq \mathbb{E}[\mu(L_2)]$ *(This can be also seen from Jensen's inequality:* $\mathbb{E}[f[X]] \geq f(\mathbb{E}[X])$ *for any concave* f*). This result matches our intuition that a risk averse player would choose* L_1*. Conversely, we can see* L_2 *is preferable under any convex utility function, which is typically associated with risk-prone behaviors.*

Chapter 3

Related Work

We begin with discussing work in managing uncertain data and probabilistic databases in a broad sense, and then discuss the prior work in ranking and top-k query processing over probabilistic data. Next, we discuss related work in stochastic combinatorial optimization, in particular stochastic shortest path, stochastic knapsack and stochastic matching. We also briefly mention some other work that is conceptually or technically related to the thesis.

Probabilistic Databases: There has been much work on managing probabilistic, uncertain, incomplete, and/or fuzzy data in database systems and this area has received renewed attention in the last few years (see e.g. [101, 84, 20, 122, 71, 45, 54, 157, 180, 12, 178, 120]). The seminal work of Imielinski and Lipski [101] initiated the study of incomplete database and proposed the notation of c-tables. Earlier work on managing probabilistic data includes PDM [20], ProbView [122] and PRA [71], to name a few. With a rapid increase in the number of application domains where uncertain data arises naturally, such as data integration, information extraction, sensor networks, pervasive computing etc., this area has seen renewed interest recently [75]. This work has spanned a range of issues from theoretical development of data models and data languages to practical implementation issues such as indexing techniques, and several research efforts are underway to build systems to manage uncertain data (e.g. MYSTIQ [54], Trio [180], ORION [45], MayBMS [120], PrDB [161], MCDB [103]). Much of this work has used probabilistic methods as the underlying foundation for representing uncertainty, where the uncertainty is encoded in the form of probabilities and the operations on the uncertainty itself are done in accordance with the laws of probability theory.

For efficient query evaluation over probabilistic databases, one of the key results is the dichotomy of conjunctive query evaluation on tuple-independent probabilistic databases by Dalvi and Suciu [54,55]. Briefly the result states that the complexity of evaluating a conjunctive query over tuple-independent probabilistic databases is either PTIME or #P-complete. For the former case, Dalvi and Suciu [54] also present an algorithm to find what are called *safe query plans*, that permit correct *extensional* evaluation of the query. Recently, Dalvi, Schnaitter and Suciu [53] have extended the dichotomy results to the union of conjunctive queries. There has also been much work on efficiently answering other types of queries over probabilistic databases, including aggregates [153, 105], summarization [50], clustering [51, 88], nearest neighbors [121, 42, 23, 44, 148], skyline queries [147, 14] and so on.

Succinct Probability Correlation Models: Those probabilistic databases differ further based on whether they consider correlations or not. Most work has either assumed independence or used BID model which can only capture mutual exclusion [71, 54]. Some work has restricted the correlations that can be modeled [122, 11]. More recently, several approaches have been presented that allow succinct representation of more complex correlations, such as the probabilistic *c*-tables [86, 12] and the graphical models [159, 178, 161]. Several advanced indexing, inferencing techniques have been developed to enhance the performance of query processing on these models (see e.g. [160, 111, 112, 143, 179]).

Ranking and Top-k Query Processing over Probabilistic Databases: The area of ranking and top-k query processing has also seen much work in databases (see Ilyas et al. [100] for a survey). More recently, several researchers have considered top-k query processing in probabilistic databases. Soliman et al. [167] defined the problem of ranking over probabilistic databases, and proposed two ranking functions, U-Top *and* U-Rank, to combine tuple scores and probabilities. Yi et al. [182] present improved algorithms for the same ranking functions. Ming Hua et al. [98] recently presented a different approach called *probabilistic threshold queries*

*(*PT(h)*)*. Zhang and Chomicki [183] present a desiderata for ranking functions and a variant of the probabilistic threshold queries, called *global top-k queries*. Cormode et al. [48] also present a semantics of ranking functions and a new ranking function called *expected rank*. Ge et al. [76] propose the *typical top-k queries* to capture the score distributions. We have reviewed those ranking functions in detail in Section 2.4. Some of the above ranking/top-k queries have been considered in more general settings. Chen et al. [39] develop a dynamic data structure to answer online U-Top queries. Jin et al. [107] present a framework that answers U-Top, U-Rank and PT(h) in uncertain data streams. Li et al. [123] consider the problem of computing expected rank in a distributed setting. Chang et al. [37] study the case where the scores and probabilities are not stored in the same relation and their goal is to reduce the join cost. There has also been work on top-k query processing in probabilistic databases where the ranking is by the result tuple *probabilities* (i.e., probability and score are identical) [151]. The main challenge in that work is efficient computation of the probabilities, whereas we assume that the probability and score are either given or can be computed easily. Patil et al. [146] propose a fully dynamic data structure that can answer PRF$^{\mathsf{e}}$ query online. Their algorithm takes $O(n \log n)$ time to rank the tuples and $O(\log n)$ time for updating a tuple.

Continuous Distributions: Many probabilistic systems support continuous attribute distributions, such as ORION [45], Trio [5], PODS [175] and MCDB [103]. While query processing in ORION, Trio and PODS deals with continuous distributions directly, the MCDB system adopts the Monte-Carlo approach that simulates continuous distribution using samples. As a well known fact, the naive sampling approach is typically expensive at simulating events with tiny probabilities. We will encounter a similar situation in Section 6.2.2 and 6.5 where the Monte-Carlo method needs many samples to achieve a good error bound and to separate those tuples whose PRF scores are very close. Recently, Arumugam et al. have extended the MCDB system with a Gibbs sampler which is more capable of sampling from

the small tail of a query-result distribution [13]. Soliman and Ilyas [168] are 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 Recently, Soliman et al. [166] consider the problem of ranking with uncertain scoring functions where the scoring function (a weight vector $(w_1, \ldots, w_d)$) is assumed to be a uniformly distributed random vector in the d-dimensional simplex $\{\mathbf{w} \mid \sum_{i=1}^{d} w_i = 1\}$. They also study the sensitivity of the ranking results to the refinements of the scores functions made by the user.

Aggregating Inconsistent Information: The problem of aggregating inconsistent information from different sources arises in numerous disciplines and has been studied in different contexts over decades. Specifically, the RANK-AGGREGATION problem aims at combining k different complete ranked lists $\tau_1, \ldots, \tau_k$ on the same set of objects into a single ranking, which is the best description of the combined preferences in the given lists. This problem was considered as early as the 18th century when Condorcet and Borda proposed a voting system for elections [46,34]. In the late 50's, Kemeny proposed the first mathematical criterion for choosing the best ranking [116]. Namely, the Kemeny optimal aggregation τ is the ranking that minimizes $\sum_{i=1}^{k} \mathsf{dis}(\tau, \tau_i)$, where $\mathsf{dis}(\tau_i, \tau_j)$ is the Kendall's tau distance. While computing the Kemeny optimal is shown to be NP-hard [65], 2-approximation can be easily achieved by picking the best ranking from k given ranking lists. The other well-known 2-approximation is from the fact the Spearman footrule distance, defined to be $\mathsf{dis}_F(\tau_i, \tau_j) = \sum_t |\tau_i(t) - \tau_j(t)|$, is within twice the Kendall's tau distance and the footrule aggregation can be done optimally in polynomial time [64]. We refer the readers to [97] for a survey on the early development this problem. Recently Ailon et al. [8] improved the approximation ratio to 4/3. For aggregating top-k answers, Ailon [7] obtained an 3/2-approximation based on rounding an LP solu-

tion. In parallel to our work, Soliman et al. [168] also observed the relationship between ranking in uncertain databases and the RANK-AGGREGATION problem and proposed a polynomial time algorithm under Spearman's footrule distance for full rankings. The CONSENSUS-CLUSTERING problem asks for the best clustering of a set of elements which minimizes the number of pairwise disagreements with the given k clusterings. It is known to be NP-hard [177] and a 2-approximation can also be obtained by picking the best one from the given k clusterings. The best known approximation ratio is 4/3 [8].

Stochastic Optimization: The study of stochastic optimization can be dated back to the work of Dantzig [56] in the 1950's. The focus of the work is mainly about the stochastic versions of various mathematical optimization problems, such as linear programming and convex programming. Since then, this direction has been followed in many research communities in operation research, industrial engineering and management science and developed into a whole field also known as *stochastic programming* (see e.g., [31]).

In recent years stochastic optimization problems have drawn much attention from the computer science community and stochastic versions of many classical combinatorial optimization problems have been studied. In particular, a significant portion of the efforts has been devoted to the two-stage stochastic optimization problem. In such a problem, in a first stage, we are given probabilistic information about the input but the cost of selecting an item is low; in a second stage, the actual input is revealed but the costs for the elements are higher. We are asked to make decision after each stage and minimize the expected cost. Some general techniques have been developed [93, 163]. We refer interested reader to [172] for a comprehensive survey. Another widely studied type of problems considers designing *adaptive* probing policies for stochastic optimization problems where the existence or the exact weight of an element can be only known upon a probe. There is typically a budget for the number of probes (see e.g., [87, 43]), or we require an irrevocable

decision whether to include the probed element in the solution right after the probe (see e.g., [59, 58, 40, 16, 58, 29]). However, most of those works focus on optimizing the expected value of the solution. There is also sporadic work on optimizing the overflow probability or some other objectives subject to the overflow probability constraints. In particular, a few recent works have explicitly motivated such objectives as a way to capture the risk-averse type of behaviors [6, 139, 171]. Besides those works, there has been little work on optimizing more general utility functions for combinatorial stochastic optimization problems from an approximation algorithms perspective.

Stochastic Shortest Path: The most related work to our utility maximization problem under uncertainty is the stochastic shortest path problem (Stoch-SP), which was also the initial motivation for this work. The problem has been studied extensively for several special utility functions in operation research community. Sigal et al. [164] studied the problem of finding the path with greatest probability of being the shortest path. Loui [129] showed that Stoch-SP reduces to the shortest path (and sometimes longest path) problem if the utility function is linear or exponential. Nikolova et al. [140] identified more specific utility and distribution combinations that can be solved optimally in polynomial time. Much work considered dealing with more general utility functions, such as piecewise linear or concave functions, e.g., [137, 138, 21]. However, these algorithms are essentially heuristics and the worst case running times are still exponential. Nikolova et al. [141] studied the problem of maximizing the probability that the length of the chosen path is less than some given parameter. Besides the result we mentioned before, they also considered Poisson and exponential distributions. Despite much effort on this problem, no algorithm is known to run in polynomial time and have provable performance guarantees, especially for more general utility functions or more general distributions. This is perhaps because the hardness comes from different sources, as also noted in [141]: the shortest path selection per se is combinatorial; the dis-

tribution of the length of a path is the convolution of the distributions of its edges; the objective is nonlinear; to list a few.

Stochastic Knapsack: Kleinberg et al. [119] first considered the stochastic knapsack problem with Bernoulli-type distributions and provided a polynomial-time $O(\log 1/\gamma)$ approximation where γ is the given overflow probability. For item sizes with exponential distributions, Goel and Indyk [79] provided a bi-criterion PTAS, and for Bernoulli-distributed items they gave a quasi-polynomial approximation scheme. Chekuri and Khanna [38] pointed out that a PTAS can be obtained for the Bernoulli case using their techniques for the multiple knapsack problem. Goyal and Ravi [82] showed a PTAS for Gaussian distributed sizes. Quite recently, Bhalgat et al. [29] developed a general discretizaton technique that reduces the distributions to a small number of equivalent classes which we can efficiently enumerate for both adaptive and nonadaptive versions of stochastic knapsack. They used this technique to obtain improved results for several variants of stochastic knapsack, notably a bi-criterion PTAS for the *adaptive version* of the problem. Dean at al. [59] gave the first constant approximation for the adaptive version of stochastic knapsack. The adaptive version of stochastic multidimensional knapsack (or equivalently stochastic packing) has been considered in [58, 29] where constant approximations and a bi-criterion PTAS were developed.

Stochastic and Online Matching: The online bipartite matching problem was first studied in the seminal paper by Karp et al. [114] and an optimal $1 - 1/e$ competitive online algorithm was obtained. Katriel et al. [115] considered the two-stage stochastic min-cost matching problem. In their model, we are given in a first stage probabilistic information about the graph and the cost of the edges is low; in a second stage, the actual graph is revealed but the costs are higher. The original online stochastic matching problem was studied recently by Feldman et al. [68]. They gave a 0.67-competitive algorithm, beating the optimal $1 - 1/e$-competitiveness known for worst-case models [114, 109, 133, 32, 80]. Recently, some

improved bounds on this model were obtained [15, 131]. Our model for stochastic matching differs from that in that we have a bound on the number of items each incoming buyer sees, that each edge is only present with some probability, and that the buyer scans the list linearly (until she times out) and buys the first item she likes.

Our stochastic matching problem is also related to the Adwords problem [133], which has applications to sponsored search auctions. The problem can be modeled as a bipartite matching problem as follows. We want to assign every vertex (a query word) on one side to a vertex (a bidder) on the other side. Each edge has a weight, and there is a budget on each bidder representing the upper bound on the total weight of edges that may be assigned to it. The objective is to maximize the total revenue. The stochastic version in which query words arrive according to some known probability distribution has also been studied [130].

The idea of using LP to bound the value of the optimal adaptive policy has been applied to the stochastic knapsack problems (Dean et al. [59, 58]) and multi-armed bandits (see [89, 90] and references therein). Also related is some recent work [30] on budget constrained auctions, which uses similar LP rounding ideas.

k-Set Packing: The k-set packing problem is a generalization of the maximum matching problem (see the definition in Section 9.2). For the k-set packing problem, it is known that the simply greedy algorithm provides a k-approximation and an improvement in the ratio, to $\frac{k}{2}$ can be obtained by a local search heuristic [99], which is also the best known approximation to date. Recently, $O(k)$-approximations were obtained for the more general k-column sparse packing problem (the entries of the matrix can be arbitrary positive numbers rather than just 0/1) [18]. It is also known that the k-set packing problem cannot be efficiently approximated to within a factor of $\Omega(\frac{k}{\ln k})$ unless $\mathsf{P} = \mathsf{NP}$ [94]. This is also a lower bound for our stochastic k-set packing problem. Additionally for LP-based approaches (as in this paper) k-set packing has an integrality gap of $k - 1 + \frac{1}{k}$ [72].

Approximating Functions using Exponential Sums: There is a large volume of work on approximating functions using short exponential sums over a bounded domain, e.g., [144,24,25,27]. In Chapter 8, we will develop a generic algorithm that takes such an algorithm as a subroutine and approximates the utility function in the infinite domain $[0, +\infty)$. Some works also consider using linear combinations of Gaussians or other *kernels* to approximate functions with finite support over the entire real axis $(-\infty, +\infty)$ [41]. This is however impossible using exponentials since α^x is either periodic (if $|\alpha| = 1$) or approaches to infinity when $x \to +\infty$ or $x \to -\infty$ (if $|\alpha| \neq 1$).

Chapter 4

Ranking over Probabilistic Datasets

In this chapter, we begin with comparing the prior work on top-k query processing in probabilistic databsets. We argue that a single specific ranking function may not be sufficient to capture the intricacies of ranking with uncertainty. We then define our parameterized ranking functions (PRF) in Section 4.3 and show it generalizes many prior ranking functions. Moreover, we suggest another perspective to view the top-k queries in probabilistic databases and propose the notion of CON in Section 4.4. Finally, we show that both PRF and CON can be unified and explained by the expected utility maximization principle (EUMP) in Section 4.5. Algorithms for evaluating PRF and CON and their relationships will be discussed in Chapter 5, 6 and 7.

4.1 Comparing Ranking Functions

In this section, we compare the prior semantics on top-k query processing on probabilistic datasets. The formal definitions can be found in Section 2.4. We note that TYP-Top is not really a ranking or top-k query since its answer contains more than one top-k rank list. Therefore, we will not compare TYP-Top with other ranking/top-k semantics. We compared the top-100 answers returned by the five ranking functions with each other using the normalized Kendall distance, for two datasets with 100,000 independent tuples each (see Section 5.3 for a description of the datasets). Table 4.1 shows the results of this experiment. As we can see, the five ranking functions return wildly different top-k answers for the two datasets, with no obvious trends. For the first dataset, E-Rank behaves very differently from

all other functions, whereas for the second dataset, E-Rank happens to be quite close to E-Score. However both of them deviate largely from U-Top, PT(h) and U-Rank. The behavior of E-Score is very sensitive to the dataset, especially the score distribution: it is close to PT(h) and U-Rank for the first dataset, but far away from all of them in the second dataset (by looking into the results, it shares less than 15 tuples with the Top-100 answers of the others). We observed similar behavior for other datasets, and for datasets with correlations.

	E-Score	PT(100)	U-Rank	E-Rank	U-Top
E-Score	–	0.1241	0.3027	0.7992	0.2760
PT(100)	0.1241	–	0.3324	0.9290	0.3674
U-Rank	0.3027	0.3324	–	0.9293	0.2046
E-Rank	0.7992	0.9290	0.9293	–	0.9456
U-Top	0.2760	0.3674	0.2046	0.9456	–

IIP-100,000 ($k = 100$)

	E-Score	PT(100)	U-Rank	E-Rank	U-Top
E-Score	–	0.8642	0.8902	0.0044	0.9258
PT(100)	0.8642	–	0.3950	0.8647	0.5791
U-Rank	0.8902	0.3950	–	0.8907	0.3160
E-Rank	0.0044	0.8647	0.8907	–	0.9263
U-Top	0.9258	0.5791	0.3160	0.9263	–

Syn-IND Dataset with 100,00 tuples ($k = 100$)

Table 4.1: Normalized Kendall distance among various ranking functions for two datasets

This simple experiment illustrates the issues with ranking in probabilistic databases – although several of these definitions seem natural, the wildly different answers they return indicate that none of them may be the "right" definition.

We also observe that in large datasets, E-Rank tends to give very high priority to a tuple with a high probability even if it has a low score. In our synthetic dataset Syn-IND-100,000 with expected size ≈ 50000, t_2 (the tuple with 2nd highest score) has probability approximately 0.98 and t_{1000} (the tuple with 1000th highest score) has probability 0.99. The expected ranks of t_2 and t_{1000} are approximately 10000 and 6000 respectively, and hence t_{1000} is ranked above t_2 even though t_{1000} is only

slightly more probable.

As mentioned above, the original U-Rank function may return the same tuple at different ranks (also observed by the authors [167]), which is usually undesirable. This problem becomes even severe when the dataset and k are both large. For example, in Syn-IND-100,000, the same tuple is ranked at positions 67895 to 100000. In the table, we show a slightly modified version of U-Rank to enforce distinct tuples in the answer (by not choosing a tuple at a position if it is already chosen at a higher position).

4.2 Overview of Our Approach

Before formally defining our ranking functions, we give a high-level overview of our approach to ranking over uncertain datasets. We begin with a systematic exploration of the aforementioned issues by recognizing that ranking in probabilistic databases is inherently a multi-criteria optimization problem, and by deriving a set of *features*, the key properties of a probabilistic dataset that influence the ranked result. We empirically illustrate the diverse and conflicting behavior of several natural ranking functions, and argue that a single specific ranking function may not be appropriate to rank different uncertain databases that we may encounter in practice. Furthermore, different users may weigh the features differently, resulting in different rankings over the same dataset. We then define a general and powerful ranking function, called PRF, that allows us to explore the space of possible ranking functions. We discuss its relationship to previously proposed ranking functions, and also identify two specific parameterized ranking functions, called PRF^ω and PRF^e, as being interesting. The PRF^ω ranking function is essentially a linear weighted ranking function that resembles the scoring functions typically used in information retrieval, web search, data integration, keyword query answering etc. [96, 108, 35, 60, 173]. We observe that PRF^ω may not be suitable for ranking large datasets due to its high running time, and instead propose PRF^e, which uses

a single parameter and can effectively approximate previously proposed ranking functions for probabilistic databases.

We then develop novel algorithms for evaluate PRF functions for a number of probabilistic data models. First we focus on tuple uncertainty and discrete attribute uncertainty models and propose efficient algorithms based on *generating functions* to efficiently rank the tuples using any PRF ranking function. Our algorithms can handle a probabilistic dataset with arbitrary correlations modeled by Markov networks; however, it is particularly efficient when the probabilistic database contains only *mutual exclusivity* and/or *mutual co-existence* correlations, modeled by *probabilistic and/xor trees*. The probabilistic and/xor tree model significantly generalizes previous probabilistic database models like x-tuples and block-independent disjoint models, and is of independent interest. We also consider the continuous attribute uncertainty models which arises naturally in many domains. In many of the applications discussed in the introduction, the attributes of interest are associated with continuous probability distributions. We systematically 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 are important statistics and have direct applications in information retrieval (See e.g. [174, 91]) and sensor networks (See e.g. [165]).

Lastly, we consider the more general problem of combining the results for all possible worlds in a systematic way by proposing the notion of *consensus answers*. Roughly speaking, the most consensus answer is a answer that is *closest in expectation* to the answers of the possible worlds. The "closeness" is measured via a suitably defined distance function between answers. We use this notion to reexamine the top-k queries over probabilistic databases under a variety of distance functions and discover a close relationship between the consensus top-k answers and PRF functions. This relationship further helps to justify the validity of the

semantics of PRF.

4.2.1 Our Contributions

We summarize our technical contributions as follows.

1. (The remaining part of this chapter) We formally define PRF and show it generalizes many prior ranking functions. Then, we define the notion of consensus answers (CON). Moreover, we show that both PRF and CON can be unified and explained by the expected utility maximization principle (EUMP). From the EUMP viewpoint, we obtain a classification of top-k query semantics on probabilistic datasets.

2. (Chapter 5) We focus on PRF computation with discrete probability distributions. We first present novel algorithms based on *generating functions* that enable highly efficient processing of top-k queries over very large probabilistic datasets. Our key algorithm is an $O(n \log(n))$ algorithm for ranking using a PRF^e function over low-correlation datasets (specifically, constant height probabilistic and/xor trees). The algorithm runs in $O(n)$ time if the dataset is pre-sorted by score (Section 5.1). Our algorithms apply to some of the previously proposed ranking functions as well (one of our results was also independently obtained by Yi et al. [182]). We also develop a novel, DFT-based algorithm for approximating an arbitrary weighted ranking function using a linear combination of PRF^e functions (Section 5.2.1). In case users do not know which ranking function should be used, we propose algorithms to learn the parameters for PRF^ω and PRF^e (Section 5.2.2). Moreover, we present a polynomial time algorithm for computing the PRF answers for a correlated dataset, where the correlations are represented using a Markov chain (Section 5.4). Then, we generalize the algorithm to handle correlations modeled by a bounded-treewidth Markov network. The results in this chapter 5 is mainly based on [127, 128]. Some results in Table 5.1 improve those

in [128].

3. (Chapter 6) We consider the problem of computing PRF with continuous distributions. We first present polynomial time exact algorithms for computing rank distributions for uniform and piecewise polynomial distributions based on an extension of the previous generating function technique (Section 6.1). 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. In particular, our algorithms are capable of computing the *positional probabilities*, i.e., the probability that a tuple is ranked at a particular position. We also present theoretical analyses comparing the spline technique to two popular choices, the discretization method and the *Monte Carlo* method (Section 6.2.2). For approximate computation of $\mathsf{PRF}^{\mathsf{e}}$ function values for arbitrary density functions, we propose using Legendre-Gauss Quadrature, which is much faster and more accurate (Section 6.2.3). Furthermore, we present polynomial algorithms for computing PRF^{ℓ} and E-Rank for several important continuous distributions, such as uniform, Gaussian and exponential distributions (Section 6.3). We also present an application of our algorithm to a version of the *probabilistic k-nearest-neighbor* problem. The results in this chapter 6 appear in [125].

4. (Chapter 7) We propose the notion of a *consensus answer*, which is the answer that is *closest in expectation* to the answers of the possible worlds. We develop polynomial time algorithms for computing consensus top-k (CON) answers under various metrics, such as the Symmetric difference metric, *intersection metric* and *generalized Spearman's footrule distance* [66]. We also show that there a close relationship between CON and PRF. We also consider some other types of queries (Section 7.3). In particular, for queries returning a set of tuples, we present polynomial time algorithm for the *symmetric dif-*

ference and the *Jaccard distance* metric for and/xor tree models. For datasets with more complicated correlations (generated from SPJ queries over independent base tables), we show computing consensus answers is NP-hard. We also consider the "group by count" queries and the clustering problem and show constant approximations for them. The results in this chapter 7 appear in [124].

In the next two sections, we formally define the parameterized ranking functions and the consensus answers.

4.3 Parameterized Ranking Functions (PRF)

Ranking in uncertain databases is inherently a multi-criteria optimization problem, and it is not always clear how to rank two tuples that dominate each other along different axes. Consider a database with two tuples t_1 (score = 100, $\Pr(t_1) = 0.5$), and t_2 (score = 50, $\Pr(t_2) = 1.0$). Even in this simple case, it is not clear whether to rank t_1 above t_2 or vice versa. This is an instance of the classic risk-reward tradeoff, and the choice between these two options largely depends on the application domain and/or user preferences.

We propose to follow the traditional approach to dealing with such tradeoffs, by identifying a set of *features*, by defining a parameterized ranking function over these features, and by learning the parameters (weights) themselves using user preferences [96, 108, 35, 60]. To achieve this, we propose a family of ranking functions, parameterized by one or more parameters, and design algorithms to efficiently find the top-k answer according to any ranking function from these families. Our general ranking function, PRF, directly subsumes some of the previously proposed ranking functions, and can also be used to approximate other ranking functions. Moreover, the parameters can be learned from user preferences, which allows us to adapt to different scenarios and different application domains.

Features Although it is tempting to use the tuple probability and the tuple score

$\Pr(r(t_i) = j)$	Positional prob. of t_i being ranked at position j
$\Pr(r(t_i))$	Rank distribution of t_i
PRF	Parameterized ranking function $\Upsilon_\omega(t) = \sum_{i>0} \omega(t,i) \Pr(r(t) = i)$
$\mathsf{PRF}^\omega(h)$	Special case of PRF: $\omega(t,i) = w_i$, $w_i = 0, \forall i > h$
$\mathsf{PRF}^\mathsf{e}(\alpha)$	Special case of PRF^ω: $w_i = \alpha^i, \alpha \in \mathbb{C}$
PRF^ℓ	Special case of PRF^ω: $w_i = -i$
$\delta(p)$	Delta function: $\delta(p) = 1$ if p is true, $\delta(p) = 0$ o.w.

Table 4.2: Notation

as the features, a ranking function based on just those two will be highly sensitive to the actual values of the scores; further, such a ranking function will be insensitive to the correlations in the database, and hence cannot capture the rich interactions between ranking and possible worlds.

Instead we propose to use the positional probabilities as the features: for each tuple t, we have n features,

$$\Pr(r(t) = i), i = 1, \cdots, n,$$

where n is the number of tuples in the database and $r(t)$ is the rank of t. This set of features succinctly captures the possible worlds. Further, correlations among tuples, if any, are naturally accounted for when computing the features. We note that in most cases, we do not explicitly compute all the features, and instead design algorithms that can directly compute the value of the overall ranking function.

Ranking Functions

Next we define a general ranking function which allows exploring the trade-offs discussed above. We use $r_{pw}(t)$ to denote the rank of t in possible world pw.

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:

$$\begin{aligned}\Upsilon_\omega(t) &= \sum_{pw:t\in pw} \omega(t, r_{pw}(t)) \cdot \Pr(pw) \\ &= \sum_{pw:t\in pw} \sum_{i>0} \omega(t,i) \Pr(pw \wedge r_{pw}(t) = i) \\ &= \sum_{i>0} \omega(t,i) \cdot \Pr(r(t) = i).\end{aligned}$$

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

In most cases, ω is a real positive function and we just need to find the k tuples with highest Υ_ω values. However we allow ω to be a complex function in order to approximate other functions efficiently (see Section 5.2.1). Depending on the actual function ω, we get different ranking functions with diverse behaviors. Before discussing the relationship to prior ranking functions, we define two special cases.

$\underline{\mathsf{PRF}^\omega(h)}$ One important class of ranking functions is when $\omega(t,i) = w_i$ (i.e., independent of t) and $w_i = 0\ \forall i > h$ for some positive integer h (typically $h \ll n$). This forms one of prevalent classes of ranking functions used in domains such as information retrieval and machine learning, with the weights typically learned from user preferences [96, 108, 35, 60]. Also, the weight function $\omega(i) = \frac{\ln 2}{\ln(i+1)}$ (called *discount factor*) is often used in the context of ranking documents in information retrieval [104].

$\underline{\mathsf{PRF}^\mathsf{e}(\alpha)}$ This is a special case of $\mathsf{PRF}^\omega(h)$ where $w_i = \omega(i) = \alpha^i$, where α is a constant and may be a real or a complex number. Here $h = n$ (no weights are 0). Typically we expect $|\alpha| \leq 1$, otherwise we have the counterintuitive behavior that tuples with lower scores are preferred.

PRF^ω and PRF^e form the two parameterized ranking functions that we propose in this work. Although PRF^ω is the more natural ranking function and has been used elsewhere, PRF^e is more suitable for ranking in probabilistic databases for various reasons. First, the features as we have defined above are not completely arbitrary,

and the features $\Pr(r(t) = i)$ for small i are clearly more important than the ones for large i. Hence in most cases we would like the weight function, $\omega(i)$, to be monotonically non-increasing. PRF^e naturally captures this behavior (as long as $|\alpha| \leq 1$). More importantly, we can compute the PRF^e function in $O(n \log(n))$ time ($O(n)$ time if the dataset is pre-sorted by score) even for datasets with low degrees of correlations (i.e., modeled by and/xor trees with low heights). This makes it significantly more attractive for ranking over large datasets.

Furthermore, ranking by PRF$^e(\alpha)$, with suitably chosen α, can approximate rankings by many other functions reasonably well even with only real α. Finally, a linear combination of exponential functions, with complex bases, is known to be very expressive in representing other functions [26]. We make use of this fact to approximate many ranking functions by linear combinations of a small number of PRF^e functions, thus significantly speeding up the running time (Section 5.2.1).

Relationship to other ranking functions We illustrate some of the choices of weight function, and relate them to prior ranking functions[1]. We omit the subscript ω if the context is clear. Let $\delta(p)$ denote a delta function where p is a boolean predicate: $\delta(p) = 1$ if $p = true$, and $\delta(p) = 0$ otherwise.

– **Ranking by probabilities:** If $\omega(t, i) = 1$, the result is the set of k tuples with the highest probabilities [151].

– **Expected Score:** By setting $\omega(t, i) = s(t)$, we get the E-Score:

$$\Upsilon(t) = \sum_{pw: t \in pw} s(t) \Pr(pw) = s(t) \Pr(t) = \mathbb{E}[s(t)]$$

– **Probabilistic Threshold Top-k (PT(h)):** If we choose $\omega(i) = \delta(i \leq h)$, i.e., $\omega(i) = 1$ for $i \leq h$, and $= 0$ otherwise, then we have exactly the answer for

[1]The definition of the U-Top introduced in [167] requires the retrieved k tuples belongs to a valid possible world. However, it is not required in our definition, and hence it is not possible to simulate U-Top using PRF.

PT(h).

– **Uncertain Rank-k (U-Rank):** Let $\omega_j(i) = \delta(i = j)$, for some $1 \leq j \leq k$. We can see the tuple with largest Υ_{ω_j} value is the rank-j answer in U-Rank query [167]. This allows us to compute the U-Rank answer by evaluating $\Upsilon_{\omega_j}(t)$ for all $t \in T$ and $j = 1, \ldots, k$.

– **Expected ranks (E-Rank)**: Let PRF$^\ell$ (PRF linear) be another special case of the PRF$^\omega$ function, where $w_i = \omega(i) = -i$. The PRF$^\ell$ function bears a close similarity to the notion of *expected ranks*. Recall that the expected rank of a tuple t is defined to be:

$$\mathbb{E}[r_{pw}(t)] = \sum_{pw \in PW} \Pr(pw) r_{pw}(t)$$

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) the contribution of possible worlds where t exists:

$$er_1(t) = \sum_{i>0} i \times \Pr(r(t) = i) = -\Upsilon(t)$$

where $\Upsilon(t)$ is the PRF$^\ell$ value of tuple t.[2]

(2) the contribution of worlds where t does not exist:

$$er_2(t) = \sum_{pw: t \notin pw} \Pr(pw)|pw| = (1 - p(t))(\sum_{t_i \neq t} \Pr(t_i \mid t \text{ does not exist}))$$

[2]Note that, in the expected rank approach, we pick the k tuples with the *lowest* expected rank, but in our approach, we choose the tuples with the *highest* PRF function values, hence the negation.

If the tuples are independent of each other, then we have:

$$\sum_{t_i \neq t} \Pr(t_i \mid t \text{ does not exist}) = (C - p(t))$$

Thus, the expected ranks can be computed in the same time as PRF^ℓ in tuple-independent datasets. This term can also be computed efficiently in many other cases, including in datasets where only mutual exclusion correlations are permitted. If the correlations are represented using a probabilistic and/xor tree (see Section 5.1.2) or a low-treewidth graphical model (see Section 5.4), then we can compute this term efficiently as well, thus generalizing the prior algorithms for computing expected ranks.

As we can see, many different ranking functions can be seen as special cases of the general PRF ranking function, supporting our claim that PRF can effectively unify these different approaches to ranking uncertain datasets.

4.4 Consensus Top-k Answers

The notion of *consensus answers* proposed in this thesis is largely inspired by the work in *inconsistent information aggregation*, especially the Rank-Aggregation problem [64, 8], which has been studied extensively in numerous contexts over the last half century. In our context, the set of different query answers returned from possible worlds can be thought as inconsistent information which we need to aggregate to obtain a single representative answer. To the best of our knowledge, this connection between query processing in probabilistic databases and inconsistent information aggregation, though natural, has never been realized before in any formal and mathematical way.

Roughly speaking, the consensus top-k answer (CON) is a top-k answer that is

closest in expectation to the answers of the possible worlds:

$$\tau = \arg\min_{\tau' \in \Omega} \{\mathbb{E}[\mathsf{dis}(\tau', \tau_{pw})]\}.$$

where Ω is the set of all feasible top-k answers and $\mathsf{dis}()$ can be any distance function we discussed in Section 2.5. We further distinguish the notion of *mean answers* and *median answers* based on whether we require each answer in the Ω to be the answer of some possible world, We defer the formal definition to Chapter 7. Computing CON for different distance measures can range from polynomial time solvable to NP-hard. Besides being another ranking function, CON has a close relationship with PRF. These results will be discussed in detail in Section 7.2.

4.5 A Unified Viewpoint via Expected Utility

In this section, we show that two definitions we just introduced, PRF and CON, can be unified and explained by the expected utility maximization principle (EUMP). Indeed, most top-k and ranking definitions we have discussed so far and many optimization problems with probabilistic inputs can be cast in this form with different instantiations of utility functions. We believe that viewing diverse definitions in the unified framework and comparing their corresponding utility functions are crucial for better understanding which aspect each definition is intended to capture, the relationship among many semantics, and further for wisely choosing ranking functions in particular applications.

4.5.1 Viewing Ranking as Maximizing Utility

As we have seen in Section 4.3, the PRF function generalizes quite a few of prior ranking functions and seems to be very expressive due to the flexibility in choosing parameters. A natural question is to ask whether PRF is the "ultimate" class of ranking functions (assuming the scores are the only criterion we use in ranking

and we have access to the probability distributions of the scores). In other words, we want to know whether PRF is able to encompass "all reasonable" definitions by properly setting the weight function. Let us consider the following top-k query considered in [78]:

Definition 3. (Probe-Min) *We are given a set of n independent tuples $t_1, \ldots, t_n$. Tuple t_i has a non-negative random score s_i. For simplicity, we assume the cost of a top-k answer is the smallest score among the k retrieved tuples in a possible world. Our goal is to retrieve k tuples such that the expected cost is minimized.*

It is easy to see that the user prefers tuples with smaller scores in this problem. Can we use PRF function to answer this top-k query? However, the answer is simply "No". In fact, it was shown in [78] that Probe-Min is NP-hard for discrete distributions. To the contrary, we will show the PRF function can be evaluated efficiently in polynomial time. Furthermore, we can see that Probe-Min violates the containment property (proposed in [48]) [3]. Since PRF produces a ranking of all tuples, the containment property obviously holds. A less rigorous, but more intuitive, explanation is that the contributions of the k tuples in a Probe-Min answer towards the user's satisfaction do not add up.

This observation leads us to study the ranking/top-k queries in a more fundamental perspective, that is to directly use the user's satisfaction towards the query answer as the objective to optimize. This leads us to adopt the *expected utility maximization* principle which has been widely used in decision making under uncertainty. Here, we use a slightly different form from the one in Section 1.2. Suppose PW is the set of possible worlds. Let $\mathcal{A}$ be the set of valid answers. For instance, in a top-k query, $\mathcal{A}$ is the family of all subsets of k tuples. The user preference is captured by the utility function $\mu : PW \times \mathcal{A} \to \mathbb{R}$. This means the user could obtain $\mu(A, pw)$ units of utility in possible world $pw \in PW$ if $A \in \mathcal{A}$

[3] The containment property states that the top-k answer should be contained in the top-$k+1$ answer

is the chosen answer. The expected utility maximization principle (EUMP) simply states as follows.

Definition 4. (EUMP) *The most desirable answer A is the answer that maximizes the expected utility, i.e.,*

$$A = \arg_{A' \in \mathcal{A}} \mathbb{E}_{PW}[\mu(A', pw)]$$

In fact, the utility function defined in Section 1.2 can be thought as a special case in which there is a random weight associated with each solution and the utility only depends on the weight in any realization. The EUMP formulation generalizes most existing top-k and ranking definitions and many optimization problems considered in literature. In fact, to show that EUMP is a generalization is quite straightforward for most cases. For completeness and further discussion, we list some problems in the following.

1. CON: Just let $\mu(\tau, pw) = -\mathsf{dis}(\tau, \tau_{pw})$ where τ_{pw} is the top-k answer of possible world pw and dis is the distance measure. It is easy to see that maximizing the expected utility $\mathbb{E}[\mu(\tau, pw)]$ is equivalent to minimizing the expected distance $\mathbb{E}[\mathsf{dis}(\tau, \tau_{pw})]$.

2. PRF : Let $\mu(\tau, pw) = \sum_{t \in \tau} \omega(t, r_{pw}(t))$. We can see that

   $$\mathbb{E}[\mu(\tau, pw)] = \sum_{t \in \tau} \mathbb{E}[\omega(t, r_{pw}(t))] = \sum_{t \in \tau} \sum_{i > 0} \omega(t, r_{pw}(t)) \Pr(r(t) = i) = \sum_{t \in \tau} \Upsilon(t).$$

   Therefore, the set of k tuples with maximum $\Upsilon(t)$ values maximizes the expected utility.

3. U-Top : Recall the query returns the k-tuple set that appears as the top-k answer in most possible worlds (weighted by the probabilities of the worlds). We simply let Let $\mu(\tau, pw) = 1$ if $\tau = \tau_{pw}$, i.e. τ is the top-k answer of pw. We can see that $\mathbb{E}[\mu(\tau, pw)] = \sum_{pw} \Pr(pw)\delta(\tau = \tau_{pw}) = \Pr(\tau = \tau_{pw})$.

4. Probe-Min: (See the problem definition in Section 4.5). It is not hard to see the utility function in the problem is $\mu(\tau, pw) = -\min_{t_i \in \tau} s_i(pw)$ where $s_i(pw)$ is the score of tuple t_i in possible world pw.

5. Generalized Expected Score (GES): This is a generalization of E-Score. We have a weight function $\omega : \mathbb{R} \to \mathbb{R}$. The top-$k$ answer should return the k tuples that maximize $\mathbb{E}[\omega(s(t))]$. It is easy to see the corresponding utility function is $\mu(\tau, pw) = \sum_{t \in \tau} \omega(s(t))$.

6. Volume Maximization (Vol-Max): We are given a set of n edges. The lengths of these edges are positive random variables. We would like to pick k edges to span a k-dimensional rectangle such that the expected volume of the rectangle is maximized. We can easily see the utility of a set of k edges is simply the product of their lengths.

7. Top-k Query with Set Interpretation (Top-SI): In the deterministic setting, the answer to a top-k query is to return k tuples with the smallest total weight. However, in a probabilistic dataset, the weight of the tuples are uncertain. Under the expected utility maximization principle, we can define the top-k semantics, that is to find the k tuples maximizing the expected utility, where the utility function is a function of the total weight of the subset.

8. Probabilistic k-center/median Clustering: We consider the unassigned version in [49,88]: Let $\mathcal{P}$ be a finite metric space. There are n input nodes. The position p_i of node i is an independent random variable over $\mathcal{P}$. The probabilistic k-center (or k-median) objective is to find a set τ of k deterministic points in $\mathcal{P}$ [4] such that $\mathbb{E}\big[\max_i \mathsf{dis}(p_i, \tau)\big]$ (or $\mathbb{E}\big[\sum_i \mathsf{dis}(p_i, \tau)\big]$) is minimized. It is easy to see the utility function in probabilistic k-means (or k-median) is $\mu(\tau, pw) = -\max_i \mathsf{dis}(p_i, \tau)$ (or $\mu(\tau, pw) = -\sum_i \mathsf{dis}(p_i, \tau)$).

[4] For any $a \in \mathcal{P}$ and $S \subset \mathcal{P}$, $\mathsf{dis}(a, S) = \min_{b \in S} \mathsf{dis}(a, b)$.

We would like to remark here that despite the powerful semantic expressibility, EUMP is of little help in computation. This is because the utility function may requires exponential space to store. In some cases, even the utility function has a compact representation, computing the function itself can be NP-hard (e.g., the probabilistic k-center problem). On the other hand, PRF, the subject of the this work, lies in a sweet spot where the computation can be done efficiently and a number of prior definitions can be encompassed.

4.5.2 Distinctions between Between Ranking and Top-k Queries

Top-k answers are often understood as the ranking answers truncated to the first k tuples and two terms are used interchangeably in many prior works. To be more formal, we say a top-k query *reduces* to (or is *consistent* with) a ranking query, if the top-k answer is the length-k prefix of ranking answer for any k. However, we have seen many top-k queries that do not reduce to any ranking queries (e.g., Probe-Min). Therefore, we need to draw a clear distinction between ranking and top-k queries for clarification. From the perspective of EUMP, we would like to ask the following question:

- What property the utility function should satisfy so that the corresponding top-k query reduces to a ranking query?

To answer this, we need the the following notation of *separability*.

Definition 5. *The utility function ω is called* additively separable *(or* multiplicatively separable*) if for any possible world pw, the joint utility of any top-k answer τ is exactly the sum (or product) of the utilities of individual tuples in τ, i.e., $\omega(pw, \tau) = \sum_{t \in \tau} \omega(pw, t)$ (or $\omega(pw, \tau) = \prod_{t \in \tau} \omega(pw, t)$).*

For example, it is not difficult to check that the utility functions corresponding to PRF and Vol-Max are additively separable and multiplicative separable, respec-

tively. If ω is additively separable, we can write

$$\mathbb{E}_{PW}[\omega(pw,\tau)] = \mathbb{E}_{PW}[\sum_{t\in\tau}\omega(pw,t)] = \sum_{t\in\tau}\mathbb{E}_{PW}[\omega(pw,t)].$$

It is easy to see that the expected utility can be maximized by choosing the k tuples with the maximum individual utilities. Equivalently, the values $\mathbb{E}_{PW}[\omega(pw,t)]$ define a ranking of the tuples and the top-k list is simply the length-k truncation of the ranking answer.

In fact, for a multiplicatively separable utility function μ, there is an additively separable $\mu' = \log\mu$ that captures the same preference. This is because log is an increasing function and the answer that maximizes μ also maximizes μ'. The above argument can be easily generalized to the following proposition.

Proposition 4.1. *A top-k query, which maximizes $\mathbb{E}[\mu]$ for some utility function μ, is consistent with some ranking query if there is a monotone increasing function f such that $f(\mu)$ is additively separable.*

However, we note that the converse of the above proposition, that is, for every top-k query that reduces to a ranking query there exists a monotone increasing function f such that $f(\mu)$ is additively separable, is not necessarily true.

4.5.3 A Classification of Top-k Semantics

In this subsection, we provide a classification of the existing top-k semantics based on their corresponding utility functions.

1. Score-based: The utility $\mu(\tau, pw)$ depends only on the scores of the tuples in the top-k answer τ in possible world pw. Such a utility function is relevant if the scores (or some known function of the scores) are linearly associated with the user satisfaction (e.g., a tuple with score 2 is two times better than a tuple with score 1). Such utility functions are typically associated with

monetary applications where the scores are the profits or cost, measured in currency units.

(a) Separable utility function: This class is exactly GES if the utility function is additively separable. Note that E-Score is a special case of GES. Vol-Max also belongs to this class since its utility function is multiplicatively separable. By Proposition 4.1, any top-k query in this class reduces to a ranking query (rank the tuples by $\mathbb{E}[\mu(s(t))]$).

(b) Nonseparable utility function: We can see that Probe-Min and Top-SI belongs to this class. Many problems in this class are NP-hard or even harder. New algorithmic techniques are typically required to answer (or approximate) top-k queries in this class. Probe-Min is studied in [78]. Top-SI will be handled in Section 8.3.1.

2. Position-based: The utility $\mu(\tau, pw)$ depends only on the ranks of the tuples in the top-k answer τ in possible world pw. Such a utility function is preferred if the ranks determines the user satisfaction, or the magnitudes of the scores do not directly correspond to the user satisfaction (or at least we do not know what is the exact correspondence). Such utility functions typically appear in applications such as search engines where the rank of an object is what matters.

(a) Separable utility function: PRF^ω belongs to this class. By Proposition 4.1, any top-k query in this class reduces to a ranking query (rank the tuple by $\Upsilon_\mu(t) = \mathbb{E}[\mu(r(t))]$).

(b) Nonseparable utility function: Many members in CON belong to this class, e.g., CON under Spearman's Footrule or Kendall distance.

A more general class of utility functions may depend on both the scores and positions of the answer. In fact, the general form of PRF corresponds to such a utility function, which is also additively separable. In practical applications, such

utility functions are supposed to be designed by domain experts who are familiar with the interaction between the scores, the ranks and the user satisfaction levels.

As we have already seen, nonseparable utility functions are in general much harder to deal with. In Chapter 8, we develop approximation techniques to optimize expected utility for large class of combinatorial problems where the corresponding utility functions are nonseparable. As one application of the technique, we can approximate the Top-SI query (see Section 8.3.1).

Chapter 5

Computing PRF: Discrete Distributions

In this chapter, we present algorithms for efficiently ranking according to a PRF function when the probability distributions are discrete as well as for learning the weights for a PRF^ω function from the user feedbacks. At the end of the chapter, we conduct a comprehensive experimental study, which demonstrates the effectiveness of our parameterized ranking functions, especially PRF^e, at approximating other ranking functions and the scalability of our proposed algorithms for exact or approximate ranking.

5.1 Computing a PRF function

We first present the basic idea behind our algorithms assuming mutual independence, and then consider correlated tuples with correlations represented using an and/xor tree. We then present a very efficient algorithm for ranking using a PRF^e function, and then briefly discuss how to handle attribute uncertainty.

5.1.1 Assuming Tuple Independence

First we show how the PRF function can be computed in $O(n^2)$ time for a general weight function ω, and for a given set of tuples $T = \{t_1, \ldots, t_n\}$. We assume there is only tuple-level uncertainty. In all our algorithms, we assume that $\omega(t, i)$ can be computed in $O(1)$ time.

Clearly it is sufficient to compute $\Pr(r(t) = j)$ for any tuple t and $1 \leq j \leq n$ in $O(n^2)$ time. Given these values, we can directly compute the values of $\Upsilon(t)$ in $O(n^2)$ time. (Later, we will present several algorithms which run in $O(n)$ or

$O(n\log(n))$ time which combine these two steps for some special ω functions).

We first sort the tuples in a non-increasing order by their scores (which are assumed to be deterministic); assume $t_1, \ldots, t_n$ indicates this sorted order. Suppose now we want to compute $\Pr(r(t_i) = j)$. Let $T_i = \{t_1, t_2, \ldots, t_i\}$ and σ_i be an indicator variable that takes value 1 if t_i is present in a possible world, and 0 otherwise. Further, let $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ denote a vector containing all the indicator variables. Then, we can write $\Pr(r(t_i) = j)$ as follows:

$$\begin{aligned}
\Pr(r(t_i) = j) &= \Pr(t_i) \sum_{pw : |pw \cap T_{i-1}| = j-1} p(pw) \\
&= \Pr(t_i) \sum_{\sigma : \sum_{l=1}^{i-1} \sigma_l = j-1} \prod_{l<i : \sigma_l = 1} \Pr(t_l) \prod_{l<i : \sigma_l = 0} (1 - \Pr(t_l))
\end{aligned}$$

The first equality says that tuple t_i ranks at the jth position if and only if t_i *and* exactly $j-1$ tuples from T_{i-1} are present in the possible world. The second equality is obtained by rewriting the sum to be over the indicator vector (each assignment to the indicator vector corresponds to a possible world), and by exploiting the fact that the tuples are independent of each other. The naive method to evaluate the above formula by explicitly listing all possible worlds needs exponential time. Now, we present a polynomial time algorithm based on generating functions. Consider the following generating function over x:

$$\mathcal{F}(x) = \prod_{i=1}^{n} (a_i + b_i x)$$

The coefficient of x^k in $\mathcal{F}(x)$ is:

$$\sum_{|\beta|=k} \prod_{i:\beta_i=0} a_i \prod_{i:\beta_i=1} b_i$$

where $\beta = \langle \beta_1, \ldots, \beta_n \rangle$ is a boolean vector, and $|\beta|$ denotes the number of 1's in

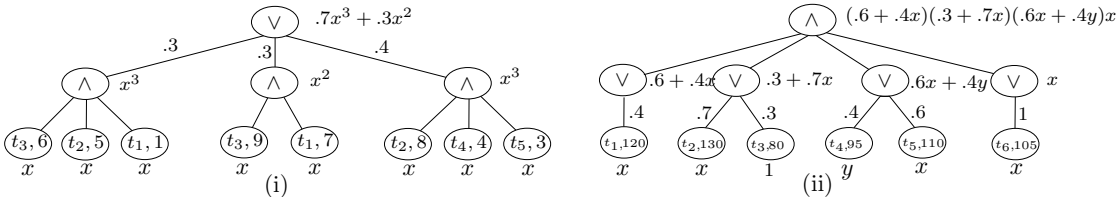


Figure 5.1: PRF computation on and/xor trees: (i) The left figure corresponds to the database in Figure 2.3; the generating function obtained by assigning the same variable x to all leaves gives us the distribution over the sizes of the possible worlds. (ii) The right figure illustrates the construction of the generating function for computing $\Pr(r(t_4) = 3)$ in the and/xor tree in Figure 2.2.

β. Now consider the following generating function:

$$\mathcal{F}^i(x) \quad = \quad \left(\prod_{t \in T_{i-1}} \left(1 - \Pr(t) + \Pr(t) \cdot x\right) \right) \Pr(t_i) \cdot x = \sum_{j \geq 0} c_j x^j.$$

We can see that the coefficient c_j of x^j in the expansion of $\mathcal{F}^i$ is exactly the probability that t_i is at rank j, i.e., $c_j = \Pr(r(t_i) = j)$. We note $\mathcal{F}^i$ contains at most $i + 1$ nonzero terms. We observe this both from the form of $\mathcal{F}^i$ above, and also from the fact that $\Pr(r(t_i) = j) = 0$ if $j > i$. Hence, we can expand $\mathcal{F}^i$ to compute the coefficients in $O(i^2)$ time. This allows us to compute $\Pr(r(t_i) = j)$ for t_i in $O(i^2)$ time; $\Upsilon(t_i)$, in turn, can be written as:

$$\Upsilon(t_i) = \sum_j \omega(t_i, j) \cdot \Pr(r(t_i) = j) = \sum_j \omega(t_i, j) c_j \tag{5.1}$$

which can be computed in $O(i^2)$ time.

Example 6. *Consider a relation with 3 independent tuples t_1, t_2, t_3 (already sorted according to the score function) with existence probabilities $0.5, 0.6, 0.4$, respectively. The generating function for t_3 is:*

$$\mathcal{F}^3(x) = (.5 + .5x)(.4 + .6x)(.4x) = .12x^3 + .2x^2 + .08x$$

This gives us:

Algorithm 1: IND-PRF-RANK(D_T)

```
F^0(x) = 1;
for i = 1 to n do
    F^i(x) = (Pr(t_i)/Pr(t_{i-1})) F^{i-1}(x)(1 − Pr(t_{i-1}) + Pr(t_{i-1})x);
    Expand F^i(x) in the form of Σ_j c_j x^j ;
    Υ(t_i) = Σ_{j=1}^{n} ω(t_i, j) c_j ;
return k tuples with largest Υ values;
```

$$\Pr(r(t_3) = 1) = .08, \Pr(r(t_3) = 2) = .2, \Pr(r(t_3) = 3) = .12$$

If we expand each $\mathcal{F}^i$ for $1 \leq i \leq n$ from scratch, we need $O(n^2)$ time for each $\mathcal{F}^i$ and $O(n^3)$ time in total. However, the expansion of $\mathcal{F}^i$ can be obtained from the expansion of $\mathcal{F}^{i-1}$ in $O(i)$ time by observing that:

$$\mathcal{F}^i(x) = \frac{\Pr(t_i)}{\Pr(t_{i-1})}\mathcal{F}^{i-1}(x)\Big(1 - \Pr(t_{i-1}) + \Pr(t_{i-1})x\Big) \tag{5.2}$$

This trick gives us a $O(n^2)$ time complexity for computing the values of the ranking function for all tuples. See Algorithm 1 for the pseudocode. Note that $O(n^2)$ time is asymptotically optimal in general since the computation involves at least $O(n^2)$ probabilities, namely $\Pr(r(t_i) = j)$ for all $1 \leq i, j \leq n$.

For some specific ω functions, we may be able to achieve faster running time. For $\mathsf{PRF}^\omega(h)$ functions, we only need to expand all $\mathcal{F}^i$'s up to x^h term since $\omega(i) = 0$ for $i > h$. Then, the expansion from $\mathcal{F}^{i-1}(x)$ to $\mathcal{F}^i(x)$ only takes $O(h)$ time. This yields an $O(n \cdot h + n\log(n))$ time algorithm. We note the above technique also gives an $O(nk + n\log(n))$ time algorithm for answering the U-Rank top-k query (all the needed probabilities can be computed in that time), thus matching the best known upper bound by Yi et al. [182] (the original algorithm in [167] runs in $O(n^2k)$ time).

We remark that the generating function technique can be seen as a variant of dynamic programming in some sense; however, using it explicitly in place of the

obscure recursion formula gives us a much cleaner view and allows us to generalize it to handle more complicated tuple correlations. This also leads to an algorithm for extremely efficient evaluation of PRF^e functions (Section 5.1.3).

5.1.2 Probabilistic And/Xor Trees

Next we generalize our algorithm to handle a correlated database where the correlations can be captured using an *and/xor* tree. In fact, many types of probability computations on and/xor trees can be done efficiently and elegantly using generating functions. Here we first provide a general result and then specialize it for PRF computation.

As before, let $T = \{t_1, t_2, \ldots, t_n\}$ denote the tuples sorted in a non-increasing order of their score function, and let $T_i = \{t_1, t_2, \ldots, t_i\}$. Let $\mathcal{T}$ denote the and/xor tree. Suppose $\mathcal{X} = \{x_1, x_2, \ldots\}$ is a set of variables. Define a mapping π which associates each leaf $l \in \mathcal{T}$ with a variable $\pi(l) \in \mathcal{X}$. Let $\mathcal{T}_v$ denote the subtree rooted at v and let $v_1, \ldots, v_h$ be v's children. For each node $v \in \mathcal{T}$, we define a generating function $\mathcal{F}_v(\mathcal{X}) = \mathcal{F}_v(x_1, x_2, \ldots)$ recursively:

- If v is a leaf, $\mathcal{F}_v(\mathcal{X}) = \pi(v)$.
- If v is a $\veebar$ node,

  $\mathcal{F}_v(\mathcal{X}) = (1 - \sum_{l=1}^{h} p_{(v,v_l)}) + \sum_{l=1}^{h} p_{(v,v_l)} \mathcal{F}_{v_l}(\mathcal{X})$
- If v is a $\wedge$ node, $\mathcal{F}_v^i(\mathcal{X}) = \prod_{l=1}^{h} \mathcal{F}_{v_l}(\mathcal{X})$.

The generating function $\mathcal{F}(\mathcal{X})$ for tree $\mathcal{T}$ is the one defined above for the root. It is easy to see, if we have a constant number of variables, the polynomial can be expanded in the form of $\sum_{i_1,i_2,\ldots} c_{i_1,i_2\ldots} x_1^{i_1} x_2^{i_2} \ldots$ in polynomial time.

Now recall that each possible world pw contains a subset of the leaves of $\mathcal{T}$ (as dictated by the $\veebar$ and $\wedge$ nodes). The following theorem characterizes the relationship between the coefficients of $\mathcal{F}$ and the probabilities we are interested in.

Theorem 5.1. *The coefficient of the term $\prod_j x_j^{i_j}$ in $\mathcal{F}(\mathcal{X})$ is the total probability of the possible worlds for which, for all j, there are exactly i_j leaves associated with variable x_j.*

Proof: Suppose $\mathcal{T}$ is rooted at r, $r_1, \ldots, r_h$ are r's children, and $\mathcal{T}_l$ is the subtree rooted at r_l. We denote by S (or S_l) the random set of leaves generated according to model $\mathcal{T}$ (or $\mathcal{T}_l$). We let $\mathcal{F}$ (or $\mathcal{F}_l$) be the generating function corresponding to $\mathcal{T}$ (or $\mathcal{T}_l$). For ease of notation, we use $\mathbf{i}$ to denote index vector $\langle i_1, i_2, \ldots \rangle$, I to denote the set of all such $\mathbf{i}$s and $\mathcal{X}^{\mathbf{i}}$ to denote $\prod_j x_j^{i_j}$. Therefore, we can write $\mathcal{F}(\mathcal{X}) = \sum_{i_1,i_2,\ldots} c_{i_1,i_2\ldots} x_1^{i_1} x_2^{i_2} \ldots = \sum_{\mathbf{i}\in I} c_{\mathbf{i}} \mathcal{X}^{\mathbf{i}}$. We use the notation $S \cong \mathbf{i}$ for some $\mathbf{i} = \langle i_1, i_2, \ldots \rangle \in I$ to denote the event that S contains i_j leaves associated with variable x_j for all j. Given the notations, we need to show $c_{\mathbf{i}} = \Pr(S \cong \mathbf{i})$.

We shall prove by induction on the height of the and/xor tree. We consider two cases. If r is a $\wedge$ node, we know from Definition 1 that $S = \cup_{l=1}^{h} S_l$. First, it is not hard to see that given $S_l \cong \mathbf{i}_l$ for $1 \le l \le h$, the event $S \cong \mathbf{i}$ happens if and only if $\sum_l \mathbf{i}_l = \mathbf{i}$. Therefore,

$$\Pr(S \cong \mathbf{i}) = \sum_{\sum_l \mathbf{i}_l = \mathbf{i}} \prod_{l=1}^{h} \Pr(S_l \cong \mathbf{i}_l). \tag{5.3}$$

Assume $\mathcal{F}_l$ can be written as $\sum_{\mathbf{i}_l} c_{l,\mathbf{i}_l} \mathcal{X}^{\mathbf{i}_l}$. From the construction of the generating function, we know that

$$\begin{aligned} \mathcal{F}(\mathcal{X}) = \prod_{l=1}^{h} \mathcal{F}_l = \prod_{l=1}^{h} \sum_{\mathbf{i}_l \in I} c_{l,\mathbf{i}_l} \mathcal{X}^{\mathbf{i}_l} = \sum_{\mathbf{i}\in I} \Big(\sum_{\sum_l \mathbf{i}_l = \mathbf{i}} \prod_{l=1}^{h} c_{l,\mathbf{i}_l} \mathcal{X}^{\mathbf{i}_l} \Big) \\ = \sum_{\mathbf{i}\in I} \Big(\sum_{\sum_l \mathbf{i}_l = \mathbf{i}} \prod_{l=1}^{h} c_{l,\mathbf{i}_l} \Big) \mathcal{X}^{\mathbf{i}} \end{aligned} \tag{5.4}$$

By induction hypothesis, we have $\Pr(S_l \cong \mathbf{i}_l) = c_{l,\mathbf{i}_l}$ for any l and $\mathbf{i}_l$. Therefore, we can conclude from (5.3) and (5.4) that $\mathcal{F}(\mathcal{X}) = \sum_{\mathbf{i}} \Pr(S \cong \mathbf{i}) \mathcal{X}^{\mathbf{i}}$.

Now let us consider the other case where r is a $\vee$ node. From Definition 1, it

is not hard to see that

$$\Pr(S \cong \mathbf{i}) = \sum_{l=1}^{h} \Pr(S_l = \mathbf{i}) p_{(r,r_l)} \tag{5.5}$$

Moreover, we have

$$\begin{aligned}\mathcal{F}(\mathcal{X}) &= \sum_{l=1}^{h} p_{(r,r_l)} \mathcal{F}_l(\mathcal{X}) = \sum_{l=1}^{h} p_{(r,r_l)} \sum_{\mathbf{i}_l} c_{l,\mathbf{i}_l} \mathcal{X}^{\mathbf{i}_l} \\ &= \sum_{\mathbf{i}} \Big(\sum_{l=1}^{h} p_{(r,r_l)} c_{l,\mathbf{i}} \Big) \mathcal{X}^{\mathbf{i}} = \sum_{\mathbf{i}} \Pr(S \cong \mathbf{i}) \mathcal{X}^{\mathbf{i}}\end{aligned}$$

where the last equality follows from (5.5) and induction hypothesis. This completes the proof. □

We first provide two simple examples to show how to use Theorem 5.1 to compute the probabilities of two events related to the size of the possible world, and then show how to use the same idea to compute $\Pr(r(t) = i)$.

Example 7. *If we associate all leaves with the same variable x, the coefficient of x^i is equal to $\Pr(|pw| = i)$. The above can be used to obtain a distribution on the possible world sizes (Figure 5.1(i)).*

Example 8. *If we associate a subset S of the leaves with variable x, and other leaves with constant 1, the coefficient of x^i is equal to $\Pr(|pw \cap S| = i)$.*

Next we show how to compute $\Pr(r(t_i) = j)$ (i.e., the probability t_i is ranked at position j). Let s denote the score of the tuple. In the and/xor tree $\mathcal{T}$, we associate all leaves with score value larger than s with variable x, the leaf (t_i, s) with variable y, and the rest of leaves with constant 1. Let the resulting generating function be $\mathcal{F}^i$. By Theorem 5.1, the coefficient of $x^{j-1}y$ in the generating function $\mathcal{F}^i$ is exactly $\Pr(r(t_i) = j)$. See Algorithm 2 for the pseudocode of the algorithm.

Example 9. *We consider the database in Figure 2.2. Suppose we want to compute $\Pr(r(t_4) = 3)$. We associate variable x to t_1, t_2, t_5 and t_6 since their scores are*

Algorithm 2: ANDXOR-PRF-RANK($\mathcal{T}$)

$\pi(t_i) \leftarrow 1 \forall i$ {$\pi(t_i)$ is the variable associated to leaf t_i};
for $i = 1$ *to* n **do**
  if $i \neq 1$ **then** $s(t_{i-1}) \leftarrow x$;
  $\pi(t_i) \leftarrow y$;
  $\mathcal{F}^i(x, y) = \mathsf{GENE}(\mathcal{T}_i, \pi)$;
  Expand $\mathcal{F}^i(x, y)$ in the form $\sum_j c'_j x^j + (\sum_j c_j x^{j-1})y$;
  $\Upsilon(t_i) = \sum_{j=1}^{n} \omega(t_i, j) c_j$;
return k tuples with largest Υ values;
Subroutine: $\mathsf{GENE}(\mathcal{T}, \pi)$;
r is the root of tree $\mathcal{T}$;
if $\mathcal{T}$ *is a singleton node* **then**
  return $\pi(r)$;
else
  $\mathcal{T}_i$ is the subtree rooted at r_i for $r_i \in Ch(r)$;
  $p = \sum_{r_i \in Ch(r)} p_{(r,r_i)}$;
  if r *is a* $\otimes$ *node* **then**
    return $1 - p + \sum_{r_i \in Ch(r)} p_{(r,r_i)} \cdot \mathsf{GENE}(\mathcal{T}_i, t)$;
  if r *is a* $\oplus$ *node* **then**
    return $\prod_{r_i \in Ch(r)} \mathsf{GENE}(\mathcal{T}_i, t)$;

larger than t_4's score. We also associate y to t_4 itself and 1 to t_3 whose score is less t_4's. The generating function for the right hand side tree in Figure 5.1 is $(.6 + .4x)(.3 + .7)(.4x + .6y)x = .168x^4 + 0.112x^3y + 0.324x^3 + 0.216x^2y + 0.108x^2 + 0.072xy$. *So we get that* $\Pr(r(t_5) = 3)$ *is the coefficient of x^2y which is* 0.216. *From Figure 2.2, we can also see* $\Pr(r(t_5) = 3) = \Pr(pw_3) + \Pr(pw_5) = .048 + .168 = .216$.

If we expand $\mathcal{F}_v^i$ for each internal node v in a naive way (i.e., we multiply the polynomials one by one), we can show the running time is $O(n^2)$ at each internal node, $O(n^3)$ for each tree $\mathcal{F}^i$ and thus $O(n^4)$ overall. We can use some tricks to improve the running time as follows. First, we can get rid of the variable y as follows. Suppose the generating function is $\mathcal{F}(x, y) = P_1(x) + yP_2(x)$. We can write $\mathcal{F}(x, y)$ in this form because the degree of y is at most 1. So, $P_1(x) = \mathcal{F}(x, 0)$ and $P_2(x) = \mathcal{F}(x, 1) - \mathcal{F}(x, 0)$. Hence, computing $\mathcal{F}(x, y)$ reduces to computing

two uni-variable polynomials $\mathcal{F}(x, 0)$ and $\mathcal{F}(x, 1)$. From now on, we only need to focus on manipulating uni-variable polynomials. In fact, expanding each $\mathcal{F}^i$ can be done in $O(n^2)$ time. We outline two algorithms in Appendix A.1. The total running time is therefore $O(n^3)$. For PRF^ω and and/xor trees with low heights, we can obtain better algorithms. See the details in Section 5.1.5.

5.1.3 Computing a PRF^e Function

Next we present an $O(n \log(n))$ algorithm to evaluate a PRF^e function (the algorithm runs in linear time if the dataset is pre-sorted by score). If $\omega(i) = \alpha^i$, then we observe that:

$$\Upsilon(t_i) = \sum_{j=1}^{n} \Pr(r(t_i) = j)\alpha^j = \mathcal{F}^i(\alpha) \tag{5.6}$$

This surprisingly simple relationship suggests we don't have to expand the polynomials $\mathcal{F}^i(x)$ at all; instead we can evaluate the numerical value of $\mathcal{F}^i(\alpha)$ directly. Again, we note that the value $\mathcal{F}^i(\alpha)$ can be computed from the value of $\mathcal{F}^{i-1}(\alpha)$ in $O(1)$ time using Equation (5.2). Thus, we have $O(n)$ time algorithm to compute $\Upsilon(t_i)$ for all $1 \leq i \leq n$ if the tuples are pre-sorted.

Example 10. *Consider Example 6 and the PRF^e function for t_3. We choose $\omega(i) = .6^i$. Then, we can see that $\mathcal{F}^3(x) = (.5 + .5x)(.4 + .6x)(.4x)$. So, $\Upsilon(t_3) = \mathcal{F}^3(.6) = (.5 + .5 \times .6)(.4 + .6 \times .6)(.4 \times .6) = .14592$.*

We can use a similar idea to speed up the computation if the tuples are correlated and the correlations are represented using an and/xor tree. Let $\mathcal{T}_i$ be the and/xor tree where $\pi(t_j) = x$ for $1 \leq j < i$, $\pi(t_i) = y$ and $\pi(t_j) = 1$ for $j > i$. Suppose the generating function for $\mathcal{T}_i$ is $\mathcal{F}^i(x, y) = \sum_j c'_j x^j + (\sum_j c_j x^{j-1})y$ and $\Upsilon(t_i) = \sum_{j=1}^{n} \alpha^j c_j$. We observe an intriguing relationship between the PRF^e value and the

generating function:

$$\begin{aligned}\Upsilon(t_i) &= \sum_j c_j\alpha^j = \Big(\sum_j c'_j\alpha^j + (\sum_j c_j\alpha^{j-1})\alpha\Big) - \sum_j c'_j\alpha^j \\ &= \mathcal{F}^i(\alpha,\alpha) - \mathcal{F}^i(\alpha,0).\end{aligned}$$

Given this, $\Upsilon(t_i)$ can be computed in linear time by bottom up evaluation of $\mathcal{F}^i(\alpha,\alpha)$ and $\mathcal{F}^i(\alpha,0)$ in $\mathcal{T}^i$. If we simply repeat it n times, once for each t_i, this gives us a $O(n^2)$ total running time.

By carefully sharing the intermediate results among computations of $\Upsilon(t_i)$, we can improve the running time to $O(n\log(n)+nd)$ where d is the height of the and/xor tree. This improved algorithm runs in iterations. Suppose the tuples are already pre-sorted by their scores. Initially, the label of all leaves, i.e., $\pi(t_i)$, is 1. In iteration i, we change the label of leaf t_{i-1} from y to x and the label of t_i from 1 to y. The algorithm maintains the following information in each inner node v: the numerical values of $\mathcal{F}^i_v(\alpha,\alpha)$ and $\mathcal{F}^i_v(\alpha,0)$. The values on node v need to be updated when the value of one of its children changes. Therefore, in each iteration, the computation only happens on the two paths, one from t_{i-1} to the root and one from t_i to the root. Since we update at most $O(d)$ nodes for each iteration, the running time is $O(nd)$. Suppose we want to update the information on the path from t_{i-1} to the root. We first update the $\mathcal{F}^i_v(.,.)$ values for the leaf t_{i-1}. Since $\mathcal{F}^i_{t_{i-1}} = \pi(t_{i-1}) = x$, we have $\mathcal{F}^i_{t_{i-1}}(\alpha,\alpha) = \alpha$ and $\mathcal{F}^i_{t_{i-1}}(\alpha,0) = \alpha$. We assume v's child, say u, just had its values changed. The updating rule for $\mathcal{F}^i_v(.,.)$(both $\mathcal{F}^i_v(\alpha,\alpha)$ and $\mathcal{F}^i_v(\alpha,0)$) in node v is as follows.

1. v is a $\textcircled{\wedge}$ node, then:

$$\mathcal{F}^i_v(.,.) \leftarrow \mathcal{F}^{i-1}_v(.,.)\mathcal{F}^i_u(.,.)/\mathcal{F}^{i-1}_u(.,.) \tag{5.7}$$

2. v is a $\textcircled{\vee}$ node, then:

$$\mathcal{F}_v^i(.,.) \leftarrow \mathcal{F}_v^{i-1}(.,.) + p_{(v,u)}\mathcal{F}_u^i(.,.) - p_{(v,u)}\mathcal{F}_u^{i-1}(.,.) \tag{5.8}$$

The values on other nodes are not affected. The updating rule for the path from t_i to the root is the same except that for the leaf t_i, we have $\mathcal{F}_{t_i}^i(\alpha,\alpha) = \alpha$ and $\mathcal{F}_{t_i}^i(\alpha,0) = 0$ since $\mathcal{F}_{t_i}^i(x,y) = \pi(t_i) = y$. See Algorithm 3 for the psuedo-code. We note that, for the case of *x-tuples*, which can be represented using a two-level tree, this gives us an $O(n\log(n))$ algorithm for ranking according to PRF^e.

Algorithm 3: ANDXOR-PRF^e-RANK($\mathcal{T}$)

```
F_{t_i}(α,α) = 1, F_{t_i}(α,0) = 1, ∀i ;
for i = 1 to n do
    if i ≠ 1 then
        F_{t_{i-1}}(α,α) = α, F_{t_{i-1}}(α,0) = α ;
        UPDATE(T, t_{i-1});
    F_{t_i}(α,α) = α, F_{t_i}(α,0) = 0 ;
    UPDATE(T, t_i);
    Υ(t_i) = F_r(α,α) − F_r(α,0);
return k tuples with largest Υ values;
Subroutine: UPDATE(T, v);
while v is not the root do
    u ← v;
    v ← parent(v);
    if v is a ⊗ node then
        F_v(.,.) ← F_v(.,.)F_u^i(.,.)/F_u(.,.);
    if v is a ⊕ node then
        F_v(.,.) ← F_v(.,.) + p_{(v,u)}F_u(.,.) − p_{(v,u)}F_u(.,.);
```

5.1.4 Attribute Uncertainty or Uncertain Scores

We briefly describe how we can do ranking over tuples with discrete attribute uncertainty where the uncertain attributes are part of the tuple scoring function (if the uncertain attributes do not affect the tuple score, then they can be ignored

for the ranking purposes). More generally, this approach can handle the case when there is a discrete probability distribution over the score of the tuple.

Assume $\sum_j p_{i,j} \leq 1$ for all i. The score s_i of tuple t_i takes value $v_{i,j}$ with probability $p_{i,j}$ and t_i does not appear in the database with probability $1 - \sum_j p_{i,j}$. It is easy to see the PRF value of t_i is

$$\begin{aligned}\Upsilon(t_i) &= \sum_{k>0} \omega(t_i, k) \Pr(r(t_i) = k) \\ &= \sum_{k>0} \omega(t_i, k) \sum_j \Pr(r(t_i) = k \wedge s_i = v_{i,j}) \\ &= \sum_j \Big(\sum_{k>0} \omega(t_i, k) \Pr(r(t_i) = k \wedge s_i = v_{i,j}) \Big)\end{aligned}$$

The algorithm works by treating the alternatives of the tuples (with a separate alternative for each different possible score for the tuple) as different tuples. In other words, we create a new tuple $t_{i,j}$ for each $v_{i,j}$ value. $t_{i,j}$ has existence probability $p_{i,j}$. Then, we add an *xor* constraint over the alternatives $\{t_{i,j}\}_j$ of each tuple t_i. We can then use the algorithm for the probabilistic and/xor tree model to find the values of the PRF function for each $t_{i,j}$ separately. Note that $\Pr(r(t_i) = k \wedge s_i = v_{i,j})$ is exactly the probability that $r(t_{i,j}) = k$ in the and/xor tree. Thus, by the above equation, we have that $\Upsilon(t_{i,j}) = \sum_{k>0} \omega(t_i, k) \Pr(r(t_i) = k \wedge s_i = v_{i,j})$ and $\Upsilon(t_i) = \sum_j \Upsilon(t_{i,j})$. Therefore, in a final step, we calculate the Υ score for each original tuple t_i by adding the Υ scores of its alternatives $\{t_{i,j}\}_j$. If the original tuples were independent, the complexity of this algorithm is $O(n^2)$ for computing the PRF function, and $O(n \log(n))$ for computing the $\mathsf{PRF}^{\mathsf{e}}$ function where n is the size of the input, i.e., the total number of different possible scores.

5.1.5 Summary

We summarize the complexities of the algorithms for different models in Table 5.1. Now, we explain the entries with a superscript lable ♯ in the table.

	PRF	$\mathsf{PRF}^\omega(h)$	PRF^e
Independent	$O(n^2)$	$O(nh)$ ♮♯	$O(n)$ ♮
And/Xor tree (height: d)	$O(n^3)$ or $O(n^2 \log(n)d)$ ♯	$O(n^2 d)$ ♯	$O(nd)$ ♮♯
x-tuples	$O(n^2)$ ♯	$O(nh)$ ♮♯	$O(n)$ ♮

Table 5.1: Summary of the running times. ♮: There is an additive $O(n \log(n))$ term if the dataset is not pre-sorted by their scores. ♯: See Section 5.1.5 for details.

1. $\mathsf{PRF}^\omega(h)$ **over independent tuples**: This matches the best known upper bounds for U-Top by Yi et al. [182, 181] (the original algorithm in [167] runs in $O(n^2 k)$ time) and for $\mathsf{PT}(h)$ by Hua et al. [98], for independent tuples.

2. PRF **over and/xor trees**: We assume the height of the and/xor tree is d. We have two choices here. We have explained the $O(n^3)$ algorithm in Section 5.1.2. Now, we show how to achieve a running time of $O(n^2 \log(n)d)$, which is much better than $O(n^3)$ if $d \ll n$. The idea is similar to the one we used for computing PRF^e on and/xor trees in Section 5.1.3. At each node v in the and/xor tree, we maintain the generating function $\mathcal{F}_v(x, y)$ which is a polynomial of x, y. In the ith iteration, we need to update the labels of t_{i-1} and t_i. The label update of a leaf f incurs the updates of the generating functions associated with the nodes on the leaf-to-root path, according to the updating rules (5.7) and (5.8). We use the FFT (fast Fourier transform) algorithm to implement the polynomial multiplication and division. Hence, each update can be done in $O(n \log(n))$ time. We have n iterations and in each iteration, we need to update at most d nodes. So the total running time is $O(n^2 \log(n)d)$.

3. $\mathsf{PRF}^\omega(h)$ **over and/xor trees**: We can decompose an PRF^ω into a linear combination of n PRF^e functions using Equation 5.10 (see the next section) and use the $O(nd)$ time algorithm to compute each of the PRF^e values. Thus, the total running time is $O(n^2 d)$. This is better than the algorithm for computing general PRF functions by a logarithmic factor.

4. $\mathsf{PRF}^{\mathsf{e}}$ **over and/xor trees**: For $\mathsf{PRF}^{\mathsf{e}}$ computation on and/xor trees, we use $\mathsf{ANDXOR\text{-}PRF^{e}\text{-}RANK}$. Now, the procedure $\mathsf{UPDATE}(\mathcal{T}, t_i)$ runs in $O(d_i)$ time where d_i is the depth of tuple t_i in the and/xor tree, i.e., the length of path from the root to t_i. Therefore, the total running time is $O(\sum_i d_i + n\log(n))$. If the height of the and/xor tree is bounded by d, the running time is simply $O(nd + n\log(n))$.

5. PRF **over** x**-tuples**: For x-tuples, we can achieve a better running time than for general and/xor trees. The algorithm is the same as in 2. But we notice that in each iteration, we only need to update the generating function for the parent of the leaf, which is a $\textcircled{\vee}$ node, and the generating function for the root, which is a $\textcircled{\wedge}$ node. Also observe that the degree of generating function for the $\textcircled{\vee}$ node is at most 1 for both variables x and y. So the update on this node can be done in $O(1)$ time. The update on the root takes at most $O(n)$ time (multiplication/division of a polynomial of degree at most n and a polynomial of degree at most 1). We have n iterations, each taking linear time. Therefore, the total running time is $O(n^2)$.

6. $\mathsf{PRF}^{\omega}(h)$ **over** x**-tuples**: The algorithm is the same as in 4, except that we only keep the first $O(h)$ terms of the polynomial associated with the root. Each update of the root takes $O(h)$ time instead of $O(n)$. Hence, the running time is $O(nh)$.

   Note that the previously best known bound for $\mathsf{U\text{-}Rank}$ (for $k = h$) over x-tuples is $O(n^2h)$ [167, 181] and the best known algorithm for $\mathsf{PT}(h)$ over x-tuples runs in $O(n^2h)$ worst case time [98]. Our algorithm improves these bounds by a factor of n and is essentially optimal.

5.2 Approximating and Learning Ranking Functions

In this section, we discuss how to choose the PRF functions and their parameters. Depending on the application domain and the scenarios, there are two approaches to this:

1. If we know the ranking function we would like to use (say $\mathsf{PT}(h)$), then we can either simulate or approximate it using appropriate PRF functions.

2. If we are instead provided user preferences data, we can learn the parameters from them.

Clearly, we would prefer to use a $\mathsf{PRF}^{\mathsf{e}}$ function, if possible, since it admits highly efficient ranking algorithms. For this purpose, we begin with presenting an algorithm to find an approximation to an arbitrary PRF^{ω} function using a linear combination of $\mathsf{PRF}^{\mathsf{e}}$ functions. We then discuss how to learn a PRF^{ω} function from user preferences, and finally present an algorithm for learning a single $\mathsf{PRF}^{\mathsf{e}}$ function.

5.2.1 Approximating PRF^{ω} using $\mathsf{PRF}^{\mathsf{e}}$ Functions

A linear combination of complex exponential functions is known to be very expressive, and can approximate many other functions very well [26]. Specifically, given a PRF^{ω} function, if we can write $\omega(i)$ as: $\omega(i) \approx \sum_{l=1}^{L} u_l \alpha_l^i$, then we have that:

$$\Upsilon(t) = \sum_i \omega(i) \Pr(r(t) = i) \approx \sum_{l=1}^{L} u_l \left(\sum_i \alpha_l^i \Pr(r(t) = i) \right)$$

This reduces the computation of $\Upsilon(t)$ to L individual $\mathsf{PRF}^{\mathsf{e}}$ function computations, each of which only takes linear time. This gives us an $O(n \log(n) + nL)$ time algorithm for approximately ranking using PRF^{ω} function for independent tuples (as opposed to $O(n^2)$ for exact ranking).

Several techniques have been proposed for finding such approximations using complex exponentials [106, 26]. Those techniques are however computationally inefficient, involving computation of the inverses of large matrices and the roots of polynomials of high orders.

In this section, we present a clean and efficient algorithm, based on Discrete Fourier Transforms (DFT), for approximating a function $\omega(i)$, that approaches zero for large values of i (in other words, $\omega(i) \geq \omega(i+1) \forall i, \omega(i) = 0, i > h$). As we noted earlier, this captures the typical behavior of the $\omega(i)$ function. An example of such a function is the step function ($\omega(i) = 1 \forall i \leq h, = 0 \forall i > h$) which corresponds to the ranking function $\mathsf{PT}(h)$. At a high level, our algorithm starts with a DFT approximation of $\omega(i)$ and then adapts it by adding several damping, scaling and shifting factors.

Discrete Fourier transformation (DFT) is a well known technique for representing a function as a linear combination of complex exponentials (also called *frequency domain representation*). More specifically, a discrete function $\omega(i)$ defined on a finite domain $[0, N-1]$ can be decomposed into exactly N exponentials as:

$$\omega(i) = \frac{1}{N} \sum_{k=0}^{N-1} \psi(k) e^{\frac{2\pi \jmath}{N} ki} \qquad i = 0, \ldots, N-1. \tag{5.9}$$

where $\jmath$ is the imaginary unit and $\psi(0), \cdots, \psi(N-1)$ denotes the DFT transform of $\omega(0), \cdots, \omega(N-1)$. An immediate consequence of the above equation is that we can decompose the PRF^{ω} value of a tuple into n $\mathsf{PRF}^{\mathsf{e}}$ values (recall that n is the number of tuples) :

$$\Upsilon(t) = \sum_{i=1}^{n} \omega(i) \Pr(r(t) = i) = \frac{1}{N} \sum_{k=0}^{n-1} \psi(k) \Upsilon_k(t) \tag{5.10}$$

where $\Upsilon_k(t) = \sum_{i=1}^{n} \alpha_K^i \Pr(r(t) = i)$ and $\alpha_k = e^{\frac{2\pi \jmath}{n} k}$.

If we want to approximate ω by fewer, say L, exponentials, we can instead use the L DFT coefficients with maximum absolute value. Assume that $\psi(0), \ldots, \psi(L-1)$

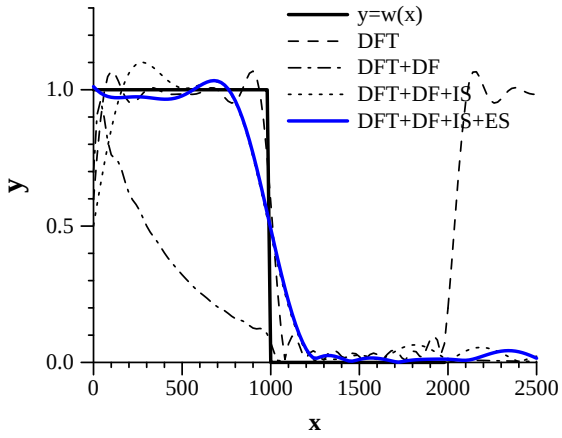


Figure 5.2: Illustrating the effect of the approximation steps: w(i) = step function with $N = 1000$, L = 20

are those coefficients. Then our approximation $\tilde{\omega}_L^{DFT}$ of ω by L exponentials is given by:

$$\tilde{\omega}_L^{DFT}(i) = \frac{1}{N}\sum_{k=0}^{L-1}\psi(k)e^{\frac{2\pi\jmath}{N}ki} \qquad i = 0, \ldots, N-1. \tag{5.11}$$

However, DFT utilizes only complex exponentials of unit norm, i.e., $e^{\jmath r}$ (where r is a real), which makes this approximation periodic (with a period of N). This is not suitable for approximating an ω function used in PRF, which is typically a monotonically non-increasing function. If we make N sufficiently large, say larger than the total number of tuples, then we usually need a large number of exponentials (L) to get a reasonable approximation. Moreover, computing DFT for very large N is computationally non-trivial. Furthermore, the number of tuples n may not be known in advance.

We next present a set of nontrivial tricks to adapt the base DFT approximation to overcome these shortcomings. We assume $\omega(i)$ takes non-zero values within interval $[0, N-1]$ and the absolute values of both $\omega(i)$ and $\omega_L^{DFT}(i)$ are bounded

by B. To illustrate our method, we use the step function:

$$\omega(i) = \begin{cases} 1, & i < N \\ 0, & i \geq N \end{cases}$$

with $N = 1000$ as our running example to show our method and the specific shortcomings it addresses. Figure 5.2 illustrates the effect of each of these adaptations.

1. **(DFT)** We perform pure DFT on the domain $[1, aN]$, where a is a small integer constant (typically < 10). As we can see in Figure 5.2 (where $N = 1000$ and $a = 2$), this results in a periodic approximation with a period of 2000. Although the approximation is reasonable for $x < 2000$, the periodicity is unacceptable if the number of tuples is larger than 2000 (since the positions between 2000 and 3000 (similarly, between 4000 and 5000) would be given high weights).

2. **(Damping Factor (DF))** To address this issue, we introduce a damping factor $\eta \leq 1$ such that $B\eta^{aN} \leq \epsilon$ where ϵ is a small positive real (for example, 10^{-5}). Our new approximation becomes:

$$\tilde{\omega}_L^{DFT+DF}(i) = \eta^i \cdot \tilde{\omega}_L^{DFT}(i) = \frac{1}{N} \sum_{k=0}^{L-1} \psi(k)(\eta e^{\frac{2\pi j}{N}k})^i. \tag{5.12}$$

   By incorporating this damping factor, the periodicity is mitigated, since we have: $\lim_{i \to +\infty} \tilde{\omega}_L^{DFT+DF}(i) = 0$. Especially, $\tilde{\omega}_L^{DFT+DF}(i) \leq \epsilon$ for $i > \alpha N$.

3. **(Initial Scaling (IS))** However the use of damping factor introduces another problem: it gives a biased approximation when i is small (see Figure 5.2). Taking the step function as an example, $\tilde{\omega}_L^{DFT+DF}(i)$ is approximately η^i for $0 \leq i < N$ instead of 1. To rectify this, we initially perform DFT on a different sequence $\hat{\omega}(i) = \eta^{-i}\omega(i)$ (rather than $\omega(i)$) on domain $\in [0, aN]$.

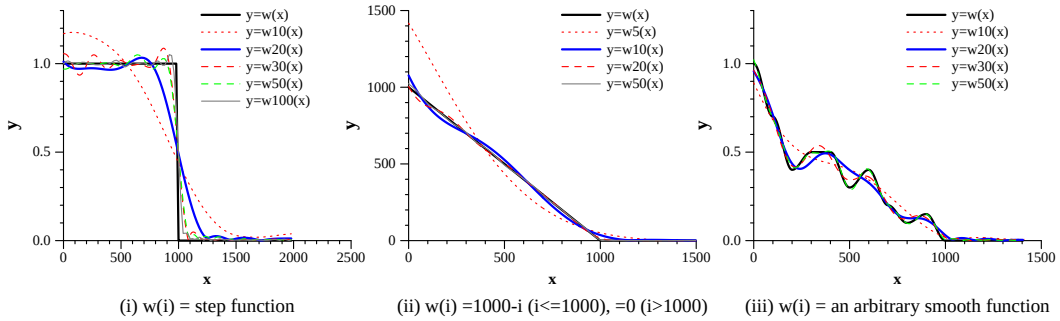


Figure 5.3: Approximating functions using linear combinations of complex exponentials: effect of increasing the number of coefficients

Therefore, $\tilde{\omega}^{DFT+IS}$ is a reasonable approximation of $\hat{\omega}$. Then, if we apply the damping factor, it will give us an unbiased approximation of ω, which we denote by $\tilde{\omega}^{DFT+DF+IS}$.

4. **(Extending and Shifting (ES))** This step is in particular tailored for optimizing the approximation performance for ranking functions. DFT does not perform well at discontinuous points, specifically at $i = 0$ (the left boundary), which can significantly affect the ranking approximation. To handle this, we extrapolate ω to make it continuous around 0. Let the resulting function be $\bar{\omega}$ which is defined on $[-bN, +\infty]$ for small $b > 0$. Again, taking the step function for example, we let $\bar{\omega}(i) = \begin{cases} 1, & -bN \leq i < N; \\ 0, & i \geq N. \end{cases}$ Then, we shift $\bar{\omega}(i)$ rightwards by bN to make its domain lie entirely in positive axis, do initial scaling and perform DFT on the resulting sequence. We denote the approximation of the resulting sequence by $\tilde{\omega}'(i)$(by performing (5.12)). For the approximation of original $\omega(i)$ values, we only need to do corresponding leftward shifting , namely $\tilde{\omega}^{DFT+DF+IS+ES}(i) = \tilde{\omega}'(i + bN)$. Figure 5.2 shows that DFT+DF+IS+ES gives a much better approximation than others around $i = 0$.

Figures 5.2 and 5.3(i) illustrate the efficacy of our approximation technique for the

step function. As we can see, we are able to approximate that function very well with just 20 or 30 coefficients. Figure 5.3(ii) and (iii) show the approximations for a piecewise linear function and an arbitrarily generated continuous function respectively, both of which are much easier to approximate than the step function.

5.2.2 Learning a PRF^ω or PRF^e Function

Next we address the question of how to learn the weights of a PRF^ω function or the α for a single PRF^e function from user preferences. To learn a linear combination of PRF^e functions, we first learn a PRF^ω function and then approximate it as above.

Prior work on learning ranking functions (e.g., [96, 108, 35, 60]) assumes that the user preferences are provided in the form of a set of pairs of tuples, and for each pair, we are told which tuple is ranked higher. Our problem differs slightly from this prior work in that, the features that we use to rank the tuples (i.e., $\Pr(r(t) = i), i = 1, \ldots, n$) cannot be computed for each tuple individually, but must be computed for the entire dataset (since the values of the features for a tuple depend on the other tuples in the dataset). Hence, we assume that we are instead given a small sample of the tuples, and the user ranking for all those tuples. We compute the features assuming this sample constitutes the entire relation, and learn a ranking function accordingly, with the goal to find the parameters (the weights w_i for PRF^ω or the parameter α for PRF^e) that minimize the number of disagreements with the provided ranking over the samples.

Given this, the problem of learning PRF^ω is identical to the problem addressed in the prior work, and we utilize the algorithm based on *support vector machines (SVM)* [108] in our experiments.

On the other hand, we are not aware of any work that has addressed learning a ranking function like PRF^e. We use a simple binary search-like heuristic to find the optimal real value of α that minimizes the Kendall distance between the user-specified ranking and the ranking according to $\mathsf{PRF}^\mathsf{e}(\alpha)$. In other words, we try to find $\arg\min_{\alpha \in [0,1]}(\mathsf{dis}(\sigma, \sigma(\alpha)))$ where $\mathsf{dis}()$ is the Kendall distance between

two rankings, σ is the ranking for the given sample and $\sigma(\alpha)$ is the one obtained by using $\mathsf{PRF}^{\mathsf{e}}(\alpha)$ function. Suppose we want to find the optimal a within the interval $[L, U]$ now. We first compute $\mathsf{dis}(\sigma, \sigma(L + i \cdot \frac{U-L}{10})$ for $i = 1, \ldots, 9$ and find i for which the distance is the smallest. Then we reduce our search range to $[\max(L, L+(i-1)\cdot\frac{U-L}{10}, \min(U, L+(i+1)\cdot\frac{U-L}{10})]$ and repeat the above recursively. Although this algorithm can only converge to a local minimum, in our experimental study, we observed that all of the prior ranking functions exhibit a uni-valley behavior (Section 5.3), and in such cases, this algorithm finds the global optimal.

5.2.3 An Interesting Property of $\mathsf{PRF}^{\mathsf{e}}$

We have seen that $\mathsf{PRF}^{\mathsf{e}}(\alpha)$ admits very efficient evaluation algorithms. We also suggest that the parameter α should be learnt from samples/feedbacks in Section 5.2.2. In fact, we do so since since we hold the promise that by changing the parameter α, $\mathsf{PRF}^{\mathsf{e}}$ can span a spectrum of rankings, and the true ranking should be in or close to some point of the spectrum. We will demonstrate this fact shortly in our experiment section (Section 5.3). In this section, we want to make some interesting theoretical observation, which may help to further reveal this fact and understand the behavior of $\mathsf{PRF}^{\mathsf{e}}$ itself.

First, we can easily observe that for $\alpha = 1$, the $\mathsf{PRF}^{\mathsf{e}}$ ranking is equivalent to the ranking of tuples by their existence probabilities; On the other hand, when α approaches to 0, $\mathsf{PRF}^{\mathsf{e}}$ tends to rank the tuples by their probabilities to be the Top-1 answer, i.e, $\Pr(r(t) = 1)$. Thus, it is a natural question to ask that is how the ranking changes when we vary α from 0 to 1. Now, we prove the following theorem which gives a important characterization of the behavior of $\mathsf{PRF}^{\mathsf{e}}$ on tuple independent databases.

Theorem 5.2. *Let τ_0 and τ_1 be the rankings obtained by sorting the tuples in a nonincreasing $\Pr(r(t) = 1)$ and $\Pr(t)$ order, respectively. Let τ_α be the ranking obtained by $\mathsf{PRF}^{\mathsf{e}}(\alpha)$.*

1. *If $t_i >_{\tau_0} t_j$ (t_i is ranked higher than t_j in τ_0) and $t_i >_{\tau_1} t_j$, then $t_i >_{\tau_\alpha} t_j$ any $0 \leq \alpha \leq 1$.*

2. *If $t_i >_{\tau_0} t_j$ and $t_i <_{\tau_1} t_j$, then there is exactly one point β such that $t_i >_{\tau_\alpha} t_j$ for $\alpha < \beta$ and $t_i <_{\tau_\alpha} t_j$ for $\alpha > \beta$.*

Proof: We denote $\Upsilon_\alpha(t_i)$ be the $\mathsf{PRF}(\alpha)$ value of tuple t_i. We know that

$$\Upsilon_\alpha(t_i) = \mathcal{F}^i(\alpha) = \left(\prod_{t \in T_{i-1}} \big(1 - \Pr(t) + \Pr(t)\alpha\big)\right) \Pr(t_i)\alpha.$$

Assume that $i < j$. Dividing $\Upsilon_\alpha(t_j)$ by $\Upsilon_\alpha(t_i)$, we get

$$\begin{aligned}\rho_{j,i}(\alpha) = \frac{\Upsilon_\alpha(t_j)}{\Upsilon_\alpha(t_i)} &= \frac{\prod_{t \in T_{j-1}} \big(1 - \Pr(t) + \Pr(t)\alpha\big)}{\prod_{t \in T_{i-1}} \big(1 - \Pr(t) + \Pr(t)\alpha\big)} \cdot \frac{\Pr(t_j)}{\Pr(t_i)} \\ &= \prod_{l=i}^{j-1} \big(1 - \Pr(t_l) + \Pr(t_l)\alpha\big) \cdot \frac{\Pr(t_j)}{\Pr(t_i)}\end{aligned}$$

Notice that $1 - \Pr(t) + \Pr(t)\alpha$ is always nonnegative and a increasing function of α. Therefore, $\rho_{j,i}(\alpha)$ is increasing in α. If $i > j$, the same argument show $\rho_{j,i}(\alpha)$ is decreasing in α. In either case, the ratio is monotone in α.

If $\rho_{j,i}(0) < 1$ and $\rho_{j,i}(1) < 1$, then $\rho_{j,i}(\alpha) < 1$ for all $0 < \alpha \leq 1$. Therefore, the first half of the theorem holds. If $\rho_{j,i}(0) < 1$ and $\rho_{j,i}(1) > 1$, then there is exactly one point $0 < \beta < 1$ such that $\rho_{j,i}(\beta) = 1$ and $\rho_{j,i}(\alpha) < 1$ for all $0 < \alpha < \beta$. and $\rho_{j,i}(\alpha) < 1$ for all $\beta < \alpha \leq 1$. This proves the second half. □

Some nontrivial questions can be immediately answered by the theorem. For example, one may ask the question "Is it possible that we get some ranking τ_1, increase α a bit and get another ranking τ_2, and increase α further and get τ_1 back?" and we can quickly see that the answer is no since if two tuples change positions, they never change back. Another example question asks "Suppose t_1 dominates t_2 (i.e., t_1 has a higher score and probability), should t_1 always rank

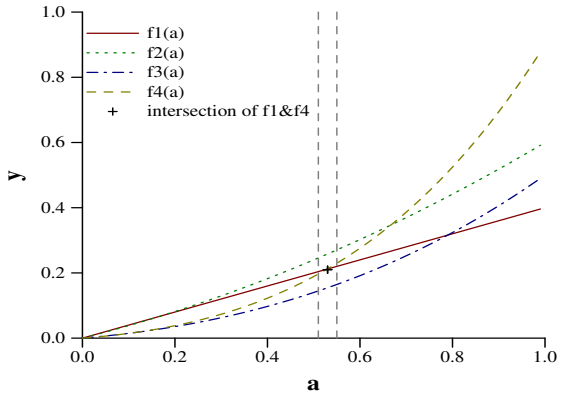


Figure 5.4: Illustration of Example 11. $f_i(\alpha) = \Upsilon_\alpha(t_i)$ for $i = 1, 2, 3, 4$.

above t_2 no matter what α is?" and we can easily say yes by just checking the fact that t_1 ranks above t_2 in both τ_0 and τ_1.

Interestingly, the process of the changing of the rank list is a reminiscence of the execution of the bubble sort algorithm. We assume the true order of the tuples is τ_1 and the initial order is τ_0. We increase α from 0 to 1 gradually. Each time when the rank list changes, the change is just a swap of a pair of adjacent tuples that is not in the right relative order initially. The number of swaps is exactly the number of reverse pairs. This is just like bubble sort! The only difference is that the order of those swaps may not be the same.

Example 11. *Suppose we have four independent tuples:* $(t_1 : 100, .4), (t_2 : 80, .6), (t_3 : 50, .5), (t_4 : 30, .9)$. *Using (5.6), it is easy to see that* $\Upsilon_\alpha(t_1) = .4\alpha, \Upsilon_\alpha(t_2) = (.6+.4\alpha).6\alpha, \Upsilon_\alpha(t_3) = (.6+.4\alpha)(.4+.6\alpha).5\alpha$ *and* $\Upsilon_\alpha(t_4) = (.6+.4\alpha)(.4+.6\alpha)(.5+.5\alpha).9\alpha$. *In Figure 5.4, each curve corresponds to one tuple. We can see in interval* $(0, 1]$, *any two curves intersect at most once. The change of the rank happens right at the intersection points and one adjacent pair swap their positions. For instance, the + sign in the figure is the intersection point of* f_1 *and* f_4. *The rank list is* $\{t_2, t_1, t_4, t_3\}$ *right before the point and* $\{t_2, t_4, t_1, t_3\}$ *right after the point.*

In fact, if we think h as a parameter of $\mathsf{PT}(h)$ and we vary h from 1 to n, the process that the rank list changes is quite similar to the one for PRF^e: On one

extreme where $h = 1$, the rank list is τ_0, i.e., the tuples are sorted by $\Pr(r(t) = 1)$ and on the other extreme where $h = n$, the rank list is τ_1, i.e., the tuples are sorted by $\Pr(r(t) \leq n) = \Pr(t)$. However, $\mathsf{PT}(h)$ is only able to explore at most n different rankings (one for each h) "between" τ_0 and τ_1, while PRF^e may explore $O(n^2)$ of them.

5.3 Experimental Study

We conducted an extensive empirical study over several real and synthetic datasets to illustrate: (a) the diverse and conflicting behavior of different ranking functions proposed in the prior literature, (b) the effectiveness of our parameterized ranking functions, especially PRF^e, at approximating other ranking functions, and (c) the scalability of our new generating functions-based algorithms for exact and approximate ranking. We discussed the results supporting (a) in Section 2.4. In this section, we focus on (b) and (c).

Datasets We mainly use the International Ice Patrol (IIP) Iceberg Sighting Dataset[1] for our experiments. This dataset was also used in prior work on ranking in probabilistic databases [107, 98]. The database contains a set of *iceberg sighting records*, each of which contains the location (*latitude, longitude*) of the iceberg, and the *number of days* the iceberg has drifted, among other attributes. Detecting the icebergs that have been drifting for long periods is crucial, and hence we use the number of days drifted as the ranking score. The sighting record is also associated with a *confidence-level* attribute according to the source of sighting: R/V (radar and visual), VIS (visual only), RAD (radar only), SAT-LOW (low earth orbit satellite), SAT-MED (medium earth orbit satellite), SAT-HIGH (high earth orbit satellite), and EST (estimated). We converted these six confidence levels into probabilities 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, and 0.4 respectively. We added a very small Gaussian noise to each probability so that ties could be broken. There are nearly a

[1] http://nsidc.org/data/g00807.html

million records available from 1960 to 2007; we created 10 different datasets for our experimental study containing $100,000$ (IIP-100,000) to $1,000,000$ (IIP-1,000,000) records, by uniformly sampling from the original dataset.

Along with the real datasets, we also use several synthetic datasets with varying degrees of correlations, where the correlations are captured using probabilistic and/xor trees. The tuple scores (for ranking) were chosen uniformly at random from $[0, 10000]$. The corresponding and/xor trees were also generated randomly starting with the root, by controlling the *height (L)*, the *maximum degree of the non-root nodes (d)*, and the *proportion of* $\veebar$ *and* $\wedge$ *nodes (X/A)* in the tree. Specifically, we use five such datasets:

1. Syn-IND (independent tuples): the tuple existence probabilities were chosen uniformly at random from $[0, 1]$.
2. Syn-XOR (L=2,X/A=∞,d=5): Note that the Syn-XOR dataset, with height set to 2 and no $\wedge$ nodes, exhibits only mutual exclusivity correlations (mimicking the x-tuples model [157, 182])
3. Syn-LOW (L=3,X/A=10,d=2)
4. Syn-MED (L=5,X/A=3,d=5)
5. Syn-HIGH (L=5,X/A=1,d=10).

Setup We use the normalized Kendall distance (Section 2.5) for comparing two top-k rankings. All the algorithms were implemented in C++, and the experiments were run on a 2.4GHz Linux PC with 2GB memory.

5.3.1 Approximability of Ranking Functions

We begin with a set of experiments illustrating the effectiveness of our parameterized ranking functions at approximating other ranking functions. Due to space

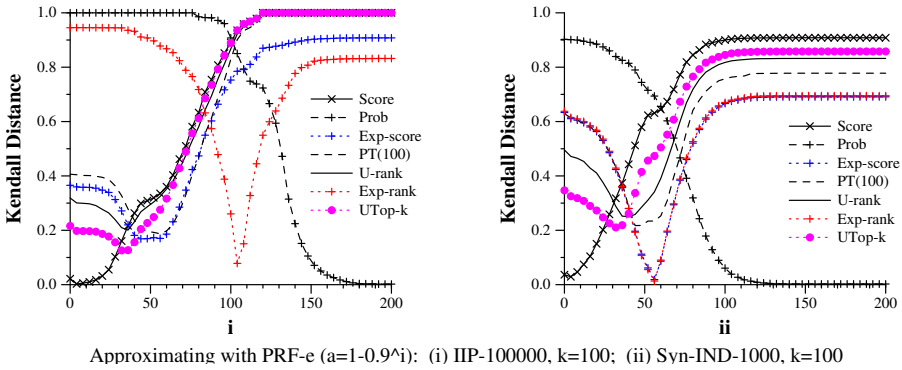
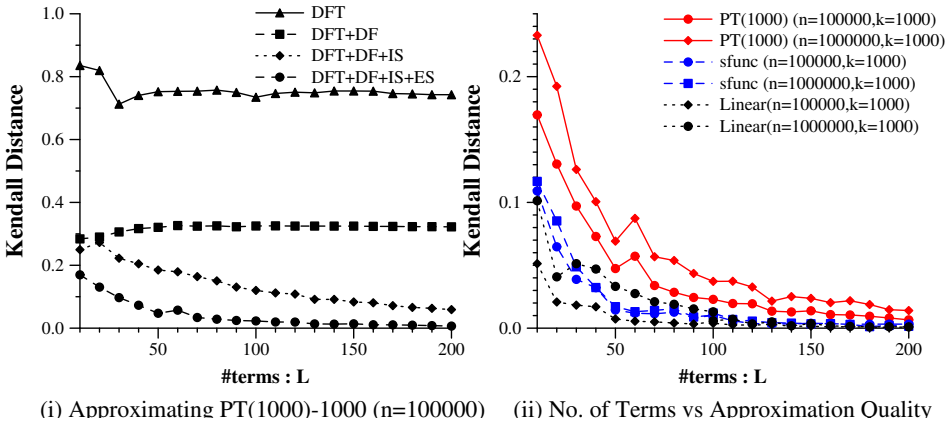


Figure 5.5: Comparing $\mathsf{PRF}^{\mathsf{e}}$ with other ranking functions for varying values of α; (i))IIP-100,000, (ii)Syn-IND-1000

constraints, we focus on $\mathsf{PRF}^{\mathsf{e}}$ here because it is significantly faster to rank according to a $\mathsf{PRF}^{\mathsf{e}}$ function (or a linear combination of several $\mathsf{PRF}^{\mathsf{e}}$ functions) than it is to rank according a PRF^{ω} function.

Figures 5.5 (i) and (ii) show the Kendall distance between the Top-100 answers computed using a specific ranking function and $\mathsf{PRF}^{\mathsf{e}}$ for varying values of α, for the IIP-100,000 and Syn-IND-1000 datasets. For better visualization, we plot i on the x-axis, where $\alpha = 1 - 0.9^i$. The reason behind this is that the behavior of the $\mathsf{PRF}^{\mathsf{e}}$ function changes rather drastically, and spans a spectrum of rankings, when α approaches 1. First, as we can see, the $\mathsf{PRF}^{\mathsf{e}}$ ranking is close to ranking by *Score* alone for small values of α, whereas it is close to the ranking by *Probability* when α is close to 1 (in fact, for $\alpha = 1$, the $\mathsf{PRF}^{\mathsf{e}}$ ranking is equivalent to the ranking of tuples by their existence probabilities)[2]. Second, we see that, for all other functions (E-Score, $\mathsf{PT}(h)$, U-Rank, E-Rank), there exists a value of α for which the distance of that function to $\mathsf{PRF}^{\mathsf{e}}$ is very small, indicating that $\mathsf{PRF}^{\mathsf{e}}$ can indeed approximate those functions quite well. Moreover we observe that this "uni-valley" behavior of the curves justifies the binary search algorithm we advocate for learning the value of α in Section 5.2.2. Our experiments with other synthetic and

[2]On the other hand, for $\alpha = 0$, $\mathsf{PRF}^{\mathsf{e}}$ ranks the tuples by their probabilities to be the Top-1 answer.



(i) Approximating PT(1000)-1000 (n=100000) (ii) No. of Terms vs Approximation Quality

Figure 5.6: (i) Approximating $\mathsf{PT}(1000)$ using a linear combination of PRF^e functions; (ii) Approximation quality for three ranking functions for varying number of exponentials.

real datasets indicated a very similar behavior by the ranking functions.

Next we evaluate the effectiveness of our approximation technique presented in Section 5.2. In Figure 5.6 (i), we show the Kendall distance between the top-k answers obtained using $\mathsf{PT}(h)$ (for $h = 1000, k = 1000$) and using a linear combination of PRF^e functions found by our algorithms. As expected, the approximation using the vanilla DFT technique is very bad, with the Kendall distance close to 0.8 indicating little similarity between the top-k answers. However, the approximation obtained using our proposed algorithm (indicated by DFT+DF+IS+ES curve) achieves a Kendall distance of less than 0.1 with just $L = 20$ exponentials.

In Figure 5.6 (ii), we compare the approximation quality (found by our algorithm DFT+DF+IS+ES) for three ranking functions for two datasets: IIP-100,000 with $k = 1000$, and IIP-1,000,000 dataset with $k = 10000$. The ranking functions we compared were: (1) $\mathsf{PT}(h)$ ($h = 1000$), (2) an arbitrary smooth function, $sfunc$, and (3) a linear function (Figure 5.6(ii)). We see that $L = 40$ suffices to bring the Kendall distance to < 0.1 in all cases. We also observe that smooth functions (for which the absolute value of the first derivative of the underlying continuous function is bounded by a small value) are usually easier to approximate. We only need $L = 20$ exponentials to achieve a Kendall distance less than 0.05 for $sfunc$.

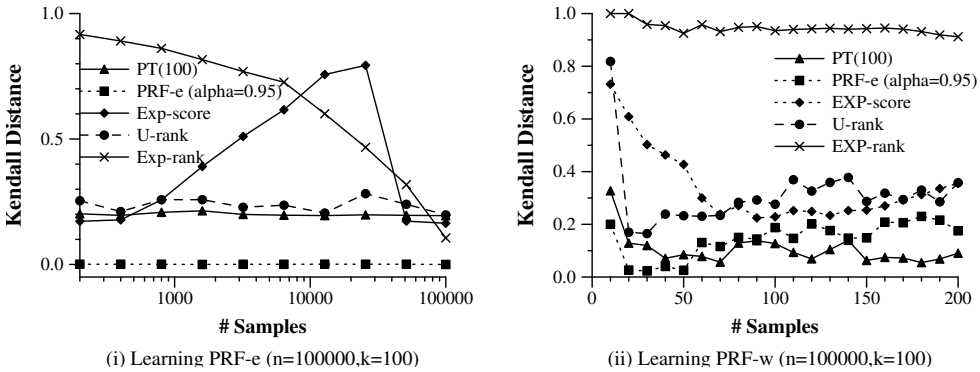


Figure 5.7: (i) Learning $\mathsf{PRF}^{\mathsf{e}}$ from user preferences; (ii) Learning PRF^{ω} from user preferences.

The Linear function is even easier to approximate.

5.3.2 Learning Ranking Functions

Next we consider the issue of learning ranking functions from user preferences. Lacking real user preference data, we instead assume that the user ranking function, denoted *user-func*, is identical to one of: $\mathsf{E\text{-}Score}$, $\mathsf{PT}(h)$, $\mathsf{U\text{-}Rank}$, $\mathsf{E\text{-}Rank}$, or $\mathsf{PRF}^{\mathsf{e}}(\alpha = 0.95)$. We generate a set of user preferences by ranking a random sample of the dataset using *user-func* (thus generating five sets of user preferences). These are then fed to the learning algorithm, and finally we compare the Kendall distance between the learned ranking and the true ranking for the entire dataset.

In Figure 5.7(i), we plot the results for learning a single $\mathsf{PRF}^{\mathsf{e}}$ function (i.e., for learning the value of α) using the binary search-like algorithm presented in Section 5.2.2. The experiment reveals that when the underlying ranking is done by $\mathsf{PRF}^{\mathsf{e}}$, the value of α can be learned perfectly. When one of $\mathsf{PT}(h)$ or $\mathsf{U\text{-}Rank}$ is the underlying ranking function, the correct value a can be learned with a fairly small sample size, and increasing the number of samples does not help in finding a better α. On the other hand, $\mathsf{E\text{-}Rank}$ cannot be learned well by $\mathsf{PRF}^{\mathsf{e}}$ unless the sample size approaches the total size of whole dataset. This phenomenon can be partly

explained using Figure 5.5(i) and (ii) in which the curves for $\mathsf{PT}(h)$ and U-Top have a fairly smooth valley, while the one for E-Rank is very sharp and the region of α values where the distance is low is extremely small ($[1 - 0.9^{90}, 1 - 0.9^{110}]$). Hence, the minimum point for E-Rank is harder to reach. Another reason is that E-Rank is quite sensitive to the size of the dataset, which makes it hard to learn it using a smaller-sized sample dataset. We also observe that while extremely large samples are able to learn E-Score well, the behavior of E-Score is quite unstable when the sample size is smaller.

Note that if we already know the form of the ranking function, we don't need to learn it in this fashion; we can instead directly find an approximation for it using our DFT-based algorithm.

In Figure 5.7 (ii), we show the results of an experiment where we tried to learn a PRF^ω function (using the SVM-lite package [108]). We keep our sample size ≤ 200 since SVM-lite becomes drastically slow with larger sample sizes. First we observe that $\mathsf{PT}(h)$ and PRF^e can be learned very well from a small size sample (distance < 0.2 in most cases) and increasing the sample size does not benefit significantly. U-Rank can also be learned, but the approximation isn't nearly as good. This is because U-Rank cannot be written as a single PRF^ω function. We observed similar behavior in our experiments with other datasets. Due to space constraints, we omit a further discussion on learning a PRF^ω function; the issues in learning such weighted functions have been investigated in prior literature, and if the true ranking function can be written as a PRF^ω function, then the above algorithm is expected to learn it well given a reasonable number of samples.

5.3.3 Effect of Correlations

Next we evaluate the behavior of ranking functions over probabilistic datasets modeled using probabilistic and/xor trees. We use the four synthetic correlated datasets, Syn-XOR, Syn-LOW, Syn-MED, and Syn-HIGH, for these experiments. For each dataset and each ranking function considered, we compute the rank-

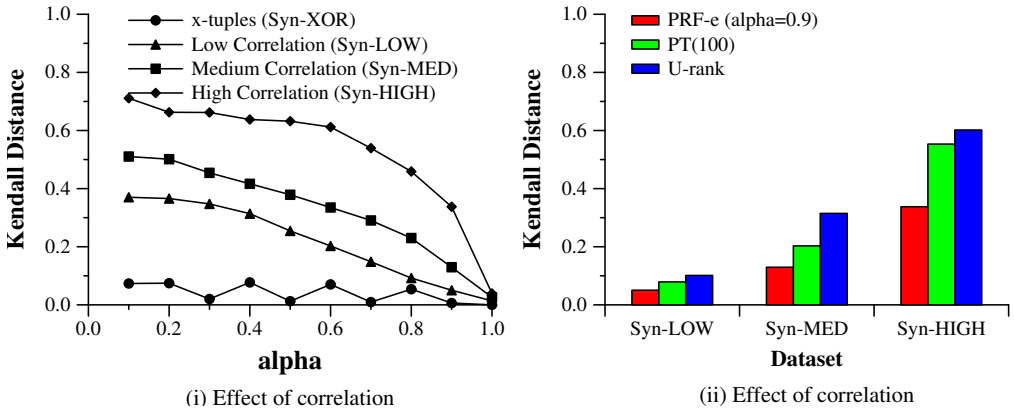


Figure 5.8: (i) Effect of correlations on $\mathsf{PRF}^{\mathsf{e}}$ ranking as a varies; (ii) Effect of correlations on $\mathsf{PRF}^{\mathsf{e}}$, U-Rank and $\mathsf{PT}(h)$.

ings by considering the correlations, and by ignoring the correlations, and then compute the Kendall distance between these two (e.g., for $\mathsf{PRF}^{\mathsf{e}}$, we compute the rankings using **PROB-ANDOR-PRF-RANK** and **IND-PRF-RANK** algorithms). Figure 5.8(i) shows the results for the $\mathsf{PRF}^{\mathsf{e}}$ ranking function for varying α, whereas in Figure 5.8(ii), we plot the results for $\mathsf{PRF}^{\mathsf{e}}(\alpha = 0.9)$, $\mathsf{PT}(100)$, and U-Rank.

As we can see, on highly correlated datasets, ignoring the correlations can result in significantly inaccurate top-k answers. This is not as pronounced for the Syn-XOR dataset. This is because, in any group of tuples that are mutually exclusive, there are typically only a few tuples that may have sufficiently high probabilities to be part of the top-k answer; the rest of the tuples may be ignored for ranking purposes. Because of this, assuming tuples to be independent of each other does not result in significant errors. As α approaches 1, $\mathsf{PRF}^{\mathsf{e}}$ tends to sort the tuples by probabilities, so all four curves in Figure 5.8(i) become close to 0. We note that ranking by E-Score is invariant to the correlations, which is a significant drawback of that function.

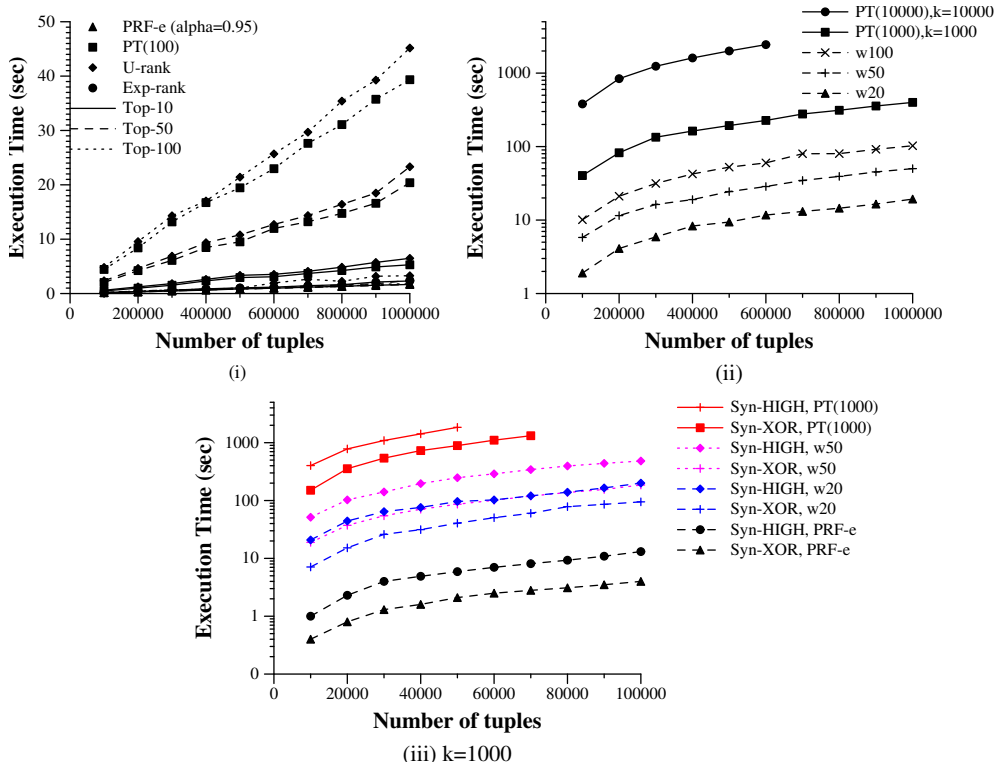


Figure 5.9: Experiments comparing the execution times of the ranking algorithms (note that the y-axis is log-scale for (ii) and (iii))

5.3.4 Execution Times

Figure 5.9(i) shows the execution times for four ranking functions: PRF^e, $\mathsf{PT}(h)$, U-Rank and E-Rank, for the IIP-datasets, for different dataset sizes and k. We note that the running time for PRF^ω is similar to that of $\mathsf{PT}(h)$. As expected, ranking by PRF^e or E-Rank is very efficient (1000000 tuples can be ranked within 1 or 2 seconds). Indeed, after sorting the dataset in an non-decreasing score order, PRF^e needs only a single scan of the dataset, and E-Rank needs to scan the dataset twice. Execution times for $PT(h)$ and U-Rank-k increase linearly with h and k respectively and the algorithms become very slow for high h and k. The running times of both PRF^e and E-Rank are not significantly affected by k.

Figure 5.9(ii) compares the execution time for $\mathsf{PT}(h)$ and its approximation using a linear combination of PRF^e functions (see Figure 5.6(i)), for two different

values of k. $w50$ indicates that 50 exponentials were used in the approximation (note that the approximate ranking, based on PRF^e, is insensitive to the value of k). As we can see, for large datasets and for higher values of k, exact computation takes several orders of magnitude more time to compute than the approximation. For example, the exact algorithm takes nearly 1 hour for $n = 500,000$ and $h = 10,000$ while the approximate answer obtained using $L = 50$ PRF^e functions takes only 24 seconds and achieves a Kendall distance 0.09.

For correlated datasets, the effect is even more pronounced. In Figure 5.9(iii), we plot the results of a similar experiment, but using two correlated datasets: Syn-XOR and Syn-HIGH. Note that the number of tuples in these datasets is smaller by a factor of 10. As we can see, our generating functions-based algorithms for computing PRF^e are highly efficient, even for datasets with high degrees of correlation. As above, approximation of the $\mathsf{PT}(h)$ ranking function using a linear combination of PRF^e functions is significantly cheaper to compute than using the exact algorithm.

Combined with the previous results illustrating that a linear combination of PRF^e functions can approximate other ranking functions very well, this validates the unified ranking approach that we propose in this paper.

5.4 PRF Computation for Graphical Models

In this chapter, we present an algorithm for computing the PRF function values for all tuples of a correlated dataset when the correlations are represented using a graphical model. We assume that the probability distributions considered in this chapter are discrete. The resulting algorithm is a non-trivial dynamic program over the *junction tree* of the graphical model. Our main result is that we can compute the PRF function in polynomial time if the junction tree of the graphical model has bounded treewidth. It is worth noting that this result cannot subsume our algorithm for and/xor trees (Section 5.1.2) since the treewidth of the moralized

graph of a probabilistic and/xor tree may not be bounded. In some sense, this is close to *instance-optimal* since the complexity of the underlying inference problem is itself exponential in the treewidth of the graphical model (this however does not preclude the possibility that the ranking itself could be done more efficiently without computing the PRF function explicitly – however, such an algorithm is unlikely to exist).

5.4.1 Problem Simplification

We begin with describing the first step of our algorithm, and defining a reduced and simpler to state problem. Let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of tuples, sorted in an non-increasing order of their score values. For each tuple t in T, we associate an indicator random variable X_t, which is 1 if t is present, and 0 otherwise. Let $\mathcal{X} = \{X_{t_1}, \ldots, X_{t_n}\}$ and $\mathcal{X}_i = \{X_{t_1}, \ldots, X_{t_i}\}$. Assume that the junction tree over $\mathcal{X}$ is known to us.

Recall that our goal is to rank the tuples according to $\Upsilon(t_i) = \sum_{j>0} \omega(j) \Pr(r(t_i) = j)$. For this purpose, we first compute the positional probabilities, $\Pr(r(t_i) = j)$ $\forall j \;\; \forall t_i$, using the algorithms presented in the next two subsections. Given those, the values of $\Upsilon(t_i)$ can be computed in $O(n^2)$ time for all tuples, and the ranking itself can be done in $O(n \log(n))$ time (by sorting). The positional probabilities ($Pr(r(t_i) = j)$) may also be of interest by themselves.

For each tuple t_i, we compute $\Pr(r(t_i) = j)\ \forall j$ at once. Recall that $\Pr(r(t_i) = j)$ is the probability that t_i exists (i.e., $X_i = 1$) and exactly $j - 1$ tuples with scores higher than t_i are present (i.e., $\sum_{l=1}^{i-1} X_l = j - 1$). In other words:

$$\begin{aligned} \Pr(r(t_i) = j) &= \Pr(X_i = 1 \wedge \sum_{l=1}^{i-1} X_l = j - 1) \\ &= \Pr((\sum_{l=1}^{i-1} X_l = j - 1) | X_i = 1) \Pr(X_i = 1) \end{aligned}$$

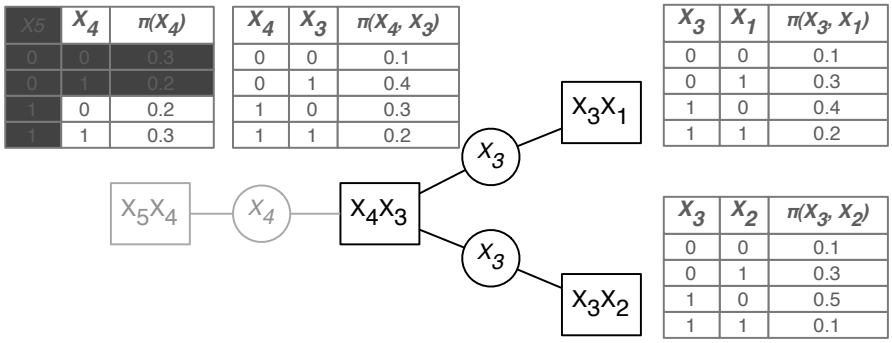


Figure 5.10: Conditioning on $X_5 = 1$ results in a smaller junction tree, with uncalibrated potentials, that captures the distribution over X_1, X_2, X_3, X_4 given $X_5 = 1$.

Hence, we begin with first conditioning the junction tree by setting $X_i = 1$, and re-calibrating. This is done by identifying all cliques and separators which contain X_i, and by updating the corresponding probability distributions by removing the values corresponding to $X_i = 0$. More precisely, we replace a probability distribution $\Pr(X_{i_1}, \ldots, X_{i_k}, X_i)$, by a potential $\pi(X_{i_1}, \ldots, X_{i_k})$ computed as:

$$\pi(X_{i_1} = v_1, \ldots, X_{i_k} = v_k) = \Pr(X_{i_1} = v_1, \ldots, X_{i_k} = v_k, X_i = 1)$$

π is not a probability distribution since the entries in it may not sum up to 1. Further, the potentials may not be consistent with each other. Hence, we need to recalibrate this junction tree using message passing [69]. As mentioned earlier, this takes $O(n2^{tw})$ time. We use Example 4 (in Section 2.3.2) to illustrate the algorithm. Figure 5.10 shows the resulting (uncalibrated) junction tree after conditioning on $X_5 = 1$.

If X_i is a separator in the junction tree, then we get more than one junction tree after conditioning on $X_i = 1$. Figure 5.11 shows the two junction trees we would get after conditioning on $X_4 = 1$. The variables in these junction trees are independent of each other (this follows from the Markov property), and the junction trees can be processed separately from each other.

Since the resulting junction tree or junction trees capture the probability dis-

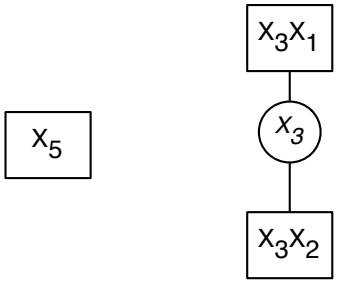


Figure 5.11: Conditioning on $X_4 = 1$ results in two junction trees.

tribution conditioned on the event $X_i = 1$, our problem now reduces to finding the probability distribution of $\sum_{l=1}^{i-1} X_l$ in those junction trees. For cleaner description of the algorithm, we associate an indicator variable δ_{X_l} with each variable X_l in the junction tree. δ_{X_l} is set to 1 if $l \leq i-1$, and is 0 otherwise. This allows us to state the key problem to be solved as follows:

Redefined Problem[3]*: Given a junction tree over m binary variables $Y_1, \ldots, Y_m$, where each variable Y_j is associated with an indicator variable $\delta_{Y_j} \in \{0, 1\}$, find the probability distribution of the random variable $PS = \sum_{l=1}^{m} Y_l \delta_l$.*

If the result of the conditioning was a single junction tree (over $m = n - 1$ variables), we multiply the resulting probabilities by $\Pr(X_i = 1)$ to get the rank distribution of t_i.

However, if we get $k > 1$ junction trees, then we need one additional step. Let $PS_1, \ldots, PS_k$ be the random variables denoting the partial sums for each of junction trees. We need to combine the probability distributions over these partial sums, $\Pr(PS_1), \ldots, \Pr(PS_k)$, into a single probability distribution over $\Pr(PS_1 + \cdots + PS_k)$. This can be done by repeatedly applying the following general formula:

$$\Pr(PS_1 + PS_2 = a) = \sum_{j=0}^{a} \Pr(PS_1 = j) \Pr(PS_2 = a - j)$$

A naive implementation of the above takes time $O(n^2)$. Although this can be

[3]We rename the variables to avoid confusion.

improved using the ideas presented in Appendix **??**, the complexity of computing $\Pr(PS_i)$ is much higher and dominates the overall complexity.

Next we present algorithms for solving the redefined problem.

5.4.2 Algorithm for Markov Sequences

We first describe an algorithm for Markov chains, a special, yet important, case of the graphical models. Markov chains appear naturally in many settings, and have been studied in probabilistic database literature as well [110, 152, 118]. Any finite-length Markov chain is a Markov network whose underlying graph is simply a path: each variable is directly dependent on only its predecessor and successor. The junction tree for a Markov chain is also a path in which each node corresponds to an edge of the Markov chain. The treewidth of such a junction tree is one. Without loss of generality, we assume that the Markov chain is $Y_1, \ldots, Y_m$ (Figure 5.12(i)). The corresponding junction tree $\mathcal{T}$ is a path with cliques $C_j = \{Y_j, Y_{j+1}\}$ as shown in the figure.

We compute the distribution $\Pr(\sum_{l=1}^{m} Y_l \delta_l)$ recursively. Let $PS_j = \sum_{l=1}^{j} Y_l \delta_l$ denote the partial sum over the first j variables $Y_1, \ldots, Y_j$.

At the clique $\{Y_{j-1}, Y_j\}$, $j \geq 1$, we recursively compute the joint probability distribution: $\Pr(Y_j, PS_{j-1})$. The initial distribution $\Pr(Y_2, PS_1)$, $PS_1 = \delta_1 Y_1$, is computed directly:

$$\begin{aligned} \Pr(Y_2, PS_1 = 0) &= \Pr(Y_2, Y_1 = 0) + (1 - \delta_i) \Pr(Y_2, Y_1 = 1) \\ \Pr(Y_2, PS_1 = 1) &= \delta_i \Pr(Y_2, Y_1 = 1). \end{aligned}$$

Given $\Pr(Y_j, PS_{j-1})$, we compute $\Pr(Y_{j+1}, PS_j)$ as follows. Observe that PS_{j-1} and Y_{j+1} are conditionally independent given the value of Y_j (by Markov property). Thus we have:

$$\Pr(Y_{j+1}, Y_j, PS_{j-1}) = \frac{\Pr(Y_{j+1}, Y_j) \Pr(Y_j, PS_{j-1})}{\Pr(Y_j)}$$

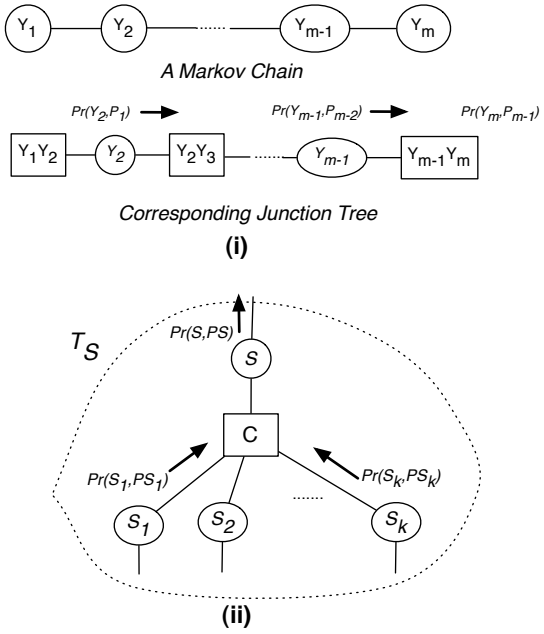


Figure 5.12: (i) A Markov chain, and the corresponding junction tree; (ii) Illustrating the recursion for general junction trees.

Using $\Pr(Y_{j+1}, Y_j, PS_{j-1})$, we can compute:

$$\begin{aligned}\Pr(Y_{j+1}, PS_j = a) &= \Pr(Y_{j+1}, Y_j = 0, PS_{j-1} = a) \\ &\quad + \Pr(Y_{j+1}, Y_j = 1, PS_{j-1} = a - \delta_j)\end{aligned}$$

At the end, we have the joint distribution: $\Pr(Y_m, PS_{m-1})$. We can compute a distribution over PS_m as:

$$\begin{aligned}\Pr(PS_m = a) &= \Pr(Y_m = 0, PS_{m-1} = a) \\ &\quad + \Pr(Y_m = 1, PS_{m-1} = a - \delta_m)\end{aligned}$$

Complexity The complexity of the above algorithm to compute $\Pr(PS_m)$ is $O(m^2)$ – although we only perform m steps, $\Pr(Y_{j+1}, PS_j)$ contains $2(j+1)$ terms, each of which takes $O(1)$ time to compute. Since we have to repeat this for every tuple, the overall complexity of ranking the dataset can be seen to be $O(n^3)$.

5.4.3 General Junction Trees

We follow the same general idea for general junction trees. Let $\mathcal{T}$ denote the junction tree over the variables $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$. We begin by rooting $\mathcal{T}$ at an arbitrary clique, and recurse down the tree. For a separator S, let $\mathcal{T}_S$ denote the subtree rooted at S. Denote by PS_S the partial sum over the variables in the subtree $\mathcal{T}_S$ that are *not present* in S, i.e.,:

$$PS_S = \sum_{j \in \mathcal{T}_S, j \notin S} \delta_j X_j$$

Consider a clique node C, and let S denote the separator between C and its parent node ($S = \phi$ for the root clique node). We will recursively compute the joint probability distribution $\Pr(S, PS_S)$ for each such separator S. Since the root clique node has no parent, at the end we are left with precisely the probability distribution that we need, i.e., $\Pr(\sum_{j=1}^{m} Y_i \delta_i)$.

<u>C **is an interior or root node**</u> Let the separators to the children of C be $S_1, \ldots, S_k$ (see Figure 5.12(ii)). We recursively compute $\Pr(S_i, PS_{S_i}), i = 1, \ldots, k$.

Let $Z = C \setminus S$. We observe that Z is precisely the set of variables that contribute to the partial sum PS_S, but do not contribute to any of the partial sums $PS_{S_1}, \ldots, PS_{S_k}$, i.e.:

$$PS_S = PS_{S_1} + \cdots + PS_{S_k} + \sum_{Z_i \in Z} \delta_{Z_i} Z_i$$

We begin with computing $\Pr(C, PS_{S_1} + \cdots + PS_{S_k})$. Observe that the variable set $C \setminus S_1$ is independent of PS_{S_1} given the values of the variables in S_1 (by Markov property). Note that it was critical that the variables in S_1 not contribute to the partial sum PS_{S_1}, otherwise this independence would not hold. Given that, we

have:

$$\begin{aligned}\Pr(C, PS_{S_1}) &= \Pr(C \setminus S_1, S_1, PS_{S_1}) \\ &= \frac{\Pr(C \setminus S_1, S_1)\Pr(S_1, PS_{S_1})}{\Pr(S_1)}\end{aligned}$$

Using PS_{S_2} is independent of $C \cup \{PS_{S_1}\}$ given S_2, we get:

$$\Pr(C, PS_{S_1}, PS_{S_2}) = \frac{\Pr(C, PS_{S_1})\Pr(S_2, PS_{S_2})}{\Pr(S_2)}$$

Now we can compute the probability distribution over $\Pr(C, PS_{S_1} + PS_{S_2})$ as follows:

$$\Pr(C, PS_{S_1} + PS_{S_2} = a) = \sum_{j=0}^{a} \Pr(C, PS_{S_1} = j, PS_{S_2} = a - j)$$
$$= \sum_{j=0}^{a} \frac{\Pr(C, PS_{S_1} = j)\Pr(S_2, PS_{S_2} = a - j)}{\Pr(S_2)}$$

By repeating this process for S_3 to S_k, we get the probability distribution: $\Pr(C, PS_{S_1} + \cdots + PS_{S_k})$.

Next, we need to add in the contributions of the variables in Z to the partial sum $PS_{S_1} + \cdots + PS_{S_k}$. Let Z contain l variables, $Z_1, \ldots Z_l$, and let $\delta_{Z_1}, \ldots, \delta_{Z_l}$ denote the corresponding indicator variables. It is easy to see that:

$$\Pr(C \setminus Z, Z_1 = v_1, \ldots, Z_k = v_k, \sum_{j=1}^{k} PS_{S_j} + \sum_{j=1}^{l} \delta_{z_j} Z_j = a)$$
$$= \Pr(C \setminus Z, Z_1 = v_1, \ldots, Z_k = v_k, \sum_{j=1}^{k} PS_{S_j} = a - \sum_{l=1}^{l} \delta_{z_j} Z_j)$$

where $v_i \in \{0, 1\}$. Although it looks complex, we only need to touch every entry of the probability distribution $\Pr(C, PS_1 + \cdots + PS_k)$ once to compute $\Pr(C, PS_S)$.

All that remains is marginalizing that distribution to sum out the variables in

$C \setminus S$, giving us $\Pr(S, PS_S)$.

C is a leaf node (i.e., $k = 0$) This is similar to the final step above. Let $Z = C \setminus S$ denote the variables that contribute to the partial sum PS_S. We can apply the same procedure as above to compute $\Pr(C, PS_S = \sum_{Z_i \in Z} \delta_{Z_i} Z_i)$, which we marginalize to obtain $\Pr(S, PS_S)$.

Overall Complexity The complexity of the above algorithm for a specific clique C is dominated by the cost of computing the different probability distributions of the form $\Pr(C, PS)$, where PS is a partial sum. We have to compute $O(n)$ such probability distributions, and each of those computations takes $O(n^2 2^{|C|})$ time. Since there are at most n cliques, and since we have to repeat this process for every tuple, the overall complexity of ranking the dataset can be seen to be: $O(n^4 2^{tw})$, where tw denotes the treewidth of the junction tree, i.e., the size of the maximum clique minus 1.

Chapter 6

Computing PRF: Continuous Distributions

Continuous attribute uncertainty models arise naturally in many domains. Prior work on ranking in probabilistic databases (or more generally query processing in probabilistic databases with some exceptions) has mostly proposed somewhat simplistic solutions to this problem. Cormode et al. [48] suggested to discretize the continuous distributions to an appropriate level of granularity, and thus reduce the problem to discrete attribute uncertainty [48, 123]. Soliman et al. made the first attempt to deal with continuous score distributions directly [168], however, their main technical tool is the Monte Carlo method which, in most cases, can only obtain an approximate solution.

In this chapter, we systematically 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. [174] and Guiver et al. [91] 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. They only consider Gaussian distributions and present heuristics to compute the rank distributions. We consider many different types of probability distribution functions, and present exact or approximate solutions depending on the functions.

In Section 6.1, we develop exact polynomial time algorithms for uniform and piecewise polynomial distributed scores. In Section 6.2, we present an efficient

approximation schemes with provable guarantees for arbitrary probability distributions based on the exact algorithm for piecewise polynomial distributions. We also show how to efficiently rank the dateset by E-Rank or PRF^ℓ in Section 6.3. In Section 6.4, we show an application of our algorithms for PRF to answering the k-nearest neighbor query in uncertain datasets We show our experimental results at the end of this chapter.

6.1 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 developed in Section 5.1.1 can be extended to handle continuous distributions.

We first introduce some necessary notations. 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 $s(t_i)$ (or 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. 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: $\rho_i(\ell) = \Pr(s_i \leq \ell) = \int_{-\infty}^{\ell} \mu_i(x)\mathrm{d}x$. Let $\bar{\rho}_i(\ell) = 1 - \rho_i(\ell)$. The notations are summarized in Table 6.1.

$p(t_i)$ or p_i	Existence prob. of t_i
$s(t_i)$ or 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 $\rho_i(\ell) = \int_{-\infty}^{l} \mu_i \mathrm{d}\ell,\ \ \bar{\rho}_i(\ell) = 1 - \rho_i(\ell)$
$\Pr(r(t_i) = j)$	Positional prob. of t_i being ranked at position j
$I = [l_I, u_I]$	A small interval and its range

Table 6.1: Notation

6.1.1 Generating Functions Framework

In this and the next subsection, we consider only attribute value uncertainty, i.e., we assume there is no tuple existence uncertainty. Let us begin with looking at the formula for computing positional probability $\Pr(r(t_i) = j)$ closely. First, we observe 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:

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

The last equality follows from independence. The following theorem provides an explicit form of the *generating function* for the positional probabilities and plays a central role in our algorithms.

Theorem 6.1. *For any tuple t_i, define*

$$F_i(x) = x \int_{-\infty}^{\infty} \mu_i(\ell) \prod_{j \neq i} \Big(\rho_j(\ell) + \bar{\rho}_j(\ell) x\Big) \mathrm{d}\ell \tag{6.1}$$

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

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

Proof: First, we note that $F_i(x)$ defined in (6.1) 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 us consider how to compute $\Pr\Big(\sum_{j\neq i} \delta(s_j > \ell) = j\Big)$ 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 $\bar{\rho}_j(\ell) = \Pr(s_j > \ell)$, compute $\Pr(j$ tuples exist). Consider the following generating function:

$$\mathcal{F}_i(x, \ell) = \prod_{j\neq i} \Big(\rho_j(\ell) + \bar{\rho}_j(\ell)x\Big). \tag{6.2}$$

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 $\Pr\Big(\sum_{j\neq i} \delta(s_j > \ell) = j\Big)$ (see Section 5.1). Thus, we can write:

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

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

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

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

In light of Theorem 6.1, we can see that the task of computing the positional probabilities reduces to expanding the polynomial $\digamma_i$ in terms of x^js and obtaining the coefficients.

6.1.2 Uniform Distribution

In this section, we consider the case where μ_i is uniform over its *support interval* $[l_i, u_i]$. It is easy to see the cdf of s_i is a piecewise linear function, i.e.,

$$\rho_i(\ell) = \Pr(s_i \leq \ell) = \begin{cases} 0, & \ell < l_i; \\ \frac{\ell - l_i}{u_i - l_i}, & l_i \leq \ell \leq u_i; \\ 1, & \ell > u_i. \end{cases}$$

6.1.2.1 Expanding $\digamma(x)$

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 6.1 for an illustrative example with 5 tuples). For convenience of exposition, we call these intervals *small intervals* (in contrast to the *support intervals* $[l_i, u_i]$).

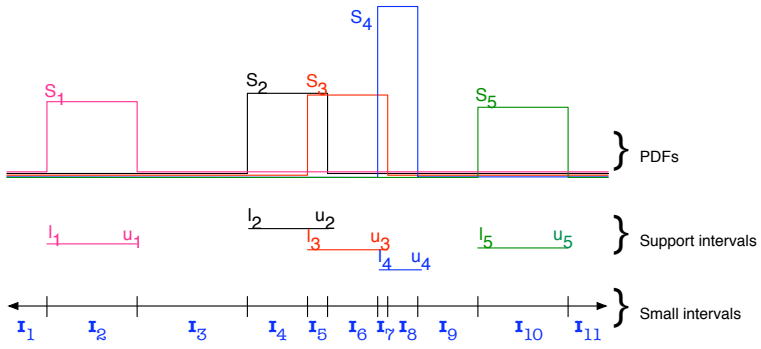


Figure 6.1: Illustration of support intervals and small intervals for five tuples with uniform probability distributions

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 12. *In the example shown in Figure 6.1,* $\mathcal{I}_2 = \{I_4, I_5\}$, *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:

$$F_i(x) = x \sum_{I \in \mathcal{I}_i} \int_{l_I}^{u_I} \mu_i(l) \prod_{j \neq i} \Big(\rho_j(\ell) + \bar{\rho}_j(\ell) x\Big) d\ell \tag{6.3}$$

$$= \frac{x}{u_i - l_i} \sum_{I \in \mathcal{I}_i} \int_{l_I}^{u_I} \mathcal{F}_i(x, \ell) \mathrm{d}\ell \tag{6.4}$$

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:

$$(a_{I,j}, b_{I,j}, c_{I,j}, d_{I,j}) = \begin{cases} (1,0,0,0), & u_I \le l_j; \\ \left(\frac{-l_j}{u_j - l_j}, \frac{1}{u_j - l_j}, \frac{u_j}{u_j - l_j}, \frac{-1}{u_j - l_j}\right), & I \in \mathcal{I}_j; \\ (0,0,1,0), & l_I \ge u_j. \end{cases} \tag{6.5}$$

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:

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

The last equality is because: $\int_{l_I}^{u_I} \ell^k \mathrm{d}\ell = \frac{1}{k+1}(u_I^{k+1} - l_I^{k+1})$.
Summing over all intervals in $\mathcal{I}_i$, we get

$$\int_{l_i}^{u_i} \mathcal{F}_i(x,\ell)d\ell = \sum_{I \in \mathcal{I}_i} \int_I \mathcal{F}(x,\ell)d\ell = \sum_j \Big(\sum_{I \in \mathcal{I}_i} \sum_k \frac{\alpha_{I,i,j,k}}{k+1}\big(u_I^{k+1} - l_I^{k+1}\big)\Big)x^j$$

Finally, combining with Theorem 6.1 and (6.3), we get

$$\begin{aligned} \Pr(r(t_i) = j) &= \frac{1}{u_i - l_i} \int_{l_i}^{u_i} \Pr\Big(\sum_{j \neq i} \delta(s_j > \ell) = j\Big)\mathrm{d}\ell \\ &= \frac{1}{u_i - l_i} \sum_{I \in \mathcal{I}_i} \sum_k \frac{\alpha_{I,i,j,k}}{k+1}\big(u_I^{k+1} - l_I^{k+1}\big) \end{aligned} \tag{6.6}$$

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

Table 6.2: Running Time. 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 on small interval I_j.

6.1.2.2 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

$$\tilde{\mathcal{F}}_{I_j}(x,\ell) = \prod_{j=1}^{n}\Big(\rho_j(\ell) + \bar{\rho}_j(\ell)x\Big) = \prod_{j\in M_j'} 1 \prod_{j\in M_j}\Big(\rho_j(\ell) + \bar{\rho}_j(\ell)x\Big)\prod_{j\in M_j''} x$$
$$= \prod_{j\in M_j}\Big(\frac{-l_j + \ell + u_j x - x\ell}{u_j - l_j}\Big)x^{m_j''} \tag{6.7}$$

From (6.2) and (6.5), we can see that on interval I_j,

$$\mathcal{F}_i(x,\ell) = \tilde{\mathcal{F}}_{I_j}(x,\ell)\frac{u_i - l_i}{-l_i + \ell + u_i x - x\ell}. \tag{6.8}$$

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 6.8, which needs $O(m_j^2)$ time. We summarize the overall steps in Algorithm 4. 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 4: 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 (6.8);  (Note that we can obtain
        coefficients α_{I_t,i,j,k}s in this step);
Compute Υ(t_i) according to (6.6);
Return k tuples with largest |Υ| values;
```

6.1.3 Extensions

We first show how to improve the running time of the above algorithm for some important special cases. We then extend the basic algorithm to handle tuple uncertainty, and piecewise polynomial distributions.

Computing $\mathsf{PRF}^{\omega}(h)$: Since $\omega(j) = 0$ for all $j > h$, we only need the probability values $\Pr(r(t_i) = j)$ for $j \leq h$. Therefore, we only need to expand $F_i(x)$ up to the x^h term. See Table 6.2 for the running time. Since h is typically much smaller than the number of tuples n, the improvement can be significant.

Computing $\mathsf{PRF}^{\mathsf{e}}$: As in Section 5.1.3, we have the same relationship between the generating function and the $\mathsf{PRF}^{\mathsf{e}}(\alpha)$ value:

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

Therefore, 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. Manipulating polynomials with a single variable can be done much faster than with two variables. See Table 6.2 for the exact running time.

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: $\rho_i(\ell) = \Pr(t_i \text{ does not exist or } s_i \leq \ell) = (1 - p_i) + p_i \int_{-\infty} \mu_i(x)\mathrm{d}x$ and still let $\bar{\rho}_i(\ell) = 1 - \rho'_i(\ell)$. It can be seen that Theorem 6.1 still holds (we omit the proof due to space constraints). Therefore, all algorithms developed can be applied with the new definitions.

The running time is reported in Table 6.2. We notice that, although the algorithms are almost the same, the running time may be a bit higher than the case without tuple uncertainty. The reason is that a tuple may contribute non-

trivially to another tuple's generating function even if their supports do not overlap.

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 and algorithms 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 6.2 shows an example of this where the pdf is expressed using 6 different polynomials, two of which are 0 (this piecewise polynomial is also a very good approximation to a Gaussian distribution).

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 every term inside the integral in (6.1) 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 similar trick to (6.8) to improve the running time. See Table 6.2 for the exact running time.

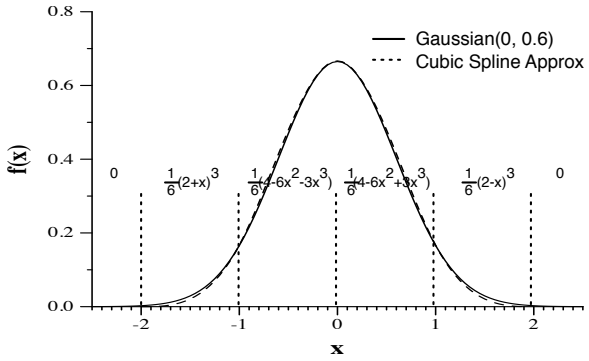


Figure 6.2: Approximating a Gaussian distribution using a Cubic Spline with 6 pieces (e.g. in the interval $[-2, -1]$, the approximation is done using $\frac{1}{6}(2+x)^3$).

6.2 Arbitrary Probability Densities

For arbitrary probability density functions, the term inside the integral of (6.1) 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. Finally, we propose a highly efficient approximation algorithm to compute PRF^e using *Legendre-Gauss Quadrature*.

6.2.1 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 $l \leq x \leq u$. We choose k breaking points τ_i such that $l = \tau_1 < \tau_2 < \ldots < \tau_k = u$. We assume for now $\tau_{i+1} - \tau_i = \frac{u-l}{k-1}$ 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.,

$$\begin{aligned} P_i(\tau_i) = \mu(\tau_i), \quad \frac{\mathrm{d}P_i}{\mathrm{d}x}(\tau_i) = \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i), \\ P_i(\tau_{i+1}) = \mu(\tau_{i+1}), \quad \frac{\mathrm{d}P_i}{\mathrm{d}x}(\tau_{i+1}) = \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_{i+1}). \end{aligned}$$

It can be shown that (see e.g. [57, 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:

$$\begin{aligned} c_{i,1} &= \mu(\tau_i), c_{i,2} = \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i), \\ c_{i,4} &= \frac{1}{(\tau_{i+1} - \tau_i)^2} \left(\frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i) + \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_{i+1}) - 2\frac{\mu(\tau_i) - \mu(\tau_{i+1})}{\tau_i - \tau_{i+1}} \right) \\ c_{i,3} &= c_{i,4}(\tau_{i+1} - \tau_i) + \frac{1}{\tau_{i+1} - \tau_i} \left(\frac{\mu(\tau_i) - \mu(\tau_{i+1})}{\tau_i - \tau_{i+1}} - \frac{\mathrm{d}\mu}{\mathrm{d}x}(\tau_i) \right) \end{aligned}$$

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 6.5. 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.

6.2.2 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 distribution has been suggested in [48], however, no further detail and analysis is provided. In this paper, we consider

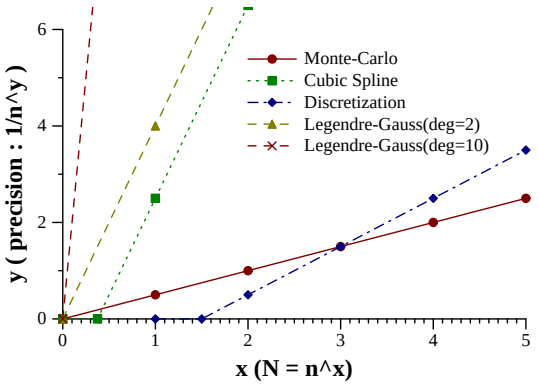


Figure 6.3: The asymptotic precision-complexity trade-offs for various methods. Note the meaning of the axis: $N = n^x$, precision is of order $1/n^y$. All constants hidden in big O are ignored.

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 $\Pr(t_{i,j}) = \int_{I_j} \mu_i(x)\mathrm{d}x$.

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 ourselves to the cubic spline approximations.

Theorem 6.2. *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* $\beta > 3/8$ *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 6.1, then*

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

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. [83, 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 $\psi(0), \ldots, \psi(n-1)$ denote the discrete Fourier transform of $\omega(0), \ldots, \omega(n-1)$. Then $\sum_{i=0}^{n-1} |\psi(i)| \leq n^{3/2}$.*

Proof:

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

The first inequality is the the CauchySchwarz 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 $\sum_{i=0}^{n} |\omega(i)|^2 = \frac{1}{n}\sum_{i=0}^{n} |\psi(i)|^2$. □

Proof of Theorem 6.2: Let $\widehat{\mu}_i$ be the approximated distribution of s_i for each i. Let $\bar{\rho}_i(\ell) = \Pr(s_i > \ell) = \int_\ell^\infty \mu_i(x)\mathrm{d}x$, $\rho_i(\ell) = 1 - \bar{\rho}_i(\ell)$, $\widehat{\rho}_i(\ell) = \int_\ell^\infty \widehat{\mu}_i(x)\mathrm{d}x$ and $\widehat{\varrho}_i(\ell) = 1 - \widehat{\rho}_i(\ell)$. It is known that (e.g. [57, 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 $\psi(0), \ldots, \psi(n-1)$ denote the discrete Fourier transform of $\omega(t,0), \ldots, \omega(t,n-1)$. 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 $\mathsf{PRF}^{\mathsf{e}}$ value of t with parameter $e^{\frac{2\pi\jmath}{n}k}$ by $\Upsilon_k(t)$. Therefore, we have

$$\Upsilon_\omega(t) = \frac{1}{n}\sum_{k=0}^{n-1} \psi(k)\Upsilon_k(t). \tag{6.9}$$

Now, we analyze the approximation error for the approximated $\mathsf{PRF}^{\mathsf{e}}$ value with any parameter α such that $|\alpha| = 1$. Since the $\mathsf{PRF}^{\mathsf{e}}$ 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. 6.1) 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 $\ell \in \mathbb{R}$, $|\rho_j(\ell) + \bar{\rho}_j(\ell)\alpha| \leq |\rho_j| + |\bar{\rho}_j(\ell)\alpha| = 1$. Also,

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

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 (6.9) and Lemma 3, we obtain

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

This completes the proof of the theorem. □

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 behavior for the discretization method.

Theorem 6.3. *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 [136, 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 6.4. *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*

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

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 6.2, 6.3 and 6.4, 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 (use an and/xor tree to represent the attribute uncertainty and then apply the algorithm in Section 5.1.2) 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 6.3 for a clearer illustration of the trade-off.

In many applications, very high precision is often required. Now, we give a contrived but still simple example. 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., $\Pr(r(t) = 1)$ (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 $\Upsilon(t_i) = \Pr(t_i = 1)$ for all $1 \leq i \leq 10$ are in an order of magnitude $O(10^{-11})$, and it is likely that the Monte Carlo estimates for them are all zero with even $O(10^9)$ samples. Therefore, in order to get a relatively accurate estimate, an astronomical number of samples are needed. On the other hand, by partitioning $[-10, 10]$ into 10^5 small intervals (let $\beta = 4$), the spline approximation can give us an estimate with error in an order of $O(10^{-14})$ by Theorem 6.2, which should be fairly good estimates of $\Upsilon(t_i)$.

Now, we discuss the assumptions we made for Theorem 6.2 and Theorem 6.3. For some distributions, for example, the Gaussian distribution, the support is not bounded. However, in many cases, 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, like what we did in the previous example, i.e., we truncated Gaussian at ± 10. The assumption $|\mathsf{supp}(\mu_i)| = O(1)$ 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 satisfied in practice.

In the end, we would like to remark that all analyses done in this section are worst case analyses and 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. [52] [1]).

6.2.3 Approximating $\mathsf{PRF}^{\mathsf{e}}(\alpha)$ by Legendre-Gauss Quadrature for $\alpha \in \mathbb{R}$

As we discussed in Section 6.1.3, the $\mathsf{PRF}^{\mathsf{e}}(\alpha)$ value of tuple t_i has a closed form expression, which is the value of the generating function (6.1) evaluated at α, i.e., $F_i(\alpha)$. For arbitrary distributions, we 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 $f(\ell) = \prod_{j \neq i}(\rho_j(\ell) + \bar{\rho}_j(\ell)\alpha)\mu(\ell)$, which we

[1] Actually, the aim of [52] is to obtain an estimation with a relative error. It is straightforward to translate the result in terms of additive error. However, the worst case is the same as in Theorem 6.4.

can evaluate at any point ℓ in polynomial time[2]. Given this, we can use Legendre-Gauss quadrature, an existing numerical integration method, to achieve a much more efficient approximation of $\mathsf{PRF}^{\mathsf{e}}(\alpha)$ for real α. For completeness, we briefly describe the integration technique next.

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 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:

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

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 [170] and we can use these values directly.

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

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

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

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 Legendre-Gauss quadrature 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) [150, 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. See Figure 6.3 for the asymptotic Error-N (precision-complexity) trade-offs. In our experimental study, we use Legendre-Gauss quadrature of degree 10.

6.3 Expected Ranks and PRF^l

Recall that PRF$^\ell$ 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$^\ell$ is its close relationship to *expected ranks* (Section 4.3). From the definition, we can see that:

$$\mathsf{PRF}^\ell(t) = -\sum_{i>0} i \times \Pr(r(t) = i) + np(t)$$

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

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

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

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 6.1.2, 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 $\sum_{j>0} i \Pr(r(t_i) = j)$ 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 $\Pr(s_j \geq s_i)$. Also, it is well known that $s_j - s_i \sim \mathcal{N}(\lambda_j - \lambda_i, \sigma_j^2 + \sigma_i^2)$. Therefore,

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

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 [2]. It is a built-in function in many programming languages now-a-days. Therefore, it is very reasonable to assume that it can be computed in $O(1)$ time even though it does 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 $\Pr(s_j \geq s_i)$ was also observed by Cormode et al. [48], who use it to to obtain algorithms for discrete distributions.

We can generalize this algorithm to handle *convex combinations of Gaussians*. We refer the reader to the extended version of the paper for details.

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:

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

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 $O(\gamma \sum_j m_j + \sum_j m_j(\gamma^2+\gamma)) = O(\gamma^2 \sum_j m_j)$.

[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 which takes $O(\gamma^2)$ time.

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.

6.4 Application to Probabilistic k-Nearest Neighbor

The nearest neighbor (NN) and k-nearest neighbor queries (k-NN) are of great importance on both graphs and relations. Given a distance or dissimilarity function and a query point q, the NN (or k-NN) query returns the node (or the k nodes) that is closest to q according to the distance function. Both queries have been extended to probabilistic datasets in recent years [121, 42, 23, 44, 148].

In this section, we briefly sketch how to apply our algorithms for PRF to *probabilistic nearest neighbor* (Prob-NN) and *probabilistic k-nearest neighbor* (Prob-k-NN) queries over uncertain objects. For generality, we only consider Prob-k-NN since Prob-NN is just special case of Prob-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* $p_i : \mathbb{R}^d \rightarrow \mathbb{R}^+$ 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 6. *Given a query point $q \in \mathbb{R}^d$, a* Prob-k-NN *query retrieves k objects that have highest P_{knn} values where $P_{knn}(t_i, q) = \Pr(t_i \in kNN(q))$.*

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. [121] and Cheng et al. [42] 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. [23]

studied exactly the above query with $k = 1$. Cheng et al. [44] 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 [167]). This is not captured by the above definition (and cannot be captured using a PRF function either).

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

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

and the pdf of t_i's score being $\mu_i(x) = \Pr(\mathsf{dis}(t_i, q) = x)$. If p is a deterministic point, all μ_is are independent and we can apply our exact or approximate algorithms developed in Section 6.1 and 6.2 directly, depending on the type of the probability distributions μ_is.

Example 13. *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 $q \geq l_i$ or $q \leq u_i$, 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 6.1.*

Beskales et al. [23] 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 cannot be directly applied since the probabilities $\Pr(\mathsf{dis}(t_{i,q}) = x)$ are correlated for different objects t_i. Theoretically, we can generalize the generating function technique we developed in Section 6.1 to handle this special correlation. However, this may introduce integration in higher dimensional space. How to handel such integration is an interesting research challenge. 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.

6.5 Experimental Study

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 certain 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 6.1.2) 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 6.2.2). 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 Section 5.1.2 to compute PRF value for x-tuples.
- Mc: We run the Monte Carlo method (outlined in Section 6.2.2) 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 [117].

6.5.1 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 6.4(a), when the granularity is finer than 0.5, Spline converges to

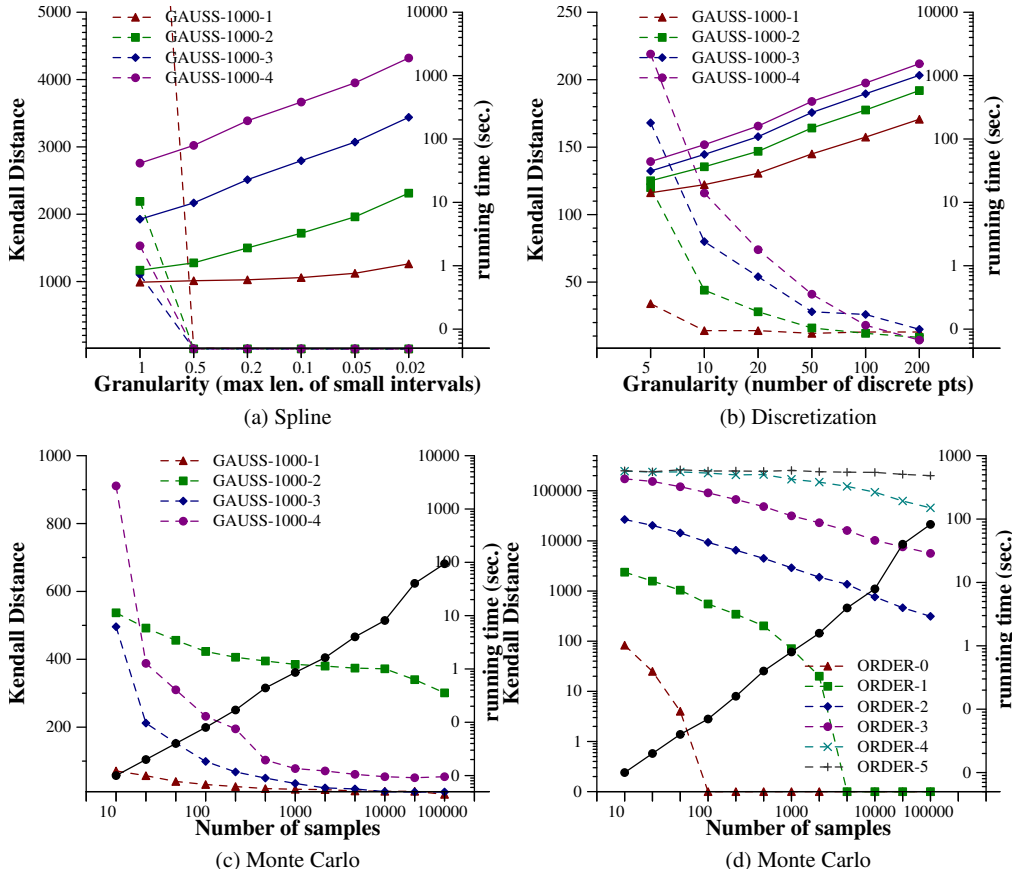


Figure 6.4: The comparison of various methods for computing general PRF (weight function $\omega(t, j) = 1/j$). Solid lines indicate the running times (with axes drawn on the right hand side), whereas dashed lines indicate the Kendall distance (an error measure).

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 also note that the running time of Spline depends heavily on the overlap numbers.

Figure 6.4(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 6.4(b), we can see that the convergence rate of Disc is slower than Spline, but much faster than Mc. We can see that 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 6.4(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 a few other weight functions ω, such as piecewise linear function, and observed similar behaviors. We omit those curves due to space constraints.

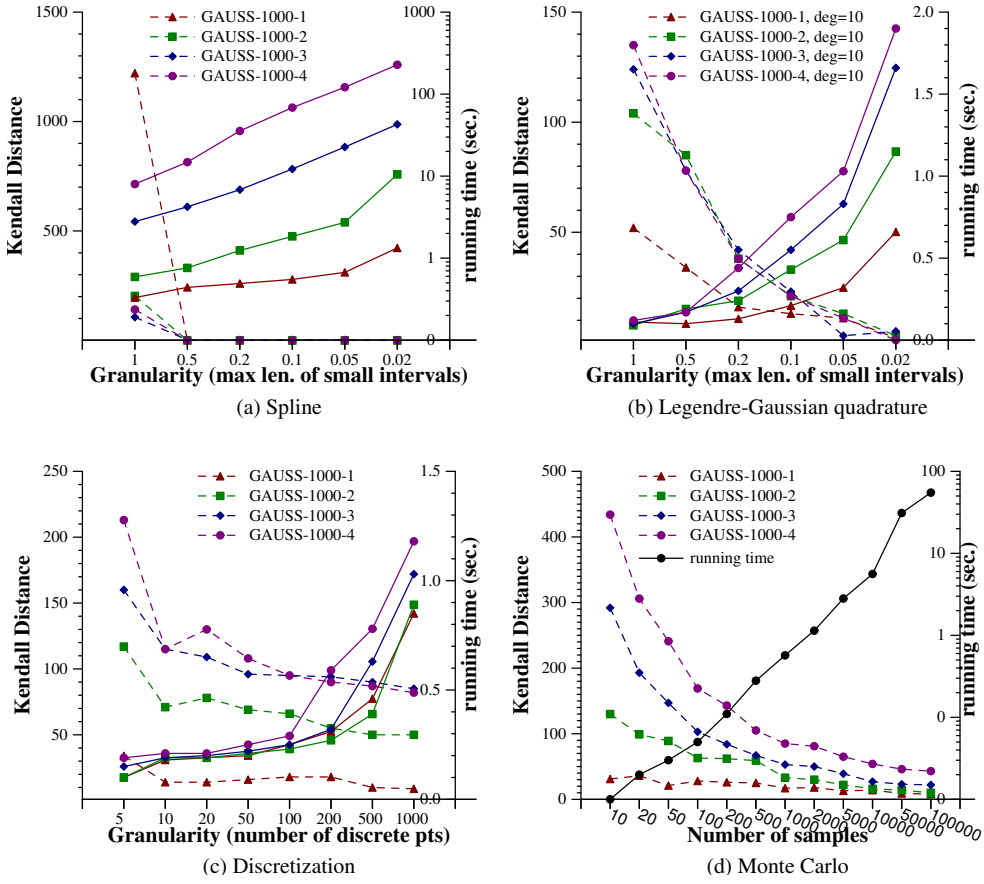


Figure 6.5: The comparison of various methods for computing $\mathsf{PRF}^{\mathsf{e}}(\alpha = 0.99)$. Solid lines indicate the running times (with axes drawn on the right hand side), whereas dashed lines indicate the Kendall distance (an error measure).

6.5.2 LG Quadrature vs. Monte Carlo vs. Discretization for $\mathsf{PRF}^{\mathsf{e}}$

We compared the four techniques for $\mathsf{PRF}^{\mathsf{e}}$ computation: (1) Spline (Spline), (2) Legendre-Gauss Quadrature (Lgq), (3) Monte Carlo (Mc), and (4) Discretization (Disc). 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 $\mathsf{PRF}^{\mathsf{e}}$.

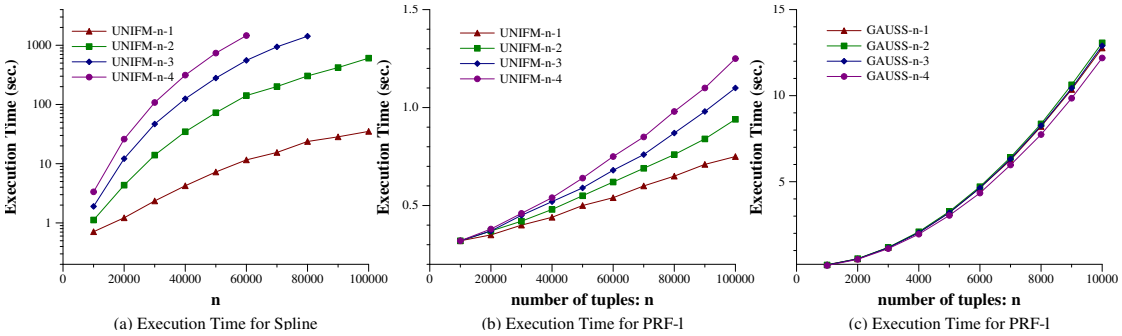
Figure 6.5(a),(b),(c) and (d) show the execution times and convergence rates for Spline-e, Lgq, Disc-e, and Mc, respectively. The "true" ranking is computed

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 6.5(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 6.5(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.

6.5.3 Execution Times for Exact Algorithms

Figure 6.6(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 speaking, 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



(a) Execution Time for Spline (b) Execution Time for PRF-l (c) Execution Time for PRF-l

Figure 6.6: Execution times for (a) Spline on UNIFM-datesets, PRF^ℓ on UNIFM-datasets and (b)PRF^ℓ on GAUSS-datasets.

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 6.3 for the details of the algorithm), the execution time is faster than the general Spline method. Figure 6.6(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 (running time is $O(\sum_j m_j)$) and is independent of d on GAUSS-datasets (running time is $O(n^2)$).

Chapter 7

Computing Consensus Answers

Recall that in the possible worlds semantics, a probabilistic database is considered to correspond to a probability distribution over a set of deterministic databases called possible worlds. Dierent possible worlds may generate dierent top-k answers, which signicantly complicate the semantics of the top-k queries over probabilistic databases, as we have already seen from previous chapters. In fact, many other queries on probabilistic databases have similar semantical issues. One approach to addressing this issue is to "combine" the possible answers somehow to obtain a more compact representation of the result. We note that for select-project-join queries, for instance, one proposed approach is to union all the possible answers, and compute the probability of each result tuple by adding the probabilities of all the possible answers it belongs to [54]. This approach, however, cannot be easily extended to top-k queries or other types of queries like aggregate queries.

In this chapter, we propose another systematic way to to combine the top-k answers for all possible worlds by putting it in the context of *inconsistent information aggregation* which has been studied extensively in numerous contexts over the last half century. In our context, the set of different query answers returned from possible worlds can be thought as inconsistent information which we need to aggregate to obtain a single representative answer. Concretely, we propose the notion of *the consensus answer*. Roughly speaking, the consensus answer is a answer that is *closest in expectation* to the answers of the possible worlds. To measure the closeness of two answers τ_1 and τ_2, we need to define suitable distance function $\mathsf{dis}(\tau_1, \tau_2)$ over the answer space. If the most consensus answer can be taken from any point in the answer space, we refer it as the *mean answer*. A *median answer*,

on the other hand, must be the answer for some possible world with non-zero probability. We use CON as a shorthand notation to denote the mean answer for top-k queries.

We briefly summarize the algorithmic results in this chapter. The problem of aggregating inconsistent rankings has been well-studied under the name of *rank aggregation* [64]. We develop polynomial time algorithms for computing mean and median top-k answers under the symmetric difference metric, and the mean answers under *intersection metric* and *generalized Spearman's footrule distance* [66] [1], for the and/xor tree model. If the query results are sets and we use the set difference metrics as the distance function, we show that the mean and the median answers can be found in polynomial time for the *symmetric difference* metric for and/xor tree model. For the Jaccard distance metric, we present a polynomial time algorithm to compute the mean and median world for a tuple independent database. Then, we study a specific type of *group by count* queries and present a 4-approximation to the problem of finding a median answer (finding mean answers is trivial). We also consider the consensus clustering problem for the and/xor tree model and get a constant approximation by extending a previous result [8].

We first formally define the consensus answers in Section 7.3.3. Then, we develop efficient exact or approximate algorithms for computing consensus top-k answers under different distance functions in Section 7.2. Lastly, we consider some other types of queries, such as SPJ queries and aggregate queries, in Section 7.3.

7.1 Consensus Answers

We formally define the most consensus answers in this section. We first set up some notations. Suppose we have n tuples $t_1, \ldots, t_n$, where tuple t_i has a score $s(t_i)$. In the tuple-level uncertainty model, $s(t_i)$ is fixed for each t_i, while in the attribute-level uncertainty model, it is a random variable. In the and/xor tree

[1] We have reviewed these distance functions in Section 2.5.

model, we assume that the attribute field is the score (uncertain attributes that don't contribute to the score can be ignored). We further assume no two tuples can take the same score for avoiding ties. We use $r(t)$ to denote the random variable indicating the rank of t and $r_{pw}(t)$ to denote the rank of t in possible world pw. If t does not appear in the possible world pw, then $r_{pw}(t) = \infty$. So, $\Pr(r(t) > i)$ includes the probability that t's rank is larger than i and that t doesn't exist. We say t_1 *ranks higher* than t_2 in possible world pw if $r_{pw}(t_1) < r_{pw}(t_2)$. We use the symbol τ to denote a top-k ranked list, and τ^i to denote the restriction of τ to the first i items. We use $\tau(i)$ to denote the i^{th} item in the list τ for positive integer i, and $\tau(t)$ to denote the position of $t \in T$ in τ.

We denote the domain of answers for a query by Ω and the distance function between two top-k answers by $\mathsf{dis}()$.

Definition 7. *The* most consensus answer τ *is defined to be a feasible query answer such that the expected distance between* τ *and the answer* τ_{pw} *of the (random) world* pw *is minimized, i.e,*

$$\tau = \arg\min_{\tau' \in \Omega} \{\mathbb{E}[\mathsf{dis}(\tau', \tau_{pw})]\}.$$

We call the most consensus answer in Ω *the mean answer* when Ω is the set of all feasible answers. If Ω is restricted to be the set of possible answers (answers of some possible worlds with non-zero probability), we call the most consensus answer in Ω *the median answer*. Taking the example of the top-k queries, the median answer must be the top-k answer of some possible world while the mean answer can be any sorted list of size k. $\mathsf{dis}()$ can be any distance function discussed in the last section. We use CON to denote the query that asks for the mean answer for a top-k query on a probabilistic database.

Example 14. *Consider the example in Figure 2.2. Assume* $k = 2$ *and the distance function is the symmetric difference metric* $\mathsf{dis}_\Delta = |(\tau_1 \backslash \tau_2) \cup (\tau_2 \backslash \tau_1)|$. *The most consensus top-*2 *answer is* $\{t_2, t_5\}$ *and the expected distance is* $\mathbb{E}[\mathsf{dis}(\tau', \tau_{pw})] = .112 \times 2 + .168 \times 2 + .048 \times 4 + .072 \times 4 + .168 \times 2 + .252 \times 0 + .072 \times 4 + .108 \times 2$.

We first show that a CON answer under symmetric difference is equivalent to PT(k), a special case of PRF$^\omega$. Then, we generalize the result and show that any PRF$^\omega$ function is in fact equivalent to some CON answer under some suitably defined distance function that generalizes symmetric difference. This new connection further justifies the semantics of PRF$^\omega$ from an optimization point of view in that the top-k answer obtained by PRF$^\omega$ minimizes the expected value of some distance function, and it may shed some light on designing the weight function for PRF$^\omega$ in particular applications. We also consider the problem of evaluating consensus answers for other distance metrics.

7.2 Algorithms for Different Metrics

7.2.1 Symmetric Difference and PT(k) Ranking Function

In this section, we show how to find mean and median top-k answers under symmetric difference metric in the and/xor tree model. Recall PT(k) query returns k tuples with the largest $\Pr(r(t) \leq k)$.

Theorem 7.1. *If $\tau = \{\tau(1), \tau(2), \ldots, \tau(k)\}$ is the set of k tuples with the largest $\Pr(r(t) \leq k)$, then τ is the mean top-k answer under metric dis_Δ, i.e, the answer minimizes $\mathbb{E}[\mathsf{dis}_\Delta(\tau, \tau_{pw})]$.*

Proof: Suppose τ is fixed. We write $\mathbb{E}[\mathsf{dis}_\Delta(\tau, \tau_{pw})]$ as follows:

$$\begin{aligned}
\mathbb{E}[\mathsf{dis}_\Delta(\tau, \tau_{pw})] &= \mathbb{E}[\sum_{t \in T} \delta(t \in \tau \wedge t \notin \tau_{pw}) + \delta(t \in \tau_{pw} \wedge t \notin \tau)] \\
&= \sum_{t \in T \setminus \tau} \mathbb{E}[\delta(t \in \tau_{pw})] + \sum_{t \in \tau} \mathbb{E}[\delta(t \notin \tau_{pw})] \\
&= \sum_{t \in T \setminus \tau} \Pr(r(t) \leq k) + \sum_{t \in \tau} \Pr(r(t) > k) \\
&= k + \sum_{t \in T} \Pr(r(t) \leq k) - 2 \sum_{t \in \tau} \Pr(r(t) \leq k)
\end{aligned}$$

The first two terms are invariant with respect to τ. Therefore, it is clear that the set of k tuples with the largest $\Pr(r(t) \leq k)$ minimizes the expectation. $\square$

To find a median answer, we essentially need to find the top-k answer τ of some possible world such that $\sum_{t \in \tau} \Pr(r(t) \leq k)$ is maximum. Next we show how to do this given an and/xor tree in polynomial time.

We write $P(t) = \Pr(r(t) \leq k)$ for ease of notation. We can't simply pick k tuples with the highest $P(t)$ values since some of them may be mutually exclusive. We use dynamic programming over the tree structure. For each possible attribute value $a \in A$ (A value is used to rank the tuples in the deterministic setting), let $\mathcal{T}^a$ be the tree which contains all leaves with attribute value at least a. We recursively compute the set of tuples $pw^a(v, i)$, which maximizes the value $\sum_{t \in pw^a(v,i)} P(t)$ among all possible worlds generated by the subtree $\mathcal{T}^a_v$ rooted at v and is of size i, for each node v in $\mathcal{T}^a$ and $1 \leq i \leq k$. We compute this for all different a values, and the optimal solution can be chosen to be $\max_a(pw^a(r, k))$.

Suppose $v_1, v_2, \ldots, v_l$ are v's children. The recursion formula is:

1. If v is a $\textcircled{\vee}$ node,

$$pw^a(v, i) = \arg \max_{pw \in PW(\mathcal{T}^a_{v_i})} \sum_{t \in pw} P(t) = \arg \max_{1 \leq j \leq l} pw^a(v_j, i).$$

2. If v is a $\textcircled{\wedge}$ node, $pw^a(v, i) = \cup_{1 \leq j \leq l} pw_j$ such that $pw_j \in PW(\mathcal{T}^a_{v_j}), \sum_j |pw_j| = i$ and $\sum_{t \in \cup_j pw_j} P(t)$ is maximized.

In the latter case, the maximum value can be computed by dynamic programming again as follows.

We denote by $pw^a([v_1 \ldots v_h], i)$ the set $\cup_{j=1}^h pw_j$ such that $pw_j \in PW(\mathcal{T}^a_{v_j}), \sum_{j=1}^h |pw_j| = i$ and $\sum_{t \in \cup_{j=1}^h pw_j} P(t)$ is maximized. $pw^a([v_1, \ldots v_h], i)$ can also be computed recursively. Let

$$p = \arg \max_{0 \leq p \leq i} \sum_{t \in pw^a([v_1 \ldots v_{h-1}], p) \cup pw^a(v_h, i-p)} P(t).$$

Then, we have

$$pw^a([v_1 \dots v_h], i) = pw^a([v_1 \dots v_{h-1}], p) \cup pw^a(v_h, 1-p).$$

Finally, it is easy to see $pw^a(v, i)$ is simply $pw^a([v_1, \dots, v_l], i)$.

Theorem 7.2. *The median top-k answer under symmetric difference metric can be found in polynomial time for a probabilistic and/xor tree.*

7.2.2 Weighted Symmetric Difference and PRF^ω

We present a generalization of Theorem 7.2.2, that is the equivalence between any $\mathsf{PRF}^\omega(k)$ function and CON under *weighted symmetric difference* distance functions which generalize the symmetric difference. Suppose ω is a positive function defined on $\mathbb{Z}^+$ and $\omega(i) = 0 \forall i > k$.

Definition 8. *The weighted symmetric difference with weight ω of two top-k answers τ_1 and τ_2 is defined to be*

$$\mathsf{dis}_\omega(\tau_1, \tau_2) = \sum_{i=1}^{k} \omega(i)\delta(\tau_2(i) \notin \tau_1).$$

Intuitively, if the i^{th} item of τ_2 cannot be found in τ_1, we pay a penalty of $\omega(i)$ and the distance is just the total penalty. If ω is a decreasing function, the distance function captures the intuition that top ranked items should carry more weight. If ω is a constant function, it reduces to the ordinary symmetric difference distance. Note that dis_ω is not necessarily symmetric [2]. Now, we present the theorem which is a generalization of Theorem 7.2.2.

Theorem 7.3. *Suppose ω is a positive function defined on $\mathbb{Z}^+$ and $\omega(i) = 0 \forall i > k$. If $\tau = \{\tau(1), \tau(2), \dots, \tau(k)\}$ is the set of k tuples with the largest $\Upsilon_\omega(t)$ values,*

[2] Rigorously, a distance function (or metric) should satisfy positive definiteness, symmetry and triangle inequality. Here we abuse this term a bit

then τ is the CON *answer under the weighted symmetric difference* dis_ω*, i.e, the answer minimizes* $\mathbb{E}[\mathsf{dis}_\omega(\tau, \tau_{pw})]$.

Proof: The proof mimics the one for Theorem . Suppose τ is fixed. We can write $\mathbb{E}[\mathsf{dis}_\omega(\tau, \tau_{pw})]$ as follows:

$$\begin{aligned}\mathbb{E}\left[\mathsf{dis}_\omega(\tau, \tau_{pw})\right] &= \mathbb{E}\left[\sum_{t\in T} \omega(\tau_{pw}(t))\delta(t \in \tau_{pw} \wedge t \notin \tau)\right] \\ &= \sum_{t\in T\setminus\tau} \mathbb{E}[\omega(\tau_{pw}(t))\delta(t \in \tau_{pw})] \\ &= \sum_{t\in T\setminus\tau} \sum_{i=1}^{k} \omega(i)\Pr(r(t) = i) = \sum_{t\in T\setminus\tau} \Upsilon_\omega(t)\end{aligned}$$

Therefore, it is clear that the set of k tuples with the largest $\Upsilon_\omega(t)$ values minimizes the above quantity. □

Although the weighted symmetric difference appears to be a very rich class of distance functions, its relation with other well studied distance functions, such at Spearman's rho and Kendall's tau, is still not well understood. We left it as an interesting open problem.

7.2.3 Intersection Metric

Note that the intersection metric dis_I is a linear combination of the normalized symmetric difference metric dis_Δ. Using a similar approach used in the proof of Theorem 7.2.2, we can show that:

$$\begin{aligned}\mathbb{E}[\mathsf{dis}_I(\tau, \tau_{pw})] &= \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[\mathsf{dis}_\Delta(\tau^i, \tau^i_{pw})] \\ &= \frac{1}{k}\sum_{i=1}^{k} \frac{1}{i}\left(k + \sum_{t\in T}\Pr(r(t) \le k) - 2\sum_{t\in\tau^i}\Pr(r(t) \le i)\right)\end{aligned}$$

Thus we need to find τ which maximizes the last term, $A(\tau) = \sum_{i=1}^{k}\left(\frac{1}{i}\sum_{t\in\tau^i}\Pr(r(t) \le i)\right)$.

We first rewrite the objective as follows, using the indicator (δ) function:

$$\begin{aligned} A(\tau) &= \sum_{i=1}^{k} \left(\frac{1}{i} \sum_{t \in T} \Pr(r(t) \leq i)) \delta(t \in \tau^i) \right) \\ &= \sum_{t \in T} \left(\sum_{i=1}^{k} \frac{1}{i} \Pr(r(t) \leq i) \sum_{j=1}^{i} \delta(t = \tau(j)) \right) \\ &= \sum_{t \in T} \sum_{j=1}^{k} \left(\delta(t = \tau(j)) \sum_{i=j}^{k} \frac{1}{i} \Pr(r(t) \leq i) \right) \end{aligned}$$

The last equality holds since $\sum_{i=1}^{k} \sum_{j=1}^{i} a_{ij} = \sum_{j=1}^{k} \sum_{i=j}^{k} a_{ij}$.

The optimization task can thus be written as an *assignment problem*, with each tuple t acting as an agent and each of the top-k positions j as a task. Assigning task j to agent t gains a profit of $\sum_{i=j}^{k} \frac{1}{i} \Pr(r(t) \leq i)$ and the goal is to find an assignment such that each task is assigned to at most one agent, and the profit is maximized. The best known algorithm for computing the optimal assignment runs in $O(nk\sqrt{n})$ time, via computing a maximum weight matching on the bipartite graph [134].

7.2.4 Approximating the Intersection Metric by PRF^ω

We define the following ranking function, where $H_k = \sum_{i=1}^{k} 1/i$ denotes the k^{th} Harmonic number:

$$\Upsilon_H(t) = \sum_{i=1}^{k} (H_k - H_{i-1}) \Pr(r(t) = i) = \sum_{i=1}^{k} \frac{\Pr(r(t) \leq i)}{i}.$$

This is a special case of PRF^ω with weight function $\omega(i) = H_k - H_{i-1}$. We claim that the top-k answer τ_H returned by Υ_H function, i.e., the k tuples with the highest Υ_H values, is a good approximation of the mean answer with respect to the intersection metric by arguing that $\tau_H = \{t_1, t_2, \ldots, t_k\}$ is actually an approximated maximizer of $A(\tau)$. Indeed, we prove the fact that $A(\tau_H) \geq \frac{1}{H_k} A(\tau^*)$ where τ^* is the optimal

mean top-k answer.

Let $B(\tau) = \sum_{t\in\tau} \Upsilon_H(t)$ for any top-k answer τ. It is easy to see $A(\tau^*) \leq B(\tau^*) \leq B(\tau_H)$ since τ_H maximizes the $B()$ function. Then, we can get:

$$\begin{aligned}
A(\tau_H) &= \sum_{j=1}^{k}\sum_{i=j}^{k} \frac{1}{i} \Pr(r(t_j) \leq i) \\
&\geq \sum_{j=1}^{k} (\frac{H_k - H_{j-1}}{H_k}) \sum_{i=1}^{k} \frac{1}{i} \Pr(r(t_j) \leq i) \\
&= \sum_{j=1}^{k} (\frac{H_k - H_{j-1}}{H_k}) \Upsilon_H(t_j) \geq \frac{1}{k} \sum_{i=1}^{k} (\frac{H_k - H_{i-1}}{H_k}) \sum_{i=1}^{k} \Upsilon_H(t_i) \\
&= \frac{1}{H_k} B(\tau_H) \geq \frac{1}{H_k} A(\tau^*).
\end{aligned}$$

The second inequality holds because for non-decreasing sequences $a_i (1 \leq i \leq n)$ and $c_i (1 \leq i \leq n)$, $\sum_{i=1}^{n} a_i c_i \geq \frac{1}{n} (\sum_{i=1}^{n} a_i)(\sum_{i=1}^{n} c_i)$.

7.2.5 Spearman's Footrule

For a top-k answer $\tau = \{\tau(1), \tau(2), \ldots, \tau(k)\}$, we define:

- $\Upsilon_1(t) = \sum_{i=1}^{k} \Pr(r(t) = i)$
- $\Upsilon_2(t) = \sum_{i=1}^{k} \Pr(r(t) = i) \cdot i$
- $\Upsilon_3(t, i) = \sum_{j=1}^{k} \Pr(r(t) = j))|i - j| + i \Pr(r(t) > k)$.

It is easy to see $\Upsilon_1(t), \Upsilon_2(t), \Upsilon_3(t)$ can be computed in polynomial time for a probabilistic and/xor tree using our generating functions method.

A careful rewriting of $\mathbb{E}[F^*(\tau, \tau_{pw})]$ shows that it also has the form:

$$\mathbb{E}[F^*(\tau, \tau_{pw})] = C + \sum_{t\in T} \sum_{i=1}^{k} \delta(t = \tau(i)) f(t, i)$$

where C is a constant independent of τ, and $f(t,i)$ is a function of t and i that is polynomially computable. More specifically, $f(t,i) = \Upsilon_3(t,i)+\Upsilon_2(t)-2(k+1)\Upsilon_1(t)$ The exact derivation is shown below:

$$
\begin{aligned}
\mathbb{E}[F^*(\tau,\tau_{pw})] &= \mathbb{E}\left[(k+1)|\tau\Delta\tau_{pw}| + \sum_{t\in\tau\cap\tau_{pw}} |\tau(t)-\tau_{pw}(t)| - \sum_{t\in\tau\setminus\tau_{pw}} \tau(t) - \sum_{t\in\tau_{pw}\setminus\tau} \tau_{pw}(t)\right] \\
&= (k+1)\mathbb{E}[|\tau\Delta\tau_{pw}|] + \sum_{t\in T} \mathbb{E}\left[\delta(t\in\tau\cap\tau_{pw})|\tau(t)-\tau_{pw}(t)|\right] \\
&\quad - \sum_{t\in T} \mathbb{E}\left[\delta(t\in\tau\setminus\tau_{pw})\tau(t)\right] - \mathbb{E}\left[\sum_{t\in\tau_{pw}\setminus\tau} \tau_{pw}(t)\right] \\
&= (k+1)\mathbb{E}[|\tau\Delta\tau_{pw}|] + \sum_{t\in T}\sum_{i=1}^{k}\sum_{j=1}^{k} \mathbb{E}\left[\delta(t\in\tau\cap\tau_{pw})\delta(t=\tau_{pw}(i))\delta(t=\tau(j))|i-j|\right] \\
&\quad - \sum_{t\in T}\sum_{i=1}^{k} \mathbb{E}\left[\delta(t\in\tau\setminus\tau_{pw})\delta(t=\tau(i))i\right] - \sum_{t\in T\setminus\tau} \Upsilon_2(t) \\
&= (k+1)\mathbb{E}[|\tau\Delta\tau_{pw}|] + \sum_{t\in T}\sum_{i=1}^{k}\left(\delta(t=\tau(i))\sum_{j=1}^{k}\Pr(r(t)=j)|i-j|\right) \\
&\quad - \sum_{t\in T}\sum_{i=1}^{k}(\delta(t=\tau(i))i\Pr(r(t)>k)) - \sum_{t\in T\setminus\tau}\Upsilon_2(t) \\
&= (k+1)(k+\sum_{t\in T}\Upsilon_1(t) - 2\sum_{t\in\tau}\Upsilon_1(t)) + \sum_{t\in T}\sum_{i=1}^{k}\delta(t=\tau(i))\Upsilon_3(t,i) - \sum_{t\in T\setminus\tau}\Upsilon_2(t) \\
&= (k+1)k + \sum_{t\in T}((k+1)\Upsilon_1(t)-\Upsilon_2(t)) \\
&\quad + \sum_{t\in T}\sum_{i=1}^{k}\delta(t=\tau(i))(\Upsilon_3(t,i)+\Upsilon_2(t)-2(k+1)\Upsilon_1(t))
\end{aligned}
$$

Thus, we only need to minimize the second term, which can be modeled as the assignment problem and can be solved in polynomial time.

7.2.6 Kendall's Tau Distance

Then *Kendall's tau* distance (also called Kemeny distance) dis_K between two top-k lists τ_1 and τ_2 is defined to be the number of unordered pairs (t_i, t_j) such that that the order of i and j disagree in any full rankings extended from τ_1 and τ_2, respectively. It is shown that dis_F and dis_K and a few other generalizations of

Spearman's footrule and Kendall's tau metrics form a big equivalence class, i.e., they are within a constant factor of each other [66]. Therefore, the optimal solution for dis_F implies constant approximations for all metrics in this class (the constant for dis_K is 2).

However, we can also easily obtain a 3/2-approximation for dis_K by extending the 3/2-approximation for partial rank aggregation problem due to Ailon [7]. The only information used in their algorithm is the proportion of lists where t_i is ranked higher than t_j for all i, j. In our case, this corresponds to $\Pr(r(t_i) < r(t_j))$. This can be easily computed in polynomial time using the generating functions method.

We also note that the problem of optimally computing the mean answer is NP-hard for probabilistic and/xor trees. This follows from the fact that probabilistic and/xor trees can simulate arbitrary possible worlds, and previous work has shown that aggregating even 4 rankings under this distance metric is NP-Hard [64].

7.3 Consensus Answers for Other Types of Queries

7.3.1 Set Distance Measures

We first consider the problem of finding the consensus world for a probabilistic relation under two set distance measures: symmetric difference, and Jaccard distance; the probabilistic relation may be an existing relation in the database, or the result of executing a conjunctive query over it.

7.3.1.1 Symmetric Difference

The symmetric difference distance between two sets S_1, S_2 is defined to be

$$\mathsf{dis}_\Delta(S_1, S_2) = |S_1 \Delta S_2| = |(S_1 \setminus S_2) \cup (S_2 \setminus S_1)|.$$

Note that two different alternatives of a tuple are treated as different tuples here.

Theorem 7.4. *The mean world under the symmetric difference distance is the set of all tuples with probability* > 0.5.

Proof: Suppose S is a fixed set of tuples and $\bar{S} = T - S$. Let $\delta(p) = \begin{cases} 1, & \text{if } p = true \\ 0, & \text{if } p = false \end{cases}$ be the indicator function. We write $E_{pw \in PW}[\mathsf{dis}_\Delta(S, pw)]$ as follows:

$$\begin{aligned} \mathbb{E}[\mathsf{dis}_\Delta(S, pw)] &= \mathbb{E}[\sum_{t \in S} \delta(t \notin pw) + \sum_{t \in \bar{S}} \delta(t \in pw)] \\ &= \sum_{t \in S} \mathbb{E}[\delta(t \notin pw)] + \sum_{t \in \bar{S}} \mathbb{E}[\delta(t \in pw)] \\ &= \sum_{t \in S} \Pr(\neg t) + \sum_{t \in \bar{S}} \Pr(t) \end{aligned}$$

Thus, each tuple t contributes $\Pr(\neg t)$ to the expected distance if $t \in S$ and $\Pr(t)$ otherwise, and hence the minimum is achieved by the set of tuples with probability > 0.5. □

Thus, finding the mean answer for a conjunctive query is easy if we can decide which result tuples have probability > 0.5.

Finding the consensus median world is somewhat trickier, with the main concern being that the world that contains all tuples with probability > 0.5 may not be a possible world.

Corollary 1. *If the correlations can be modeled using a probabilistic and/xor tree, the median world is the set containing all tuples with probability greater than* 0.5.

The proof is by induction on the height of the tree, and is omitted for space constraints. This however does not hold for arbitrary correlations. Next we show that finding a median answer for a conjunctive query is NP-Hard even if result tuple probability computation is easy (i.e., even if the query has a safe plan) because of the correlations between the result tuples.

Theorem 7.5. *For conjunctive queries over databases with arbitrary correlations, finding a median answer under the symmetric difference distance is NP-Hard.*

Proof: Consider the query:

$$Q(C) := \pi_C(R \bowtie S)$$

where $R = R(C, x, b)$ are $S = S(x, b)$ are two relations independent with each other. We show finding a median world for this query is NP-Hard by showing a reduction from the MAX-2-SAT problem. Recall that in a MAX-2-SAT instance, we are given a conjunctive normal form expression with 2 literals per clause and the task is to determine the maximum number of clauses that can be simultaneously satisfied by an assignment. Let the MAX-2-SAT instance consist of n variables, $x_1, \ldots, x_n$, and k clauses. Let $S(x, b) = \{(x_1, 0), (x_1, 1), (x_2, 0), (x_2, 1), \ldots\}$ contain two mutually exclusive tuples each for n variables; all tuples are equi-probable with probability 0.5. $R(C, x, b)$ is a deterministic table, and contains two tuples for each clause: Suppose x_j (or $\bar{x}_j$) is a literal in clause c_i, R contains tuple $(c_i, x_j, 1)$ (or $(c_i, x_j, 0)$). We can see that $R \bowtie S$ has the same set of tuples as R and each tuple has probability 0.5. Moreover, two tuples with the same C value are independent. Therefore, the result of $\pi_C(R \bowtie S)$ contains one tuple for each clause, associated with a probability of $1 - 0.5 \times 0.5 = 0.75$.

Now, consider the possible deterministic answer which is generated by a deterministic instance $\tilde{S}$ of S. It is easy to see the answer contain clause c_i if and only if c_i is satisfied by the assignment defined by $\tilde{S}$. According to the proof of Theorem 7.4, the median answer is the possible deterministic answer containing maximum number of tuples, which corresponds to finding the assignment that maximizes the number of satisfied clauses. □

7.3.1.2 Jaccard Distance

The Jaccard distance between two sets S_1, S_2 is defined to be

$$\mathsf{dis}_J(S_1, S_2) = \frac{|S_1 \Delta S_2|}{|S_1 \cup S_2|}.$$

Jaccard distance always lies in $[0, 1]$ and is a real metric, i.e, satisfies triangle inequality. Next we present polynomial time algorithms for finding the mean and median worlds for tuple independent databases, and median world for the BID model.

Lemma 4. *Given an and/xor tree, $\mathcal{T}$ and a possible world for it, W (corresponding to a set of leaves of $\mathcal{T}$), we can compute $\mathbb{E}[\mathsf{dis}(W, pw)]$ in polynomial time.*

Proof: A generating function $\mathcal{F}_{\mathcal{T}}$ is constructed with the variables associated with leaves as follows: for $t \in W$ ($t \notin W$), the associated variable is x (y). For example, in a tuple independent database, the generating function is:

$$\mathcal{F}(x, y) = \prod_{t \in W} (\Pr(\neg t) + \Pr(t)x) \prod_{t \notin W} (\Pr(\neg t) + \Pr(t)y)$$

From Theorem 5.1, the coefficient $c_{i,j}$ of term $x^i y^j$ in generating function $\mathcal{F}$ is equal to the total probability of the worlds such that the Jaccard distance between those worlds and W is exactly $\frac{|W|-i+j}{|W|+j}$. Thus, the distance is $\sum_{i,j} c_{i,j} \frac{|W|-i+j}{|W|+j}$. $\square$

Lemma 5. *For tuple independent databases, if the mean world contains tuple t_1 but not tuple t_2, then $\Pr(t_1) \geq \Pr(t_2)$.*

Proof: Say W_1 is the mean world and the lemma is not true, i.e, $\exists t_1 \in W_1, t_2 \notin W_1$ s.t. $\Pr(t_1) < \Pr(t_2)$. Let $W = W_1 - \{t_1\}$, $W_2 = W + \{t_2\}$ and $W' = T - W - \{t_1\} - \{t_2\}$. We will prove W_2 has a smaller expected Jaccard distance, thus rendering contradiction. Suppose $|W_1| = |W_2| = k$. We let matrix $\mathbf{M} = [m_{i,j}]_{i,j}$ where $m_{i,j} = \frac{k-i+j}{k+j}$. We construct generating functions as we did in Lemma 4. Suppose $\mathcal{F}_1$ and $\mathcal{F}_2$ are the generating functions for W_1 and W_2, respectively. We write $||\mathbf{A}|| = \sum_{i,j} a_{i,j}$ for any matrix $\mathbf{A}$ and let $\mathbf{A} \otimes \mathbf{B}$ the Hadamard product of $\mathbf{A}$ and $\mathbf{B}$ (take product entrywise). We denote:

$$\mathcal{F}'(x, y) = \prod_{t \in W} (\Pr(\neg t) + \Pr(t)x) \prod_{t \in W'} (\Pr(\neg t) + \Pr(t)y)$$

We can easily see that:

$$\mathcal{F}_1(x,y) = \mathcal{F}'(x,y)\left(\Pr(\neg t_1) + \Pr(t_1)x\right)\left(\Pr(\neg t_2) + \Pr(t_2)y\right)$$

$$\mathcal{F}_2(x,y) = \mathcal{F}'(x,y)\left(\Pr(\neg t_1) + \Pr(t_1)y\right)\left(\Pr(\neg t_2) + \Pr(t_2)x\right)$$

Then, taking the difference, we get $\bar{\mathcal{F}} = \mathcal{F}_1(x,y) - \mathcal{F}_2(x,y)$ is equal to:

$$\mathcal{F}'(x,y)\left(\Pr(\neg t_1)\Pr(t_2) - \Pr(t_1)\Pr(\neg t_2)\right)(y-x) \tag{7.1}$$

Let $\mathbf{C}_{\mathcal{F}} = [c_{i,j}]$ be the coefficient matrix of $\mathcal{F}$ where $c_{i,j}$ is the coefficient of term $x^i y^j$. Using the proof of Lemma 4:

$$\mathbb{E}[\mathsf{dis}(W_1, pw)] - \mathbb{E}[\mathsf{dis}(W_2, pw)] \;=\; ||\mathbf{C}_{\mathcal{F}_1} \otimes \mathbf{M}|| - ||\mathbf{C}_{\mathcal{F}_2} \otimes \mathbf{M}|| = ||\mathbf{C}_{\bar{\mathcal{F}}} \otimes \mathbf{M}||$$

Let $c'_{i,j}$ and $\bar{c}_{i,j}$ be the coefficient of $x^i y_j$ in $\mathcal{F}'$ and $\bar{\mathcal{F}}$, respectively. It is not hard to see $\bar{c}_{i,j} = (c'_{i,j-1} - c'_{i-1,j})p$ from (7.1) where $p = \left(\Pr(\neg t_1)\Pr(t_2) - \Pr(t_1)\Pr(\neg t_2)\right) > 0$.

Then we have:

$$\begin{aligned}
||\mathbf{C}_{\bar{\mathcal{F}}} \otimes \mathbf{M}|| &= p\sum_{i,j}\left((c'_{i,j-1} - c'_{i-1,j})m_{i,j}\right) \\
&= p\sum_{i,j} c'_{i,j}(m_{i,j+1} - m_{i+1,j}) \\
&= p\sum_{i,j} c'_{i,j}\left(\frac{k-i+j+1}{k+j+1} - \frac{k-i-1+j}{k+j}\right)
\end{aligned}$$

The proof follows because, for any $i, j \geq 0$, we have that $\frac{k-i+j+1}{k+j+1} - \frac{k-i-1+j}{k+j} > 0$. □

The above two lemmas can be used to efficiently find the mean world for tuple-independent databases, by sorting the tuples in the decreasing order by probabilities, and computing the expected distance for every prefix of the sorted order.

Theorem 7.6. *Under Jaccard distance metric ,there is polynomial time algorithms for computing mean and median world for tuple independent databases.*

A similar algorithm can be used to find the median world for the BID model (by only considering the highest probability alternative for each tuple). Finding mean worlds or median worlds under more general correlation models remains an open problem.

7.3.2 Aggregate Queries

Consider a query of the type:

SELECT groupname, count(*)
FROM R
GROUP BY groupname

We assume the dataset is represented by the BID model in which there are m potential groups (indexed by groupname) and n independent tuples with attribute uncertainty. The probabilistic database can be specified by the matrix $\mathbf{P} = [p_{i,j}]_{n \times m}$ where $p_{i,j}$ is the probability that tuple i takes groupname j and $\sum_{j=1}^{m} p_{i,j} = 1$ for any $1 \leq i \leq n$. A query result (on a deterministic relation) is a m-dimensional vector $\mathbf{r}$ where the i^{th} entry is the number of tuples having groupname i. The natural distance metric to use is the squared vector distance.

Computing the mean answer is easy in this case, because of linearity of expectation: we simply take the mean for each aggregate separately, i.e., $\bar{\mathbf{r}} = \mathbf{1P}$ where $\mathbf{1} = (1, 1, \ldots, 1)$. We note the mean answer minimizes the expected squared vector distance to any possible answer.

The median world requires that the returned answer be a possible answer. It is not clear how to solve this problem optimally in polynomial time. To enumerate all worlds is obviously not computationally feasible. Rounding entries of $\bar{\mathbf{r}}$ to the nearest integers may not result in a possible answer.

Next we present a polynomial time algorithm to find a closest possible answer to

the mean world $\overline{\mathbf{r}}$. This yields a 4-approximation for finding the median answer. We can model the problem as follows: Consider the bipartite graph $B(U, V, E)$ where each node in U is a tuple, each node in V is a groupname, and an edge $(u, v), u \in U, v \in V$ indicates that tuple u takes groupname v with non-zero probability. We call a subgraph G' such that $deg_{G'}(u) = 1$ for all $u \in U$ and $deg_{G'}(v) = \mathbf{r}[v]$, an $\mathbf{r}$-*matching* of B for some m-dimensional integral vector $\mathbf{r}$. Given this, our objective is to find an $\mathbf{r}$-matching of B such that $||\mathbf{r} - \overline{\mathbf{r}}||_2^2$ is minimized. Before presenting the main algorithm, we need the following lemma.

Lemma 6. *The possible world* $\mathbf{r}^*$ *that is closest to* $\overline{\mathbf{r}}$ *is of the following form:* $\mathbf{r}^*[i]$ *is either* $\lfloor \overline{\mathbf{r}}[i] \rfloor$ *or* $\lceil \overline{\mathbf{r}}[i] \rceil$ *for each* $1 \leq i \leq m$.

Proof: Let M^* be the corresponding $\mathbf{r}^*$-matching. Suppose the lemma is not true, and there exists i such that $|\mathbf{r}^*[i] - \overline{\mathbf{r}}[i]| > 1$. W.l.o.g, we assume $\mathbf{r}^*[i] > \overline{\mathbf{r}}[i]$. The other case can be proved the same way. Consider the connected component $K = \{U', V', E(U', V')\}$ containing i. We claim that there exists $j \in V'$ such that $\mathbf{r}^*[j] < \overline{\mathbf{r}}[j]$ and there is an *alternating path* P with respect to M^* connecting i and j [3]. Therefore, $M' = M^* \Delta P = (M^* \setminus P) \cup (P \setminus M^*)$ is also a valid matching. Suppose M' is a $\mathbf{r}'$-matching. But:

$$
\begin{aligned}
||\mathbf{r}' - \overline{\mathbf{r}}||_2^2 &= \sum_{v=1}^{m} (\mathbf{r}'[v] - \overline{\mathbf{r}}[v])^2 \\
&= \sum_{v=1}^{m} (\mathbf{r}^*[v] - \overline{\mathbf{r}}[v])^2 - (\mathbf{r}^*[i] - \overline{\mathbf{r}}[i])^2 - \\
&\quad (\mathbf{r}^*[j] - \overline{\mathbf{r}}[j])^2 + (\mathbf{r}'[i] - \overline{\mathbf{r}}[i])^2 + (\mathbf{r}'[j] - \overline{\mathbf{r}}[j])^2 \\
&= ||\mathbf{r}^* - \overline{\mathbf{r}}||_2^2 - (\mathbf{r}^*[i] - \overline{\mathbf{r}}[i])^2 - (\mathbf{r}^*[j] - \overline{\mathbf{r}}[j])^2 \\
&\quad + (\mathbf{r}^*[i] - 1 - \overline{\mathbf{r}}[i])^2 + (\mathbf{r}^*[j] + 1 - \overline{\mathbf{r}}[j])^2 \\
&= ||\mathbf{r}^* - \overline{\mathbf{r}}||_2^2 + 2 - 2\mathbf{r}^*[i] + 2\overline{\mathbf{r}}[i] + 2\mathbf{r}^*[j] - 2\overline{\mathbf{r}}[j] \\
&< ||\mathbf{r}^* - \overline{\mathbf{r}}||_2^2.
\end{aligned}
$$

[3] An alternating path is a path with alternating unmatched and matched edges [47].

This contradicts the assumption $\mathbf{r}^*$ is the vector closest to $\bar{\mathbf{r}}$.

Now, we prove the claim. We grow a *alternating path tree* (w.r.t. M^*) rooted at i in a Bread-First-Search (BFS) manner [4]. Let $Odd \subseteq V$ be the set of nodes at odd depth (the root is at depth 1) and $Even \subseteq U$ the set of nodes at even depth. For any subset S of vertices, let $N_B(S)$ denote the set of neighbors of S in graph B. It is easy to see $N_B(Even) = Odd$, $Even \subseteq N_B(Odd)$ and $\sum_{v \in Odd} \mathbf{r}^*[v] = |Even|$. Suppose $\mathbf{r}^*[v] \geq \bar{\mathbf{r}}[v]$ for all v and $\mathbf{r}^*[i] > \bar{\mathbf{r}}[i]$. However, the contradiction follows since:

$$\begin{aligned} |Even| &= \sum_{v \in Odd} \mathbf{r}^*[v] > \sum_{v \in Odd} \bar{\mathbf{r}}[v] = \sum_{v \in Odd} \sum_{u \in N_B(Odd)} \mathbf{P}[u, v] \\ &= \sum_{v \in Odd} \sum_{u \in Even} \mathbf{P}[u, v] = |Even|. \end{aligned}$$

Therefore, there must be a vertex j such that $\mathbf{r}^*[j] < \bar{\mathbf{r}}[j]$ in the alternating path tree. □

With Lemma 6 at hand, we can construct the following min-cost network flow instance to compute the vector $\mathbf{r}^*$ closest to $\bar{\mathbf{r}}$. Add to B a source s and a sink t. Add edges (s, u) with capacity upper bound 1 for all $u \in U$. For each $v \in V$ and $\bar{\mathbf{r}}[v]$ is not integer, add two edges $e_1(v, t)$ and $e_2(v, t)$. $e_1(v, t)$ has both lower and upper bound of capacity $\lfloor \bar{\mathbf{r}}[v] \rfloor$ and $e_2(v, t)$ has capacity upper bound 1 and cost $(\lceil \bar{\mathbf{r}}[v] \rceil - \bar{\mathbf{r}}[v])^2 - (\lfloor \bar{\mathbf{r}}[v] \rfloor - \bar{\mathbf{r}}[v])^2$. If $\bar{\mathbf{r}}[v]$ is a integer, we only add $e_1(v, t)$. We find a min-cost integral flow of value n on this network. For any v such that $e_2(v, t)$ is saturated, we set $\mathbf{r}^*[v]$ to be $\lceil \bar{\mathbf{r}} \rceil$ and $\lfloor \bar{\mathbf{r}} \rfloor$ otherwise. Such a flow with minimum cost suggests the optimality of the vector $\mathbf{r}^*$ due to Lemma 6.

Theorem 7.7. *There is a polynomial time algorithm for finding the vector* $\mathbf{r}^*$ *to* $\bar{\mathbf{r}}$ *such that* $\mathbf{r}^*$ *corresponds to some possible answer with non-zero probability.*

Finally, we can prove that:

[4] An alternating path tree is a tree in which each path from the root to another node is an alternating path with its first edge being a matched edge [47].

Corollary 2. *There is a polynomial time deterministic 4-approximation for finding the median aggregate answer.*

Proof: Suppose $\mathbf{r}^*$ is the possible answer closest to the mean answer $\bar{\mathbf{r}}$ and $\mathbf{r}^m$ is the optimal median answer. Let $\mathbf{r}$ be the vector corresponding to the random answer. Then:

$$\begin{aligned}\mathbb{E}[\mathsf{dis}(\mathbf{r}^*, \mathbf{r})] &\leq \mathbb{E}[2(\mathsf{dis}(\mathbf{r}^*, \bar{\mathbf{r}}) + \mathsf{dis}(\bar{\mathbf{r}}, \mathbf{r}))] = 2\left(\mathsf{dis}(\mathbf{r}^*, \bar{\mathbf{r}}) + \mathbb{E}[\mathsf{dis}(\bar{\mathbf{r}}, \mathbf{r})]\right) \\ &\leq 4\mathbb{E}[\mathsf{dis}(\bar{\mathbf{r}}, \mathbf{r})] \leq 4\mathbb{E}[\mathsf{dis}(\mathbf{r}^m, \mathbf{r})].\end{aligned}$$

The proof is complete. □

7.3.3 Clustering

The Consensus-Clustering problem is defined as follows: given k clusterings $\mathcal{C}_1, \ldots, \mathcal{C}_k$ of V, find a clustering $\mathcal{C}$ that minimizes $\sum_{i=1}^{k} \mathsf{dis}(\mathcal{C}, \mathcal{C}_i)$. In the setting of probabilistic databases, the given clusterings are the clusterings in the possible worlds, weighted by the existence probability. The main problem with extending the notion of consensus answers to clustering is that the input clusterings are not well-defined (unlike ranking where the score function defines the ranking in any world). We consider a somewhat simplified version of the problem, where we assume that two tuples t_i and t_j are clustered together in a possible world, if and only if they take the same value for the value attribute A (which is uncertain). Thus, a possible world pw uniquely determines a clustering $\mathcal{C}_{pw}$. We define the distance between two clustering $\mathcal{C}_1$ and $\mathcal{C}_2$ to be the number of unordered pairs of tuples that are clustered together in $\mathcal{C}_1$, but separated in the other (the Consensus-Clustering metric). To deal with nonexistent keys in a possible world, we artifically create a cluster containing all of those.

Our task is to find a mean clustering $\mathcal{C}$ such that $\mathbb{E}[\mathsf{dis}(\mathcal{C}, \mathcal{C}_{pw})]$. Approximation with factor of 4/3 is known for Consensus-Clustering [8], and can

be adapted to our problem in a straightforward manner. In fact, that approximation algorithm simply needs w_{t_i,t_j} for all t_i, t_j, where w_{t_i,t_j} is the fraction of input clusters that cluster t_i and t_j together, and can be computed as: $w_{t_i,t_j} = \sum_{a \in A} \Pr(i.A = a \wedge j.A = a)$.

To compute these quantities given an and/xor tree, we associate a variable x with all leaves with value (i, a) and (j, a), and constant 1 with the other leaves. From Theorem 5.1, $\Pr(i.A = a \wedge j.A = a)$ is simply the coefficient of x^2 in the corresponding generating function.

Chapter 8

Maximizing Expected Utility for Stochastic Combinatorial Optimization Problems

8.1 Introduction

In this chapter, we study a broad class of combinatorial optimization problems in presence of uncertainty. The deterministic version of the problem has the following form: we are given a ground set of elements $U = \{e_i\}_{i=1...n}$; each element e is associated with a weight w_e; each feasible solution is a subset of the elements satisfying some property. Let $\mathcal{F}$ denote the set of feasible solutions. The objective for the deterministic problem is to find a feasible solution S with the minimum total weight $w(S) = \sum_{e \in S} w_e$. We can see that many combinatorial problems such as shortest path, minimum spanning tree, and minimum weight matching belong to this class. In the stochastic version of the problem, the weight w_e of each element e is a nonnegative random variable. We assume all w_es are independent of each other. We use $p_e(.)$ to denote the probability density function for w_e (or probability mass function in discrete case). As we have argued in Section 1.2, we use the expected utility as the decision criterion. Hence, we are also given a utility function $\mu : \mathbb{R}^+ \to \mathbb{R}^+$ which maps a weight value to a utility value. By the expected utility maximization principle, our goal here is to find a feasible solution $S \in \mathcal{F}$ that maximizes the expected utility, i.e., $\mathbb{E}[\mu(w(S))]$. We call this problem the *expected utility maximization* (EUM) problem.

Let us use the following toy example to illustrate the rationale behind EUM. There is a graph with two nodes s and t and two parallel links e_1 and e_2. Edge e_1 has a fixed length 1 while the length of e_2 is 0.9 with probability 0.9 and 1.9

with probability 0.1 (the expected value is also 1). We want to choose one edge to connect s and t. It is not hard to imagine that a risk-averse user would choose e_1 since e_2 may turn out to be a much larger value with a nontrivial probability. We can capture such behavior using the utility function (8.1) (defined in Section 8.1.1). Similarly, we can capture the risk-prone behavior by using, for example, the utility function $\mu(x) = \frac{1}{x+1}$. It is easy to see that e_1 maximizes the expected utility in the former case, and e_2 in the latter.

8.1.1 Our Contributions

We discuss in detail our result for EUM. We assume μ is part of the specification of the problem but not part of the input. Moreover, we assume $\mu(x)$ is upper bounded by a constant and $\lim_{x\to\infty} \mu(x) = 0$. The later captures the fact that if the weight of a solution is too large, it becomes almost useless for us. W.l.o.g. we can also assume $0 \leq \mu(x) \leq 1$ for $x \geq 0$, by scaling. We say a function $\widetilde{\mu}(x)$ is an *ϵ-approximation* of $\mu(x)$ if $|\widetilde{\mu}(x) - \mu(x)| \leq \epsilon \forall x \geq 0$. For ease of exposition, we let $\widetilde{\mu}(x)$ be a complex function. Recall that a polynomial time approximation scheme (PTAS) is an algorithm which takes an instance of a minimization problem and a parameter ϵ and produces a solution whose cost is within a factor $1 + \epsilon$ of the optimum, and the running time, for any fixed ϵ, is polynomial in the size of the input. We use A to denote the deterministic combinatorial optimization problem under consideration. The *exact version* of a problem A asks the question whether there is a feasible solution of A with weight exactly equal to a given number K. We say an algorithm runs in *pseudopolynomial time* for the exact version of A if the running time is polynomial in n and K. Our first main theorem is the following.

Theorem 8.1. *Assume that there is a pseudopolynomial algorithm for the exact version of A. Further assume that given any $\epsilon > 0$, we can find an ϵ-approximation of the utility function μ as $\widetilde{\mu}(x) = \sum_{k=1}^{L} c_k \phi_k^x$, where L is a constant and $|\phi_k| \leq 1 \forall k$; ϕ_k may be complex numbers. Then, there is an algorithm that runs in time*

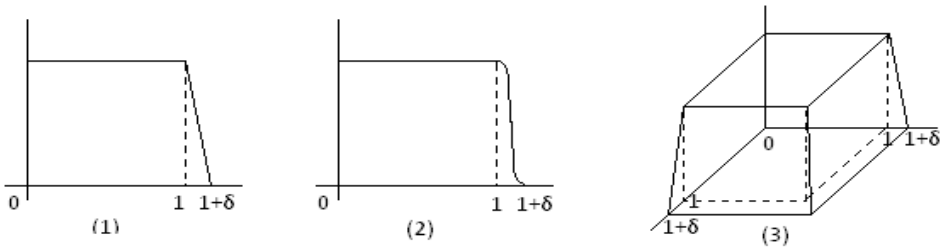


Figure 8.1: (1) The utility function $\widetilde{\chi}(x)$, a continuous variant of the threshold function $\chi(x)$; (2) A smoother variant of $\chi(x)$; (3) The utility function $\widetilde{\chi}_2(x)$, a continuous variant of the 2-d threshold function $\chi_2(x)$.

$(n/\epsilon)^{O(L)}$ *that approximates* $\mathsf{EUM}(A)$ *with an* additive *error* $O(\epsilon)$*. If the optimal expected utility is* $\Theta(1)$*, we obtain a PTAS.*

For many combinatorial problems, a pseudopolynomial algorithm for the exact version is known. Examples include shortest path, spanning tree, matching and knapsack. Hence, the only task left is to find a short exponential sum that ϵ-approximates μ. For this purpose, we adopt the Fourier series technique. However, the technique cannot be used directly since it works only for periodic functions with bounded periodicities. In order to get a good approximation for $x \in [0, \infty)$, we leverage the fact that $\lim_{x\to\infty} \mu(x) = 0$ and develop a general framework that uses the Fourier series decomposition as a subroutine. Generally speaking, such an approximation is only possible if the function is "well behaved", i.e., it satisfies some continuity or smoothness conditions. In particular, we prove Theorem 8.2. We say that the utility function μ satisfies the α-Hölder condition if $|\mu(x)-\mu(y)| \leq C\,|x-y|^{\alpha}$, for some constant C and some constant α.

Theorem 8.2. *If* μ *satisfies the* α*-Hölder condition for some constant* $\alpha > 1/2$*, then, for any* $\epsilon > 0$*, we can obtain an exponential sum with* $O(poly(\frac{1}{\epsilon}))$ *terms which is an* ϵ*-approximation of* μ *for* $x \geq 0$*.*

Consider the utility function

$$\widetilde{\chi}(x) = \begin{cases} 1 & x \in [0,1] \\ -\frac{x}{\delta} + \frac{1}{\delta} + 1 & x \in [1, 1+\delta] \\ 0 & x > 1+\delta \end{cases} \tag{8.1}$$

where $\delta > 0$ is a small constant (See Figure 8.1(1)). We can verify that $\widetilde{\chi}$ satisfies 1-Hölder condition with $C = \frac{1}{\delta}$. Therefore, Theorem 8.2 is applicable. This example is interesting since it can be viewed as a continuous variant of the threshold function

$$\chi(x) = \begin{cases} 1 & x \in [0,1] \\ 0 & x > 1 \end{cases}, \tag{8.2}$$

for which maximizing the expected utility is equivalent to maximizing $\Pr(w(S) \leq 1)$. This special case has been considered several times in literature for various problems including stochastic shortest path [141], stochastic spanning tree [102,77], stochastic knapsack [79] and some other stochastic problems [6, 139].

It is interesting to compare our result with the result for the stochastic shortest path problem considered by Nikolova et al. [141,139]. In [141], they show that there is an exact $O(n^{\log n})$ time algorithm for maximizing the probability that the length of the path is at most 1, i.e., $\Pr(w(S) \leq 1)$, assuming all edges are normally distributed and there is a path with its mean at most 1. Later, Nikolova [139] extends the result to an FPTAS for any problem under the same assumptions, if the deterministic version of the problem has a polynomial time exact algorithm. We can see that under such assumptions, the optimal probability is at least 1/2.[1] Therefore, provided the same assumption and further assuming that $\Pr(w_e < 0)$ is miniscule,[2] our algorithm is a PTAS for the continuous variant of the problem.

[1]The sum of multiple Gaussians is also a Gaussian. Hence, if we assume the mean of the length of a path (which is a Gaussian) is at most 1, the probability that the length of the path is at most 1 is at least 1/2.

[2]Our technique can only handle distributions with positive supports. Thus, we have to assume

Indeed, we can translate this result to a bi-criterion approximation result of the following form: for any fixed $\delta, \epsilon > 0$, we can find in polynomial time a solution S such that

$$\Pr(w(S) \leq 1 + \delta) \geq (1 - \epsilon) \Pr(w(S^*) \leq 1).$$

where S^* is the optimal solution (Corollary 4). We note that such a bi-criterion approximation was only known for exponentially distributed edges before [141].

Let us consider another application of our results to the stochastic knapsack problems defined in [79]. Given a set U of independent random variables $\{x_1, \ldots, x_n\}$, with associated profits $\{v_1, \ldots, v_n\}$ and an overflow probability γ, we are asked to pick a subset S of U such that

$$\Pr(\sum_{i \in S} x_i \geq 1) \leq \gamma$$

and the total profit $\sum_{i \in S} v_i$ is maximized. Goel and Indyk [79] showed that, for any $\epsilon > 0$, there is a polynomial time algorithm that can find a solution S with the profit as least the optimum and $\Pr(\sum_{i \in S} x_i \geq 1 + \epsilon) \leq \gamma(1 + \epsilon)$ for exponentially distributed variables. They also gave a quasi-polynomial time approximation scheme for Bernoulli distributed random variables. Quite recently, in parallel with our work, Bhalgat et al. [29] obtained the same result for arbitrary distributions under the assumption that $\gamma = \Theta(1)$. Their technique is based on discretizing the distributions and is quite involved. Our result, applied to stochastic knapsack, matches that of Bhalgat et al. We remark that our algorithm is much simpler and has a much better running time (Theorem 8.5). Despite a little loss in the approximation guarantees in some cases, our technique can be applied to almost all positive probability distributions, and a much richer class of utility functions.

Equally importantly, we can extend our basic approximation scheme to handle

that the probability that a negative value appears is miniscule and can be safely ignored.

generalizations such as multiple utility functions and multidimensional weights. Interesting applications of these extensions include generalizations of stochastic knapsack, such as *stochastic multiple knapsack* (Theorem 8.8) and *stochastic multidimensional knapsack (stochastic packing)* (Theorem 8.9).

8.2 Algorithm

We first note that EUM is #P-hard in general since the problem of computing the overflow probability of a set of items with Bernoulli distributions, a very special case of our problem, is #P-hard [119].

Our approach is very simple. We first observe that the problem is easy if the utility function is an exponential function. We approximate the utility function $\mu(x)$ by a short exponential sum, i.e., $\sum_{i=1}^{L} c_i \phi_i^x$ with L being a constant (c_i and ϕ_i may be complex numbers). Hence, $\mathbb{E}[\mu(w(S))]$ can be approximated by $\sum_{i=1}^{L} c_i \mathbb{E}[\phi_i^{w(S)}]$. Then, we consider the following multi-criterion version of the problem with L objectives $\{\mathbb{E}[\phi_i^{w(S)}]\}_{i=1,\ldots,L}$: given L complex numbers $v_1, \ldots, v_L$, we want to find a solution S such that $\mathbb{E}[\phi_i^{w(S)}] \approx v_i$ for $i = 1, \ldots, L$. We achieve this by utilizing the pseudopolynomial time algorithm for the exact version of the problem. We argue that we only need to consider a polynomial number of $v_1, \ldots, v_L$ combinations (which we call *configurations*) to find out the approximate optimum. In Section 8.2.1, we show how to solve the multi-criterion problem provided that a short exponential sum approximation of μ is given. In particular, we prove Theorem 8.1. Then, we show how to approximate μ by a short exponential sum by proving Theorem 8.2 in Section 8.2.2 and Section 8.2.3.

Let us first consider the exponential utility function $\mu(x) = \alpha^x$ for any $\alpha \in \mathbb{C}$. Fix an arbitrary solution S and $\alpha > 0$. Due to the independence of the elements,

we can see that

$$\mathbb{E}[\alpha^{w(S)}] = \mathbb{E}[\alpha^{\sum_{e\in S} w_e}] = \mathbb{E}[\prod_{e\in S} \alpha^{w_e}] = \prod_{e\in S} \mathbb{E}[\alpha^{w_e}]$$

Taking log on both sides, we get $\log \mathbb{E}[\alpha^{w(S)}] = \sum_{e\in S} \log \mathbb{E}[\alpha^{w_e}]$. If α is a positive real number and $\mathbb{E}[\alpha^{w_e}] \leq 1$ (or equivalently, $-\log \mathbb{E}[\alpha^{w_e}] \geq 0$), this reduces to the deterministic optimization problem.

We still need to show how to compute $\mathbb{E}[\alpha^{w_e}]$. If w_e is a discrete random variable with a polynomial size support, we can easily compute $\mathbb{E}[\alpha^{w_e}]$ in polynomial time. If w_e has an infinite discrete or continuous support, we cannot compute $\mathbb{E}[\alpha^{w_e}]$ directly and may need to approximate it. We briefly discuss this issue and its implications on our results in Section 8.2.4.

8.2.1 Proof of Theorem 8.1

Now, we prove Theorem 8.1. We start with some notations. We use $|c|$ and $\arg(c)$ to denote the absolute value and the argument of the complex number c, respectively. In other words, $c = |c|(\cos(\arg(c)) + i\sin(\arg(c))) = |c|e^{i\arg(c)}$. We always require $\arg(c) \in [0, 2\pi)$ for any $c \in \mathbb{C}$. Recall that we say the exponential sum $\sum_{i=1}^{L} c_i\phi_i^x$ is an ϵ-*approximation* for $\mu(x)$ if the following holds:

$$|\mu(x) - \sum_{i=1}^{L} c_i\phi_i^x| \leq \epsilon \quad \forall x \geq 0$$

We first show that if the utility function can be decomposed exactly into a short exponential sum, we can approximate the optimal expected utility well.

Theorem 8.3. *Assume $\widetilde{\mu}(x) = \sum_{k=1}^{L} c_k\phi_k^x$ is the utility function where $|\phi_k| \leq 1$ for $1 \leq k \leq L$. We also assume that there is a pseudopolynomial algorithm for the exact version of A. Then, for any $\epsilon > 0$, there is an algorithm that runs in time*

$(n/\epsilon)^{O(L)}$ and finds a solution S such that

$$|\mathbb{E}[\widetilde{\mu}(w(S))] - \mathbb{E}[\widetilde{\mu}(w(\widetilde{S}))]| < \epsilon$$

where $\widetilde{S} = \arg\max_{S'} |\mathbb{E}[\widetilde{\mu}(w(S'))|$.

We use the scaling and rounding technique that has been used often in multi-criterion optimization problems (e.g., [156, 145]). Since our objective function is not additive and not monotone, the general results for multi-criterion optimization [145, 135, 156, 3] do not directly apply here. We briefly sketch our algorithm. Let $\gamma = \delta = \frac{\epsilon}{Ln}$. For each $e \in U$, we associate it with a $2L$ dimensional integer vector

$$\langle a_1(e), b_1(e), \ldots, a_L(e), b_L(e)\rangle \text{ where } a_i(e) = \lfloor \frac{-\ln |\mathbb{E}[\phi_i^{w_e}]|}{\gamma} \rfloor \text{ and } b_i(e) = \lfloor \frac{\arg(\mathbb{E}[\phi_i^{w_e}])}{\delta} \rfloor.$$

$a_i(e)$ and $b_i(e)$ are the scaled and rounded versions of $-\ln |\mathbb{E}[\phi_i^{w_e}]|$ and $\arg(\mathbb{E}[\phi_i^{w_e}])$, respectively. Since $|\phi_i| \leq 1$, we can see that $a_i(e) \geq 0$ for any $e \in U$. We maintain $(JK)^L$ configurations where $J = \lceil \frac{-\ln(\epsilon/L)}{\gamma} \rceil$ and $K = \lceil \frac{2\pi n}{\delta} \rceil$. The number of configurations is $(n/\epsilon)^{O(L)}$. Each configuration $\sigma(\mathbf{a})$ is indexed by a $2L$-dimensional vector $\mathbf{a} = \langle \alpha_1, \beta_1, \ldots, \alpha_L, \beta_L \rangle$ where $1 \leq \alpha_i \leq J$ and $1 \leq \beta_i \leq K$ for $i = 1, \ldots, L$. In other words, the configurations are $\sigma(\langle 1, 1, \ldots, 1, 1\rangle), \ldots, \sigma(\langle J, K, \ldots, J, K\rangle))$. For vector $\mathbf{a} = \langle \alpha_1, \beta_1, \ldots, \alpha_L, \beta_L \rangle$, configuration $\sigma(\mathbf{a}) = 1$ if and only if there is a feasible solution $S \in \mathcal{F}$ such that for all $j = 1, \ldots, L$, $\beta_j = \sum_{e \in S} b_j(e)$, and $\alpha_j = \min(J, \sum_{e \in S} a_j(e))$. Otherwise, $\sigma(\mathbf{a}) = 0$. Lemma 7 tells us the expected utility for the rounded instance is close to the true value of the expected utility. Lemma 8 shows we can compute those configurations in polynomial time.

Lemma 7. *For vector $\mathbf{a} = \langle \alpha_1, \beta_1, \ldots, \alpha_L, \beta_L \rangle$, $\sigma_v(\mathbf{a}) = 1$ if and only if there is a solution S such that*

$$|\mathbb{E}[\widetilde{\mu}(w(S))] - \sum_{k=1}^{L} c_k e^{-\alpha_k \gamma + i\beta_k \delta}| \leq O(\epsilon).$$

Proof: We first notice that

$$\mathbb{E}[\widetilde{\mu}(w(S))] = \mathbb{E}[\sum_{k=1}^{L} c_k \phi_k^{w(S)}] = \sum_{k=1}^{L} c_k \mathbb{E}[\phi_k^{w(S)}].$$

Therefore, it suffices to show that for all $k = 1, \ldots, L$,

$$|\mathbb{E}[\phi_k^{w(S)}] - e^{-\alpha_k \gamma + i\beta_k \delta}| \leq O(\frac{\epsilon}{L}).$$

First, we can see that

$$\arg(\mathbb{E}[\phi_k^{w(S)}]) - \beta_k \delta = \sum_{e \in S} (\arg(\mathbb{E}[\phi_k^{w_e}]) - b_k(e)\delta) \leq \sum_{e \in S} \delta \leq n\delta = \frac{\epsilon}{L}.$$

If $\sum_{e \in S} a_k(e) > J$, we know that

$$-\ln(|\mathbb{E}[\phi_k^{w(S)}]|) = \sum_{e \in S} (-\ln(|\mathbb{E}[\phi^{w_e}|)) > J\gamma.$$

In this case, we have $\alpha_k = J$. Thus, we have

$$\left| |\mathbb{E}[\phi_k^{w(S)}]| - |e^{-\alpha_k \gamma}| \right| < e^{-J\gamma} = e^{\gamma \lceil \frac{\ln(\epsilon/L)}{\gamma} \rceil} < \frac{\epsilon}{L}.$$

If $\sum_{e \in S} a_k(e) \leq J$, we can see that

$$-\ln(|\mathbb{E}[\phi^{w(S)})|) - \alpha_k \gamma = \sum_{e \in S} (-\ln(|\mathbb{E}[\phi^{w_e}|) - \alpha_k(e)\gamma) \leq \sum_{e \in S} \gamma \leq n\gamma \leq \frac{\epsilon}{L}.$$

Since the derivative of e^x is less than 1 for $x < 0$, we can get

$$\left| |\mathbb{E}[\phi_k^{w(S)}]| - |e^{-\alpha_k \gamma}| \right| \leq |e^{-\alpha_k \gamma - \frac{\epsilon}{L}} - e^{-\alpha_k \gamma}| \leq \frac{\epsilon}{L}.$$

For any two complex numbers a, b with $|a| \leq 1$ and $|b| \leq 1$, if $\left| |a| - |b| \right| < h$ and $|\arg(a) - \arg(b)| < h$, we can easily show that $|a - b| < O(h)$. The proof is

complete. □

Lemma 8. *Suppose there is a pseudopolynomial time algorithm for the exact version of A, which runs in time polynomial in n and t (t is the maximum integer in the instance of A). Then, we can compute the values for these configurations in time* $(\frac{n}{\epsilon})^{O(L)}$.

Proof: For each element e, we associate a new vector $\bar{\mathbf{a}}_e = \langle \bar{a}_1, \bar{b}_1, \ldots, \bar{a}_L, \bar{b}_L \rangle$. If $a_i(e) > J$, we let $\bar{a}_i(e) = n(J+1)$ and $\bar{a}_i(e) = a_i(e)$ otherwise. Let $\bar{b}_i(e) = b_i(e)$ for all e and i. For each node v and each vector $\mathbf{a} = \langle \alpha_1, \beta_1, \ldots, \alpha_L, \beta_L \rangle$ such that $0 \leq \alpha_i \leq n^2(J+1) \forall i$ and $0 \leq \beta_i \leq K \forall i$, we want to compute the value $\bar{\sigma}_v(\mathbf{a})$ which is defined as follows: $\bar{\sigma}_v(\mathbf{a}) = 1$ if and only if there is a feasible solution $S \in \mathcal{F}$ such that for all $j = 1, \ldots, L$, $\beta_j = \sum_{e \in S} \bar{b}_j(e)$, and $\alpha_j = \sum_{e \in S} \bar{a}_j(e)$ (or more compactly, $\mathbf{a} = \sum_{e \in S} \bar{\mathbf{a}}_e$) ; $\bar{\sigma}_v(\mathbf{a}) = 0$ otherwise.

We can encode each vector as a nonnegative integer upper bounded by $(n^2JK)^L = (\frac{n}{\epsilon})^{O(L)}$. Then, determining the value of a configuration is equivalent to determining whether there is a feasible solution S such that the total weight of S is exactly a given value. Suppose the pseudopolynomial time algorithm for the exact version of A runs in time $P_A(n,t)$ for some polynomial P_A. Therefore, the value of each such $\bar{\sigma}_v(\mathbf{a})$ can be also computed in time $P_A(n, (\frac{n}{\epsilon})^{O(L)}) = (\frac{n}{\epsilon})^{O(L)}$. Since J and K are bounded by $(\frac{n}{\epsilon})^{O(1)}$, the number of configuration is $(\frac{n}{\epsilon})^{O(L)}$. The value of $\sigma(\langle \alpha_1, \beta_1, \ldots, \alpha_L, \beta_L \rangle)$ can be easily answered from the values of $\bar{\sigma}$s as follows :

1. If $\alpha_i < J \ \forall i$, $\sigma_v(\mathbf{a}) = \bar{\sigma}_v(\mathbf{a})$;
2. Denote $\mathbf{a}' = \langle \alpha'_1, \beta'_1, \ldots, \alpha'_L, \beta'_L \rangle$ and $S = \{i \mid \alpha_i = J\}$. $\sigma_v(\mathbf{a}) = \max_{\mathbf{a}'}(\bar{\sigma}_v(\mathbf{a}') \mid \beta'_i = \beta_i \forall i, \alpha'_i \geq J \forall i \in S, \alpha'_i = \alpha_i \forall i \notin S)$.

The total running time is $(\frac{n}{\epsilon})^{O(L)} \times (\frac{n}{\epsilon})^{O(L)} = (\frac{n}{\epsilon})^{O(L)}$. □

Now, we can easily prove Theorem 8.3.

Proof of Theorem 8.3: We first use the algorithm in Lemma 8 to compute the values for all configurations. Then, we find the configuration $\sigma(\langle \alpha_1, \beta_1, \ldots, \alpha_L, \beta_L \rangle)$ that has value 1 and that maximizes the quantity $|\sum_{k=1}^{L} c_k e^{-\alpha_k \gamma + i\beta_k \delta}|$. The feasible solution S corresponding to this configuration is our final solution. It is easy to see that the theorem follows from Lemma 7. □

Theorem 8.1 can be readily obtained from Theorem 8.3 and the fact $\widetilde{\mu}$ is an ϵ-approximation of μ.

Proof of Theorem 8.1: Suppose S is our solution and S^* is the optimal solution for utility function μ. From Theorem 8.3, we know that $|\mathbb{E}[\widetilde{\mu}(w(S))] \geq \mathbb{E}[\widetilde{\mu}(w(S^*))]| - \epsilon$. Since $\widetilde{\mu}$ is an ϵ-approximation of μ, we can see that

$$\left|\mathbb{E}[\mu(w(S))] - \mathbb{E}[\widetilde{\mu}(w(S))]\right| = \left|\int (\mu(x) - \widetilde{\mu}(x)) p_S(x) \mathrm{d}x\right| \leq \left|\int \epsilon p_S(x) \mathrm{d}x\right| \leq \epsilon$$

for any solution S, where p_S is the probability density function of S. Therefore, we have

$$|\mathbb{E}[\mu(w(S))]| \geq |\mathbb{E}[\widetilde{\mu}(w(S))]| - \epsilon \geq |\mathbb{E}[\widetilde{\mu}(w(S^*))]| - 2\epsilon \geq |\mathbb{E}[\mu(w(S^*))]| - 3\epsilon$$

The proof is complete. □

8.2.2 Approximating the Utility Function

In this subsection, we discuss the issue of approximating μ. In particular, we develop a generic algorithm that takes as a subroutine an algorithm AP for approximating functions in a bounded interval domain, and approximates $\mu(x)$ in the infinite domain $[0, +\infty)$. In the next subsection, we use the Fourier series expansion as the choice of AP and show that important classes of utility functions can be approximated well.

There are many works on approximating functions using short exponential sums, e.g., the Fourier decomposition approach [169], Prony's method [144], and

many others [24,25]. However, their approximations are done over a finite interval domain, say $[-\pi, \pi]$ or over a finite number of discrete points. No error bound can be guaranteed outside the domain. Our algorithm is a generic procedure that turns an algorithm that can approximate functions over $[-\pi, \pi]$ into one that can approximate our utility function μ over $[0, +\infty)$, by utilizing the fact that $\lim_{x\to\infty} \mu(x) = 0$.

Since $\lim_{x\to\infty} \mu(x) = 0$, for any ϵ, there exist a point T_ϵ such that $\mu(x) \leq \epsilon \ \forall x > T_\epsilon$. Since we assume the utility function μ is specified as a part of the problem but not a part of the input instance, T_ϵ is a constant for any constant ϵ. We also assume there is an algorithm AP that, for any function f (under some conditions specified later), can produce an exponential sum $\widehat{f}(x) = \sum_{i=1}^{L} c_i \phi_i^x$ which is an ϵ-approximation of $f(x)$ in $[-\pi, \pi]$ such that $|\phi_i| \leq 1$ and L depends only on ϵ and f. In fact, we can assume w.l.o.g. that AP can approximate $f(x)$ over $[-B, B]$ for any $B = O(1)$. This is because we can apply AP to the scaled version $g(x) = f(x \cdot \frac{B}{\pi})$ (which is defined on $[-\pi, \pi]$) and then scale the obtained approximation $\widehat{g}(x)$ back to $[-B, B]$, i.e., the final approximation is $\widehat{f}(x) = \widehat{g}(\frac{\pi}{B} \cdot x)$. Scaling a function by a constant factor $\frac{B}{\pi}$ typically does not affect the smoothness of f in any essential way and we can still apply AP. Recall that our goal is to produce an exponential sum that is an ϵ-approximation for $\mu(x)$ in $[0, +\infty)$. We denote this procedure by ESUM.

Algorithm: ESUM

1. Initially, we slightly change function $\mu(x)$ to a new function $\widehat{\mu}(x)$ as follows: We require $\widehat{\mu}(x)$ is a "smooth " function in $[-2T_\epsilon, 2T_\epsilon]$ such that $\widehat{\mu}(x) = \mu(x)$ for all $x \in [0, T_\epsilon]$; $\widehat{\mu}(x) = 0$ for $|x| \geq 2T_\epsilon$. We choose $\widehat{\mu}(x)$ in $[-2T_\epsilon, 0]$ and $[T_\epsilon, 2T_\epsilon]$ such that $\widehat{\mu}(x)$ is smooth. We do not specify the exact smoothness requirements now since they may depend on the choice of AP. Note that there may be many ways to interpolate μ such that the above conditions are satisfied (see Example 15 below). The only properties we need are: (1) $\widehat{\mu}$ is amenable to algorithm AP; (2) $|\widehat{\mu}(x) - \mu(x)| \leq \epsilon \quad \forall x \geq 0$.

2. We apply AP to $f(x) = \eta^x \widehat{\mu}(x)$ over domain $[-hT_\epsilon, hT_\epsilon]$ ($\eta \geq 1$ and $h \geq 2$ are constants to be determined later). Suppose the resulting exponential sum $\widehat{f}(x) = \sum_{i=1}^{L} c_i \phi_i^x$ which is an ϵ-approximation of f on $[-hT_\epsilon, hT_\epsilon]$.

3. Let $\widetilde{\mu}(x) = \sum_{i=1}^{L} c_i (\frac{\phi_i}{\eta})^x$, which is our final approximation of $\mu(x)$ on $[0, \infty)$.

Example 15. *Consider the utility function $\mu(x) = 1/(x+1)$. Let $T_\epsilon = \frac{1}{\epsilon} - 1$. So $\mu(x) < \epsilon$ for all $x > T_\epsilon$. Now we create function $\widehat{\mu}(x)$ according to the first step of* ESUM. *If we only require $\widehat{\mu}(x)$ to be continuous, then we can use, for instance, the following piecewise function: $\widehat{\mu}(x) = \frac{1}{x+1}, x \in [0, T_\epsilon]; \widehat{\mu}(x) = \frac{x}{\epsilon T} + \frac{2}{\epsilon}, x \in [T_\epsilon, 2T_\epsilon]; \widehat{\mu}(x) = 0, x > 2T_\epsilon; \widehat{\mu}(x) = -\widehat{\mu}(x), x < 0$. It is easy to see that $\widehat{\mu}$ is continuous and ϵ-approximates μ.* □

By setting $\eta = 2$ and

$$h \geq \frac{\log(\sum_{i=1}^{L} |c_i|/\epsilon)}{T_\epsilon}, \tag{8.3}$$

we can show the following theorem.

Lemma 9. *$\widetilde{\mu}(x)$ is a 2ϵ-approximation of $\mu(x)$.*

Proof: We know that $|\widehat{f}(x) - f(x)| \leq \epsilon$ for $x \in [0, hT_\epsilon]$. Therefore, we have that

$$|\widetilde{\mu}(x) - \widehat{\mu}(x)| = |\frac{\widehat{f}(x)}{\eta^x} - \frac{f(x)}{\eta^x}| \leq \frac{\epsilon}{\eta^x} \leq \epsilon.$$

Combining with $|\widehat{\mu}(x) - \mu(x)| \leq \epsilon$, we obtain $|\widetilde{\mu}(x) - \mu(x)| \leq 2\epsilon$ for $x \in [0, hT_\epsilon]$. For $x > hT_\epsilon$, we can see that

$$|\widetilde{\mu}(x)| = |\sum_{i=1}^{L} c_i(\frac{\phi_i}{\eta})^x| \leq \sum_{i=1}^{L} |c_i(\frac{\phi_i}{\eta})^x| \leq \frac{1}{2^x}\sum_{i=1}^{L}|c_i| \leq \frac{1}{2^{hT_\epsilon}}\sum_{i=1}^{L}|c_i| \leq \epsilon$$

Since $\mu(x) < \epsilon$ for $x > hT_\epsilon$, the proof is complete. □

Remark: Since we do not know c_i before applying AP, we need to set h to be a constant (only depending on μ and ϵ) such that (8.3) is always satisfied. In particular, we need to provide an upper bound for $\sum_{i=1}^{L}|c_i|$. In the next subsection, we use the Fourier series decomposition as the choice for AP, which allows us to provide such a bound for a large class of functions.

8.2.3 A Particular Choice of AP: The Fourier Series Approach

Now, we discuss the choice of algorithm AP and the conditions that $f(x)$ needs to satisfy so that it is possible to approximate $f(x)$ by a short exponential sum in a bounded interval. In fact, if we know in advance that there is a short exponential sum that can approximate f, we can use the algorithms developed in [25, 27] (for continuous case) and [24] (for discrete case). However, those works do not provide an easy characterization of the class of functions. From now on, we restrict ourselves to the classic Fourier series technique, which has been studied extensively and allows such characterizations.

Consider the partial sum of the Fourier series of the function $f(x)$:

$$(S_N f)(x) = \sum_{k=-N}^{N} c_k e^{ikx}$$

where the Fourier coefficient $c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-ikx}\mathrm{d}x$. It has $L = 2N + 1$ terms. Since $f(x)$ is a real function, we have $c_k = c_{-k}$ and the partial sum is also real. We are interested in the question under which conditions does the function $S_N f$ converge to f (as N increases) and what is convergence rate? Roughly speaking, the more "smooth" f is, the faster $S_N f$ converges to f. In general, this question is extremely intricate and deep and is one of the central topics in the area of harmonic analysis. In the following, we give one classic result about the convergence of Fourier series and show how to use it in our problem. Then we provide a few concrete examples.

We say f satisfies the *α-Hölder condition* if $|f(x) - f(y)| \leq C\,|x - y|^{\alpha}$, for some constant C and $\alpha > 0$ and any x and y. The constant C is called the *Hölder coefficient* of f, also denoted as $|f|_{C^{0,\alpha}}$. We say f is *C-Lipschitz* if f satisfies 1-Hölder condition with coefficient C.

Example 16. *It is easy to check that the utility function μ in Example 15 is 1-Lipschitz since $|\frac{\mathrm{d}\mu(x)}{\mathrm{d}x}| \leq 1$ for $x \geq 0$. We can also see that (8.1) is $\frac{1}{\delta}$-Lipschitz.*

We need the following classic result of Jackson.

Theorem 8.4. (See e.g., [149]) *If f satisfies the α-Hölder condition, it holds that*

$$|f(x) - (S_N f)(x)| \leq O\Big(\frac{|f|_{C^{0,\alpha}} \ln N}{N^{\alpha}}\Big).$$

For later development, we need a few simple lemmas. The proofs of these lemmas are straightforward and thus omitted here.

Lemma 10. *Suppose $f : [a, c] \to \mathbb{R}$ is a continuous function which consists of two pieces $f_1 : [a, b] \to \mathbb{R}$ and $f_2 : [b, c] \to \mathbb{R}$. If both f_1 and f_2 satisfy the α-Hölder condition with Hölder coefficient C, then $|f|_{C^{0,\alpha}} \leq 2C$.*

Lemma 11. *Suppose $f : [a, c] \to \mathbb{R}$ is a continuous function satisfying the α-Hölder condition with Hölder coefficient C. Then, for $g(x) = f(hx)$ for some constant h, we have $|g|_{C^{0,\alpha}} \leq Ch^{\alpha}$.*

Using Theorem 8.4 and Lemma 11, we obtain the following corollary.

Corollary 3. *Suppose* $f \in C^0[-hT_\epsilon, hT_\epsilon]$ *satisfies the* α*-Hölder condition with* $|f|_{C^{0,\alpha}} = O(1)$ *and* $N = O\big(hT_\epsilon(\frac{1}{\epsilon}\log\frac{1}{\epsilon})^{1/\alpha}\big)$. *Then, it holds that* $|f(x) - (S_N f)(x)| \leq \epsilon$ *for* $x \in [-hT_\epsilon, hT_\epsilon]$.

Everything is in place to prove Theorem 8.2. Consider the algorithm AP. If μ is α-Hölder with coefficient $O(1)$, we can construct $\widehat{\mu}$ which is also α-Hölder with coefficient $O(1)$, by Lemma 10. Then, we can easily see that $f(x) = \eta^x\widehat{\mu}(x)$ is also α-Hölder with coefficient $O(1)$ in $[-hT_\epsilon, hT_\epsilon]$ for any $\eta = 2$. Hence, we can apply Corollary 3. By Lemma 9, we complete the proof of Theorem 8.2.

How to Choose h: Now, we discuss the issue left in Section 8.2.2, that is how to choose h (the value should be independent of c_is and L) to satisfy (8.3), when μ satisfies the α-Hölder condition for some $\alpha > 1/2$. Indeed, we can choose $h = O(\frac{1}{T_\epsilon}\log\frac{1}{\epsilon})$.

8.2.4 Computing $\mathbb{E}[\alpha^{w_e}]$

If X is a random variable, then the *characteristic function* of X is defined as

$$G(z) = \mathbb{E}[e^{izX}].$$

We can see $\mathbb{E}[\alpha^{w_e}]$ is nothing but the value of the characteristic function of w_e evaluated at $-i\ln\alpha$ (here ln is the complex logarithm function). For many important distributions, including negative binomial, Poisson, exponential, Gaussian, Chi-square and Gamma, a closed-form characteristic function is known. See [142] for a more comprehensive list.

Example 17. *Consider the Poisson distributed* w_e *with mean* λ, *i.e.,* $\Pr(w_e = k) = \lambda^k e^{-\lambda}/k!$. *Its characteristic function is known to be* $G(z) = e^{\lambda(e^{iz}-1)}$. *Therefore,*

$$\mathbb{E}[\alpha^{w_e}] = G(-i\ln\alpha) = e^{\lambda(\alpha-1)}.$$

Example 18. *For Gaussian distribution $N(\mu, \sigma^2)$, we know its characteristic function is $G(z) = e^{iz\mu - \frac{1}{2}\sigma^2 z^2}$. Therefore,*

$$\mathbb{E}[\alpha^{w_e}] = G(-i \ln \alpha) = \alpha^{u + \frac{1}{2}\sigma^2 \ln \alpha}.$$

For some continuous distributions, no closed-form characteristic function is known and we need proper numerical approximation method.

If the support of the distribution is bounded, we can use for example Gauss-Legendre quadrature [150] (see Section 6.2.3 for a brief description). If the support is infinite, we can truncate the distribution and approximate the integral over the remaining finite interval. A typical practice is to use *composite rule*, i.e., to partition $[a, b]$ into N subintervals and approximate the integral using some quadrature formula over each subinterval. Assuming continuity of the $2k$th derivative of $f(x)$ for some constant k, if we partition $[a, b]$ into M subintervals and apply Gauss-Legendre quadrature of degree k to each subinterval, the approximation error is

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

where ξ is some point in (a, b) [150, pp.116]. Let $\Delta = \frac{b-a}{M}$. If we treat k as a constant, the behavior of the error (in terms of Δ) is $Error(\Delta) = O(\Delta^{2k} \max_\xi f^{(2k)}(\xi))$. Therefore, if the support and $\max_\xi f^{(2k)}(\xi)$ are bounded by a polynomial, we can approximate the integral, in polynomial time, such that the error is $O(1/n^\beta)$ for any fixed integer β.

The next lemma shows that we do not lose too much even though we can only get an approximation of $\mathbb{E}[\alpha^{w_e}]$.

Lemma 12. *Suppose in Theorem 8.3, we can only compute an approximate value of $\mathbb{E}[\phi_i^{w_e}]$, denoted by $E_{e,i}$, for each e and i, such that $|\mathbb{E}[\phi_i^{w_e}] - E_{e,i}| \leq O(n^{-\beta})$ for some positive integer β. Denote $E(S) = \sum_{k=1}^{L} c_k \prod_{e \in S} E_{e,i}$. For any solution S,*

we have that

$$|\mathbb{E}[\widetilde{\mu}(w(S))] - E(S)| \leq O(n^{1-\beta}).$$

Proof: We need the following simple result (see [125] for a proof): $a_1, \ldots, a_n$ and $e_1, \ldots, e_n$ are complex numbers such that $|a_i| \leq 1$ and $|e_i| \leq n^{-\beta}$ for all i and some $\beta > 1$. Then, we have

$$\left|\prod_{i=1}^{n}(a_i + e_i) - \prod_{i=1}^{n} E_i\right| \leq O(n^{1-\beta}).$$

Since $|\phi_i| \leq 1$, we can see that

$$|\mathbb{E}[\phi_i^{w_e}]| = |\int_{x \geq 0} \phi_i^x p_e(x) \mathrm{d}x| \leq 1.$$

The lemma simply follows by applying the above result and noticing that L and all c_ks are constants. □

We can show that Theorem 8.1 still holds even though we only have the approximations of the $\mathbb{E}[\alpha^{w_e}]$ values. The proof is straightforward and omitted.

8.3 Applications

We first consider two utility functions $\chi(x)$ and $\widetilde{\chi}(x)$ presented in the introduction. Note that maximizing $\mathbb{E}[\chi(w(S))]$ is equivalent to maximizing $\Pr(w(S) \leq 1)$. The following lemma is straightforward.

Lemma 13. *For any solution S,*

$$\Pr(w(S) \leq 1) \leq \mathbb{E}[\widetilde{\chi}(w(S))] \leq \Pr(w(S) \leq 1 + \delta).$$

Corollary 4. *Suppose there is a pseudopolynomial time algorithm for the exact version of A. Then, for any fixed constants $\epsilon > 0$ and $\delta > 0$, there is an algorithm*

that runs in time $(\frac{n}{\epsilon})^{O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$*, and produces a solution* $S \in \mathcal{F}$ *such that*

$$\Pr(w(S) \leq 1+\delta) + \epsilon \geq \max_{S' \in \mathcal{F}} \Pr(w(S') \leq 1)$$

Proof: By Theorem 8.1, Theorem 8.2 and Lemma 13, we can easily obtain the corollary. Note that we can choose $T_\epsilon = 2$ for any $\epsilon > 0$. Thus $h = O(\log\frac{1}{\epsilon})$ and $L = O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})$. □

Now, let us see some applications of our general results to specific problems.

8.3.1 Top-k Query with Set Interpretation (Top-SI)

Imagine a top-k query where we would like to return k tuples with the smallest total weight (the order of these tuples does not matter). However, the weights of the tuples are uncertain. In this case, we can define the top-k semantics under the expected utility maximization principle, i.e., to find the size-k subset maximizing the expected utility, where the utility function is a function of the total weight of the subset. It is not hard to see that the exact version of the problem in the deterministic setting, i.e., to find a size-k set of tuples with a given target weight, can be solved in pseudopolynomial time by dynamic programming. Thus, our result directly gives us a way to maximize the expected utility for a utility function satisfying the condition of Theorem 8.2.

8.3.2 Stochastic Shortest Path

Finding a path with the exact target length (we allow non-simple paths)[3] can be easily done in pseudopolynomial time by dynamic programming. Therefore, as discussed in Section 8.1.1, Corollary 4 generalizes several results for stochastic shortest path in prior work [141, 139].

[3]The exact version of *simple* path is NP-hard, since it includes the Hamiltonian path problem as a special case.

8.3.3 Stochastic Spanning Tree

Our objective is to find a spanning tree T in the given probabilistic graph such that $\Pr(w(T) \leq 1)$ is maximized. Polynomial time algorithms have been developed for Gaussian distributed edges [102, 77]. To the best of our knowledge, no approximation algorithm with provable guarantee is known for other distributions. Noticing there exists a pseudopolynomial time algorithm for the exact spanning tree problem [19], we can directly apply Corollary 4.

8.3.4 Stochastic k-Median on Trees

The problem asks for a set S of k nodes in the given probabilistic tree G such that $\Pr(\sum_{v \in V(G)} \mathsf{dis}(v, S) \leq 1)$ is maximized, where $\mathsf{dis}(v, S)$ is the minimum distance from v to any node in S in the tree metric. The k-median problem can be solved optimally in polynomial time on trees by dynamic programming [113]. In fact, we can easily modify the dynamic program to get a pseudopolynomial time algorithm for the exact version. We omit the details.

8.3.5 Stochastic Knapsack with Random Sizes

We are given a set U of n items. Each item i has a random size w_i and a deterministic profit v_i. We are also given a positive constant $0 \leq \gamma \leq 1$. The goal is to find a subset $S \subseteq U$ such that $\Pr(w(S) \leq 1) \geq \gamma$ and the total profit $v(S) = \sum_{i \in S} v_i$ is maximized.

If the profits of the items are polynomially bounded integers, we can see the optimal profit is also a polynomially bounded integer. We can first guess the optimal profit. For each guess g, we solve the following problem: find a subset S of items such that the total profit of S is exactly g and $\mathbb{E}[\widetilde{\chi}(w(S))]$ is maximized. The exact version of the deterministic problem is to find a solution S with a given total size and a given total profit, which can be easily solved in pseudopolynomial time by dynamic programming. Therefore, by Corollary 4, we can easily show that

we can find in polynomial time a set S of items such that the total profit $v(S)$ is at least the optimum and $\Pr(w(S) \leq 1+\epsilon) \geq (1-\epsilon)\gamma$ for any constant ϵ and γ.

If the profits are general integers, we can use the standard scaling technique to get a $(1-\epsilon)$-approximation for the total profit as follows. We first make a guess of the optimal profit, rounded down to the nearest power of $(1+\epsilon)$. There are at most $\log_{1+\epsilon} \frac{n \max_i v_i}{\min_i v_i}$ guesses. For each guess g, we solve the following problem. We discard all items with a profit larger than g. Let $\Delta = \frac{\epsilon g}{n^2}$. For each item with a profit smaller than $\frac{\epsilon g}{n}$, we set its new profit to be $\bar{v}_i = 0$. Then, we scale each of the rest profits v_i to $\bar{v}_i = \Delta \lfloor \frac{v_i}{\Delta} \rfloor$. Now, we define the feasible set

$$\mathcal{F}(g) = \{S \mid \sum_{i \in S} (1-2\epsilon)g \leq \sum_{i \in S} \bar{v}_i \leq (1+2\epsilon)g\}.$$

Since there are at most $\frac{n^2}{\epsilon}$ distinct $\bar{v}$ values, we can easily show that finding a solution S in $\mathcal{F}(g)$ with a given total *size* can be solved in pseudopolynomial time by dynamic programming.

Denote the optimal solution by S^* and the optimal profit by OPT. Suppose g is the right guess, i.e., $(\frac{1}{1+\epsilon})OPT \leq g \leq OPT$. We can easily see that for any solution S, we have that

$$(1-\frac{1}{n}) \sum_{i \in S} v_i - \epsilon g \leq \sum_{i \in S} \bar{v}_i \leq \sum_{i \in S} v_i$$

where the first inequalities are due to $v_i \geq \frac{\epsilon g}{n}$ and we set at most ϵg profit to zero. Therefore, we can see $S^* \in \mathcal{F}(g)$. Applying Corollary 4, we obtain a solution S such that $\Pr(w(S) \leq 1+\delta) + \epsilon \geq \Pr(w(S^*) \leq 1+\delta)$. Moreover, the profit of this solution $v(S) = \sum_{i \in S} v_i \geq \sum_{i \in S} \bar{v}_i \geq (1-2\epsilon)g \geq (1-O(\epsilon))OPT$. In sum, we have obtained the following result.

Theorem 8.5. *For any constants $\epsilon > 0$ and $\gamma > 0$, there is a polynomial time algorithm to compute a set S of items such that the total profit $v(S)$ is within a $1-\epsilon$ factor of the optimum and $\Pr(w(S) \leq 1+\epsilon) \geq (1-\epsilon)\gamma$.*

Recently, Bhalgat et al. [29, Theorem 8.1] obtained the same result, with a running time $n^{2^{poly(1/\epsilon)}}$, while our running time is $(\frac{n}{\epsilon})^{O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})} = n^{poly(1/\epsilon)}$.

Moreover, we can easily extend our algorithm to generalizations of the knapsack problem if the corresponding exact version has a pseudopolynomial time algorithm. For example, we can get the same result for the partial-ordered knapsack problem with tree constraints [74, 156]. In this problem, items must be chosen in accordance with specified precedence constraints and these precedence constraints form a partial order and the underlining undirected graph is a tree (or forest). A pseudopolynomial algorithm for this problem is presented in [156].

8.3.6 Stochastic Knapsack with Random Profits

We are given a set U of n items. Each item i has a deterministic size w_i and a random profit v_i. The goal is to find a subset of items that can be packed into a knapsack with capacity 1 and the probability that the profit is at least a given threshold T is maximized. Henig [95] and Carraway et al. [36] studied this problem for normally distributed profits and presented dynamic programming and branch and bound heuristics to solve this problem optimally.

We can solve the equivalent problem of minimizing the probability that the profit is at most the given threshold. It is straightforward to modify our algorithm to work for the minimization problem and we can also get an ϵ additive error for any $\epsilon > 0$. In fact, we can show that violation of the capacity constraint is necessary unless $P = NP$. Consider the following knapsack instance. The profit of each item is the same as its size. The given threshold is 1. We can see that the optimal probability is 1 if and only if there is a subset of items of total size exactly 1. Otherwise, the optimal probability is 0. Therefore, it is NP-hard to approximate the original problem within any additive error less than 1 without violating the capacity constraint.

The corresponding exact version of the deterministic problem is to find a set of items S such that $w(S) \leq 1$ and $v(S)$ is equal to a given target value. In fact,

there is no pseudopolynomial time algorithm for this problem. Since otherwise we can get an ϵ additive approximation without violating the capacity constraint, contradicting the lower bound argument. Note that a pseudopolynomial time algorithm here should run in time polynomial in the profit value(not the size). However, if the sizes can be encoded in $O(\log n)$ bits (we only have a polynomial number of different sizes), we can solve the problem in time polynomial in n and the largest profit value by standard dynamic programming.

For general sizes, we can round the size of each item down to the nearest multiple of $\frac{\epsilon}{n}$. Then, we can solve the exact version in pseudopolynomial time by dynamic programming. It is easy to show that for any subset of items, its total size is at most the total rounded size plus ϵ. Therefore, the total size of our solution is at most $1 + \epsilon$.

Theorem 8.6. *If the optimal probability is* $\Omega(1)$*, we can find in polynomial time a subset* S *of items such that* $\Pr(v(S) > (1-\epsilon)T) \geq (1-\epsilon)OPT$ *and* $w(S) \leq 1+\epsilon$*, for any constant* $\epsilon > 0$*.*

8.4 Extension to Multiple Utility functions

In this and the next sections, we discuss some extensions to our basic approximation scheme. We first consider optimizing a constant number of utility functions in this section. Then, we study the problem where the weight of each element is a random vector in Section 8.5.

The problem we study in this section contains a set U of n elements. Each element e has a random weight w_e. We are also given d utility functions $\mu_1, \ldots, \mu_d$ and d positive numbers $\lambda_1, \ldots, \lambda_d$. We assume d is a constant. A feasible solution consists of d subsets of elements that satisfy some property. Our objective is to find a feasible solution $S_1, \ldots, S_d$ such that $\mathbb{E}[\mu_i(w(S_i))] \geq \lambda_i$ for all $1 \leq i \leq d$.

We can easily extend our basic approximation scheme to the multiple utility functions case as follows. We decompose these utility functions into short expo-

nential sums using ESUM as before. Then, for each utility function, we maintain $(n/\epsilon)^{O(L)}$ configurations. Therefore, we have $(n/\epsilon)^{O(dL)}$ configurations in total and we would like to compute the values for these configurations. We denote the deterministic version of the problem under consideration by A. The exact version of A asks for a feasible solution $S_1, \ldots, S_d$ such that the total weight of S_i is exactly the given number t_i for all i. Following an argument similar to Lemma 8, we can easily get the following generalization of Theorem 8.1.

Theorem 8.7. *Assume that there is a pseudopolynomial algorithm for the exact version of A. Further assume that given any $\epsilon > 0$, we can ϵ-approximate each utility function by an exponential sum with at most L terms. Then, there is an algorithm that runs in time $(n/\epsilon)^{O(dL)}$ and finds a feasible solution $S_1, \ldots, S_d$ such that $\mathbb{E}[\mu_i(w(S_i)] \geq \lambda_i - \epsilon$ for $1 \leq i \leq d$, if there is a feasible solution for the original problem.*

Now let us consider two simple applications of the above theorem.

8.4.1 Stochastic Multiple Knapsack

In this problem we are given a set U of n items, d knapsacks with capacity 1, and d constants $0 \leq \gamma_i \leq 1$. We assume d is a constant. Each item i has a random size w_i and a deterministic profit v_i. Our objective is to find d disjoint subsets $S_1, \ldots, S_d$ such that $\Pr(w(S_i) \leq 1) \geq \gamma_i$ for all $1 \leq i \leq d$ and $\sum_{i=1}^{d} v(S_i)$ is maximized. The exact version of the problem is to find a packing such that the load of each knapsack i is *exactly* the given value t_i. It is not hard to show this problem can be solved in pseudopolynomial time by standard dynamic programming. If the profits are general integers, we also need the scaling technique as in stochastic knapsack with random sizes. In sum, we can get the following generalization of Theorem 8.5.

Theorem 8.8. *For any constants $d \in \mathbb{N}$, $\epsilon > 0$ and $0 \leq \gamma_i \leq 1$ for $1 \leq i \leq d$, there is a polynomial time algorithm to compute d disjoint subsets $S_1, \ldots, S_d$ such that*

the total profit $\sum_{i=1}^{d} v(S_i)$ *is within a* $1-\epsilon$ *factor of the optimum and* $\Pr(w(S_i) \leq 1+\epsilon) \geq (1-\epsilon)\gamma_i$ *for* $1 \leq i \leq d$.

8.4.2 Stochastic Multidimensional Knapsack

In this problem we are given a set U of n items and a constant $0 \leq \gamma \leq 1$. Each item i has a deterministic profit v_i and a random size which is a random d-dimensional vector $\mathbf{w}_i = \{w_{i1}, \ldots, w_{id}\}$. We assume d is a constant. Our objective is to find a subset S of items such that $\Pr(\bigwedge_{j=1}^{d}(\sum_{i \in S} w_{ij} \leq 1)) \geq \gamma$ and total profit is maximized. This problem can be also thought as the fixed set version of the stochastic packing problem considered in [58, 29]. We first assume the components of each size vector are independent. The correlated case will be addressed in the next subsection.

For ease of presentation, we assume $d = 2$ from now on. Extension to general constant d is straightforward. We can solve the problem by casting it into a multiple utility problem as follows. For each item i, we create two copies i_1 and i_2. The copy i_j has a random weight w_{ij}. A feasible solution consists of two sets S_1 and S_2 such that S_1 (S_2) only contains the first (second) copies of the elements and S_1 and S_2 correspond to exactly the same subset of original elements. We enumerate all such pairs (γ_1, γ_2) such that $\gamma_1\gamma_2 \geq \gamma$ and $\gamma_i \in [\gamma, 1]$ is a power of $1-\epsilon$ for $i = 1, 2$. Clearly, there are a polynomial number of such pairs. For each pair (γ_1, γ_2), we solve the following problem: find a feasible solution S_1, S_2 such that $\Pr(\sum_{i \in S_j} w_{ij} \leq 1) \geq \gamma_j$ for all $j = 1, 2$ and total profit is maximized. Using the scaling technique and Theorem 8.7 for optimizing multiple utility functions, we can get a $(1-\epsilon)$-approximation for the optimal profit and $\Pr(\bigwedge_{j=1}^{2}(\sum_{i \in S_j} w_{ij} \leq 1)) = \prod_{j=1}^{2} \Pr(\sum_{i \in S_j} w_{ij} \leq 1) \geq (1 - O(\epsilon))\gamma_1\gamma_2 \geq (1 - O(\epsilon))\gamma$.

We note that the same result for independent components can be also obtained by using the discretization technique developed for the adaptive version of the problem in [29] [4]. If the components of each size vector are correlated, we cannot

[4] With some changes to the discretization technique, the correlated case can be also handled

decompose the problem into two 1-dimensional utilities as in the independent case. Now, we introduce a new technique to handle the correlated case.

8.5 Extension to Multidimensional Weight

The general problem we study contains a set U of n elements. Each element e has a random weight vector $w_i = (w_{i1}, \ldots, w_{id})$. We assume d is a constant. We are also given a utility functions $\mu : \mathbb{R}^d \to \mathbb{R}^+$. A feasible solution is a subset of elements satisfying some property. We use $w(S)$ as a shorthand notation for vector $(\sum_{i\in S} w_{i1}, \ldots, \sum_{i\in S} w_{id})$. Our objective is to find a feasible solution S such that $\mathbb{E}[\mu_i(w(S)]$ is maximized.

From now on, x and k denote d-dimensional vectors and kx (or $k \cdot x$) denotes the inner product of k and x. As before, we assume $\mu(x) \in [0, 1]$ for all $x \geq 0$ and $\lim_{|x|\to+\infty} \mu(x) = 0$, where $|x| = \max(x_1, \ldots, x_d)$, Our algorithm is almost the same as in the one dimension case and we briefly sketch it here. We first notice that expected utilities decompose for exponential utility functions, i.e., $\mathbb{E}[\phi^{k\cdot w(S)}] = \prod_{i\in S} \mathbb{E}[\phi^{k\cdot w_i}]$. Then, we attempt to ϵ-approximate the utility function $\mu(x)$ by a short exponential sum $\sum_{|k|\leq N} c_k \phi_k^{kx}$ (there are $O(N^d)$ terms). If this can be done, $\mathbb{E}[\phi^{k\cdot w(S)}]$ can be approximated by $\sum_{|k|\leq N} c_k \mathbb{E}[\phi^{k\cdot w(S)}]$. Using the same argument as in Theorem 8.1, we can show that there is a polynomial time algorithm that can find a feasible solution S with $\mathbb{E}[\mu(w(S))] \geq OPT - \epsilon$ for any $\epsilon > 0$, provided that a pseudopolynomial algorithm exists for the exact version of the deterministic problem.

To approximate the utility function $\mu(x)$, we need the multidimensional Fourier series expansion of a function $f : \mathbb{C}^d \to \mathbb{C}$ (assuming f is 2π-periodic in each axis): $f(x) \sim \sum_{k\in\mathbb{Z}^d} c_k e^{ikx}$ where $c_k = \frac{1}{(2\pi)^d} \int_{x\in[-\pi,\pi]^d} f(x) e^{-ikx}\, dx$. The *rectangular*

[28].

partial sum is defined to be

$$S_N f(x) = \sum_{|k_1|\le N} \cdots \sum_{|k_d|\le N} c_k e^{ikx}.$$

It is known that the rectangular partial sum $S_N f(x)$ converges uniformly to $f(x)$ in $[-\pi,\pi]^d$ for many function classes as n tends to infinity. In fact, a generalization of Theorem 8.4 to $[-\pi,\pi]^d$ also holds [9]: If f satisfies the α-Hölder condition, then

$$|f(x) - (S_N f)(x)| \le O\Big(\frac{|f|_{C^{0,\alpha}} \ln^d N}{N^\alpha}\Big) \quad \text{for } x \in [-\pi,\pi]^d.$$

Now, we have an algorithm AP that can approximate a function in a bounded domain. It is also straightforward to extend ESUM to the multidimensional case. Hence, we can ϵ-approximate μ by a short exponential sum in $[0,+\infty)^d$, thereby proving the multidimensional generalization of Theorem 8.2. Let us consider an application of our result.

8.5.1 Stochastic Multidimensional Knapsack (Revisited)

We consider the case where the components of each weight vector can be correlated. Note that the utility function χ_2 corresponding to this problem is the two dimensional threshold function: $\chi_2(x,y) = 1$ if $x \le 1$ and $y \le 1$; $\chi_2(x,y) = 0$ otherwise. As in the one dimensional case, we need to consider a continuous version $\widetilde{\chi}_2$ of χ_2 (see Figure 8.1(3)). By the result in this section and a generalization of Lemma 13 to higher dimension, we can get the following.

Theorem 8.9. *For any constants $d \in \mathbb{N}$, $\epsilon > 0$ and $0 \le \gamma \le 1$, there is a polynomial time algorithm for finding a set S of items such that the total profit $v(S)$ is $1-\epsilon$ factor of the optimum and* $\Pr(\bigwedge_{j=1}^{d}(\sum_{i\in S} w_{ij} \le 1+\epsilon)) \ge (1-\epsilon)\gamma$.

8.6 Discussions

Convergence of Fourier series: The convergence of the Fourier series of a function is a classic topic in harmonic analysis. Whether the Fourier series converges to the given function and the rate of the convergence typically depends on a variety of smoothness condition of the function. We refer the readers to [169] for a more comprehensive treatment of this topic. We note that we could obtain a smoother version of χ (e.g., see Figure 8.1(2)), instead of the piecewise linear $\widetilde{\chi}$, and then use Theorem 8.4 to obtain a better bound for L. This would result in an even better running time. Our choice is simply for the ease of presentation.

Discontinuous utility functions: If the utility function μ is discontinuous, e.g., the threshold function, then the partial Fourier series behaves poorly around the discontinuity (this is known as the *Gibbs phenomenon*). However, informally speaking, as the number of Fourier terms increases, the poorly-behaved strip around the edge becomes narrower. Therefore, if the majority of the probability mass of our solution lies outside the strip, we can still guarantee a good approximation of the expected utility. There are also techniques to reduce the effects of the Gibbs phenomenon (See e.g., [81]). We leave the problem of directly dealing with discontinuous utility functions, especially the threshold function, to obtain a true approximation (instead of a bi-criterion approximation) as an interesting open problem.

Chapter 9

Stochastic Matchings

9.1 Introduction

Motivated by applications in kidney exchanges and online dating, Chen et al. [40] proposed the following stochastic matching problem: we want to find a maximum matching in a random graph G on n nodes, where each edge $(i, j) \in [\binom{n}{2}]$ exists with probability p_{ij}, independently of the other edges. However, all we are given are the probability values $\{p_{ij}\}$. To find out whether the random graph G has the edge (i, j) or not, we have to try to add the edge (i, j) to our current matching (assuming that i and j are both unmatched in our current partial matching)—we call this "probing" edge (i, j). As a result of the probe, we also find out if (i, j) exists or not—and if the edge (i, j) indeed exists in the random graph G, it gets irrevocably added to M. Such policies make sense, e.g., for dating agencies, where the only way to find out if two people are actually compatible is to send them on a date; moreover, if they do turn out to be compatible, then it makes sense to match them to each other (see Section 1.3 for the details of the motivations). Finally, to model the fact that there might be a limit on the number of unsuccessful dates a person might be willing to participate in, "timeouts" on vertices are also provided. More precisely, valid policies are allowed, for each vertex i, to only probe at most t_i edges incident to i. Similar considerations arise in kidney exchanges, details of which can be found in Section 1.3.

Chen et al. [40] asked the question: how can we devise probing policies to maximize the expected cardinality (or weight) of the matching? They showed that the greedy algorithm that probes edges in decreasing order of p_{ij} (as long as their

endpoints had not timed out) was a 4-approximation to the unweighted version (i.e., all edges have the same weight 1) of the stochastic matching problem. Quite recently, Adamczyk has proved that the greedy algorithm is a 2-approximation for unweighted stochastic matching [4]. However, this greedy algorithm (and other simple greedy schemes) can be seen to be arbitrarily bad in the presence of weights, and they left open the question of obtaining good algorithms to *maximize the expected weight* of the matching produced. In addition to being a natural generalization, weights can be used as a proxy for revenue generated in matchmaking services. (The unweighted case can be thought of as maximizing the social welfare.) In this chapter, we resolve the main open question from Chen et al. [40] by obtaining constant approximations for the weighted stochastic matching problems.

9.1.1 Our Contributions

First, we consider a more general problem, called *stochastic k-set-packing*, where we try to pack k-hyperedges with random sizes and profits into a d-dimensional knapsack of a given size. The stochastic k-set-packing problem is a direct generalization of the stochastic matching problem (for $k = 4$; See the reduction in Section 9.2). We also note that this is a slight generalization of the stochastic b-matching problem of [58]. In particular, our model allows *correlations* between the profit of an item and its size-vector, whereas in [58] the profit of each item is fixed (or independent of its size-vector). Indeed, it is the discreteness in the sizes (i.e., $0-1$ values) that allows the LP-based approach to work for stochastic k-set-packing; if the instantiations were allowed to be in $[0, 1]$ then the LP has a large integrality gap even with just one constraint (see e.g., Appendix A of [92]). Moreover, our focus is on the situation where $k \ll d$. For this setting of parameters, we improve on the $\sqrt{d}$-approximation of [58] (which only holds for independent profits and sizes) by showing the following (Section 9.2).

Theorem 9.1. *There is a $2k$-approximation algorithm for the weighted stochastic*

k-set-packing problem. When the column outcomes are monotone, there is a $k+1$ approximation algorithm.

Our main idea is to use the knowledge of item probabilities to solve a linear program where each item e has a variable $0 \leq y_e \leq 1$ corresponding to the probability that a strategy packs e (over all possible realizations of the hypergraph). This is similar to the approach for stochastic packing problems considered by Dean et al. [59, 58]. Our improved approximation for monotone column outcomes is obtained using the FKG inequality to strengthen the probability bound. Our usage of the FKG inequality is similar to that in [162]. The second part of Theorem 9.1 also implies a simple 5-approximation for stochastic matching. However, using more structure in the matching problem, we could obtain the following better approximation ratios.

Theorem 9.2. *There is a 4-approximation algorithm for the weighted stochastic matching problem. For bipartite graphs, there is a 3-approximation algorithm.*

The improved approximations use the same linear program as before, but more involved rounding methods to decide which edges to probe. The rounding procedure for bipartite graphs uses *dependent rounding* [73] on the y-values to obtain a set $\widehat{E}$ of edges to be probed, and then probes edges of $\widehat{E}$ in a uniformly random order. For non-bipartite graphs, the algorithm first samples a random bipartite subgraph and then applies the bipartite rounding algorithm on it.

The probing strategy returned by the algorithm can in fact be made *matching-probing* [40]. In this alternative (more restrictive) probing model we are given an additional parameter k and edges need to be probed in k rounds, each round being a matching. It is clear that this matching-probing model is more restrictive than the usual *edge-probing* model (with timeouts $\min\{t_i, k\}$) where one edge is probed at a time. Our algorithm obtains a matching-probing strategy that is only a small constant factor worse than the optimal edge-probing strategy; hence, we also obtain the same constant approximation guarantee for weighted stochastic

matching in the matching-probing model. It is worth noting that previously only a logarithmic approximation in the unweighted case was known [40].

Theorem 9.3. *There is a* 4*-approximation algorithm for the weighted stochastic matching problem in the* matching-probing *model. For bipartite graphs, there is a* 3*-approximation algorithm.*

Apart from solving these open problems and yielding improved approximations, our LP-based analysis turns out to be applicable in a wider context.

Online Stochastic Matching with Timeouts: In a bipartite graph $(A, B; E)$ of items $i \in A$ and potential buyer types $j \in B$, p_{ij} denotes the probability that a buyer of type j will buy item i. A sequence of n buyers are to arrive online, where the type of each buyer is an i.i.d. sample from B according to some pre-specified distribution—when a buyer of type j appears, he can be shown a list L of up to t_j as-yet-unsold items, and the buyer buys the *first* item on the list according to the given probabilities $p_{\cdot,j}$. (Note that with probability $\prod_{i \in L}(1 - p_{ij})$, the buyer leaves without buying anything.) What items should we show buyers when they arrive online, and in which order, to maximize the expected weight of the matching? Building on the algorithm for stochastic matching in Section 9.2, we prove the following in Section 9.4.

Theorem 9.4. *There is a* $\frac{6e^2}{2e^2-e-1} \approx 4.008$*-approximation algorithm for the online stochastic matching problem with timeouts.*

This question is an extension of similar online stochastic matching questions considered earlier in [68]—in that paper, $w_{ij}, p_{ij} \in \{0, 1\}$ and $t_j = 1$. Our model tries to capture the facts that buyers may have a limited attention span (using the timeouts), they might have uncertainties in their preferences (using edge probabilities), and that they might buy the first item they like rather than scanning the entire list.

The results in this chapter are mainly based on [16, 17]. Theorem 9.4 improves Theorem 4 in [17].

A note on optimal solutions: We must clarify here the notion of an optimal solution. In standard worst case analysis we would compare our solution against the optimal *offline* solution, e.g. the value of the maximum matching, where the offline knows all the edge instantiations in advance (i.e. which edge will appear when probed, and which will not). However, it can be easily verified that due to the presence of timeouts, this adversary is too strong [40]. Consider the following example. Suppose we have a star where each vertex has timeout 1, and each edge has $p_{ij} = 1/n$. The offline optimum can match an edge whenever the star has an edge i.e. with probability about $1 - 1/e$, while our algorithm can only get expected $1/n$ profit, as it can only probe a single edge. Hence, for all problems in this paper we consider the setting where even the optimum does not know the exact instantiation of an edge until it is probed. This gives our algorithms a level playing field. The optimum thus corresponds to a "strategy" of probing the edges, which can be chosen from an exponentially large space of potentially adaptive strategies.

We note that our algorithms in fact yield *non-adaptive* strategies for the corresponding problems, that are only constant factor worse than the adaptive optimum. This is similar to previous results on stochastic packing problems: knapsack (Dean et al. [59, 58]) and multi-armed bandits (Guha-Munagala [89, 90] and references therein).

9.2 Stochastic k-Set Packing

We first consider a generalization of the stochastic matching problem to hypergraphs, where each edge has size at most k. Formally, the input to this *stochastic k-set packing* problem consists of

- n items/columns, where each item has a random profit $v_i \in \mathbb{R}_+$, and a random d-dimensional size $S_i \in \{0,1\}^d$; these random values and sizes are drawn from a probability distribution specified as part of the input. We

note that the size-vector S_i and profit v_i of each item i are allowed to be correlated (this is what distinguishes our model from [58]). The probability distributions for different items are independent. Additionally, for each item, there is a set C_i of at most k coordinates such that each size vector takes positive values only in these coordinates; i.e., $S_i \subseteq C_i$ with probability 1 for each item i.

- A capacity vector $b \in \mathbb{Z}_+^d$ into which the items must be packed.

The parameter k is called the *column sparsity* of the problem. The instantiation of any column (i.e., its size and profit) is known only when it is probed. The goal is to compute an adaptive strategy of choosing items until there is no more available capacity such that the expectation of the obtained profit is maximized.

Note that the stochastic matching problem can be modeled as a stochastic 4-set packing problem in the following way: we set $d = 2n$, and associate the i^{th} and $(n+i)^{th}$ coordinate with the vertex i—the first n coordinates capture whether the vertex is free or not, and the second n coordinates capture how many probes have been made involving that vertex. For any $t \in [d]$, let $e_t \in \{0,1\}^d$ denote the indicator vector with a single 1 in the t^{th} position. Now each edge (i,j) is an item which has the following distribution: with probability p_{ij} the value is w_{ij} and size is $e_i + e_j + e_{n+i} + e_{n+j}$, and with remaining probability $1 - p_{ij}$ the value is 0 and size is $e_{n+i} + e_{n+j}$. Note that for each item, its size and value are correlated. If we set the capacity vector to be $b = (1, 1, \cdots, 1, t_1, t_2, \cdots, t_n)$, this precisely captures the stochastic matching problem. In this special case each size vector has $\leq k = 4$ ones.

This stochastic k-set packing problem was studied (among many others) as the "stochastic b-matching" problem in Dean et al. [58]; however their model assumed deterministic values of items, so their results do not apply here directly. Moreover the authors of that work did not consider the 'column sparsity' parameter k and instead gave an $O(\sqrt{d})$-approximation algorithm for the general case. Here we

consider the performance of algorithms for this problem specifically as a function of the column sparsity k, and prove Theorem 9.1.

A quick aside about "safe" and "unsafe" adaptive policies: a policy is called *safe* if it can include an item only if there is *zero* probability of violating any capacity constraint. In contrast, an *unsafe* policy may attempt to include an item even if there is non-zero probability of violating capacity—however, if the random size of the item causes the capacity to be violated, then no profit is received for the overflowing item, and moreover, no further items may be included by the policy. The model in Dean et al. [58] allowed unsafe policies, whereas we are interested in safe policies. However, due to the discreteness of sizes in stochastic k-set packing, it can be shown that our approximation guarantee is relative to the optimal unsafe policy (see Subsection 9.2.2).

For each item $i \in [n]$ and constraint $j \in [d]$, let $\mu_i(j) := \mathbb{E}[S_i(j)]$, the expected value of the j^{th} coordinate in size-vector S_i. For each column $i \in [n]$, the coordinates $\{j \in [d] \mid \mu_i(j) > 0\}$ are called the *support* of column i. By column sparsity, the support of each column has size at most k. Also, let $w_i := \mathbb{E}[v_i]$, the mean profit, for each $i \in [n]$. We now consider the natural LP relaxation for this problem, as in [58].

$$\text{maximize} \sum_{i=1}^{n} w_i \cdot y_i \tag{LP1}$$

subject to

$$\sum_{i=1}^{n} \mu_i(j) \cdot y_i \leq b_j \qquad \forall j \in [d] \tag{9.1}$$

$$y_i \in [0,1] \qquad \forall i \in [n] \tag{9.2}$$

The following claim shows that the LP above is a valid relaxation for the stochastic k-set-packing problem.

Claim 1. *The optimal value for* (LP1) *is an upper bound on any (adaptive) algorithm for stochastic k-set-packing.*

Proof: Let p_i be the probability that an adaptive strategy A packs item i. To show the claim, it suffices to show that p_is satisfy the constraints (9.1) for any adaptive strategy A. Consider the jth constraint. Conditioned on any instantiation of all items, A can pack at most b_j items for which $\mu_i(j) = 1$, since A is a safe policy. Hence these constraints hold unconditionally as well, which implies that any valid strategy satisfies (9.1). □

Let y^* denote an optimal solution to this linear program, which in turn gives us an upper bound on any adaptive (safe) strategy. Our rounding algorithm proceeds as follows. Fix a constant $\alpha \geq 1$, to be specified later. The algorithm picks a uniformly random permutation $\pi : [n] \to [n]$ on all columns, and probes only a subset of the columns as follows. At any point in the algorithm, column c is *safe* iff there is positive residual capacity in *all* the coordinates in the support of c—in other words, irrespective of the instantiation of S_c, it can be feasibly packed with the previously chosen columns. The algorithm inspects columns in the order of π, and whenever it is safe to probe the next column $c \in [n]$, it does so with probability $\frac{y_c}{\alpha}$. Note that the algorithm skips all columns that are unsafe at the time they appear in π.

We now prove the first part of Theorem 9.1 by showing that this algorithm is a $2k$-approximation for a suitable value of α. For any column $c \in [n]$, let $\{\mathbf{I}_{c,\ell}\}_{\ell=1}^{k}$ denote the indicator random variables for the event that the ℓ^{th} constraint in the support of c is tight at the time when c is considered under the random permutation π. Note that the event "column c is safe when considered" is precisely $\bigwedge_{\ell=1}^{k} \neg \mathbf{I}_{c,\ell}$. By a trivial union bound, the $\Pr[\, c \text{ is safe} \,] \geq 1 - \sum_{\ell=1}^{k} \Pr[\mathbf{I}_{c,\ell}]$.

Lemma 14. *For any column $c \in [n]$ and index $\ell \in [k]$, $\Pr[\mathbf{I}_{c,\ell}] \leq \frac{1}{2\alpha}$.*

Proof: Let $j \in [d]$ be the ℓ^{th} constraint in the support of c. Let U_c^j denote the usage of constraint j, when column c is considered (according to π). We have:

$$\begin{aligned}
\mathbb{E}[U_c^j] &= \sum_{a=1}^{n} \Pr[\,\text{column } a \text{ appears before } c \text{ AND } a \text{ is probed}\,] \cdot \mu_a(j), \\
&\leq \sum_{a=1}^{n} \Pr[\,\text{column } a \text{ appears before } c\,] \cdot \frac{y_a}{\alpha} \cdot \mu_a(j), \\
&= \sum_{a=1}^{n} \frac{y_a}{2\alpha} \cdot \mu_a(j) \leq \frac{b_j}{2\alpha}.
\end{aligned}$$

Since $\mathbf{I}_{c,\ell} = \{U_c^j \geq b_j\}$, Markov's inequality implies that $Pr[\mathbf{I}_{c,\ell}] \leq \mathbb{E}[U_c^j]/b_j \leq \frac{1}{2\alpha}$. □

Again using the trivial union bound, the probability that a particular column c is safe when considered under π is at least $1 - \frac{k}{2\alpha}$, and thus the probability of actually probing c is at least $\frac{y_c}{\alpha}(1 - \frac{k}{2\alpha})$. Finally, by linearity of expectations (since the instantiation of item c is independent of the event that it is probed) the expected profit is at least $\frac{1}{\alpha}(1 - \frac{k}{2\alpha}) \cdot \sum_{c=1}^{n} w_c \cdot y_c$. Setting $\alpha = k$ implies an expected profit of at least $\frac{1}{2k} \cdot \sum_c w_c y_c$, which proves the first part of Theorem 9.1.

9.2.1 Special Case: Monotone Column Outcomes

We now consider a special case of stochastic k-set packing where the outcomes of each column e form a total order w.r.t. the vector dominance relation; ie. for any column $i \in [n]$ and outcomes $a, b \in \{0,1\}^d$ for column i, either $a \leq b$ or $b \leq a$ coordinate-wise. Observe that this is true for the stochastic matching problem. The algorithm for monotone column outcomes is identical to the one for the general case when we set parameter $\alpha = 1$. We show below that this algorithm achieves a $k+1$ approximation; this bound nearly matches the LP integrality gap of $k - 1 + \frac{1}{k}$ for even deterministic k-set packing [72].

As above, consider the indicator random variables $\{\mathbf{I}_{c,\ell}\}_{\ell=1}^{k}$ for each column

$c \in [n]$. The improvement for the monotone-outcome case comes from the following strengthened bound on $\Pr[\bigwedge_\ell(\neg \mathbf{I}_{c,\ell})]$ which is obtained via the FKG inequality ([10, Theorem 6.2.1]). Given a vector $X = \{X_1, \ldots, X_n\}$ of independent events and an event F which is a function of X, we say F is an *increasing* (decreasing) event if for any vector X that $F(X)$ holds, $F(Y)$ also holds when $Y_i \geq X_i \ \forall i$ ($X_i \geq Y_i \ \forall i$). The FKG inequality says that for any collection of increasing (decreasing) events $F_1, \ldots, F_k$, it holds that $\Pr[\bigwedge_{i=1}^k F_i] \geq \prod_{i=1}^k \Pr[F_i]$.

Lemma 15. *For any column $c \in [n]$, $\Pr[\bigwedge_\ell(\neg \mathbf{I}_{c,\ell})] \geq \frac{1}{k+1}$.*

Proof: We can assume that the random permutation π is chosen by the following random experiment: For each column e, we pick independently and uniformly at random a real number $a_e \in [0,1]$. The columns are then sorted in increasing order of these numbers to obtain π.

We first condition on $a_c = x$, and bound $\Pr[\bigwedge_\ell(\neg \mathbf{I}_{c,\ell}) | a_c = x]$. For each column $e \in [n] \setminus \{c\}$, let the random variable $B_e = 1$ if $a_e \leq x$ and $B_e = 0$ otherwise. Let Z_e be the random variable corresponding to the random outcome of column e, with values consistent with the total-order of its outcomes. Let Y_e be the indicator random variable that is 1 w.p. y_e. Observe that random variables $\{B_e, Z_e, Y_e | e \in [n] \setminus \{c\}\}$ are mutually independent. Since the outcomes of each column e forms a total ordering, we can see that $\neg \mathbf{I}_{c,\ell}$ (for each $\ell \in [k]$) is a *decreasing function* of $\{B_e, Z_e, Y_e | e \in [n] \setminus \{c\}\}$. Therefore, by the FKG inequality, we have

$$\Pr\left[\bigwedge_\ell (\neg \mathbf{I}_{c,\ell}) \mid a_c = x\right] \geq \prod_\ell \Pr[(\neg \mathbf{I}_{c,\ell}) \mid a_c = x] \tag{9.3}$$

Claim 2. *For any column $c \in [n]$ and index $\ell \in [k]$, $\Pr[\mathbf{I}_{c,\ell} \mid a_c = x] \leq x$.*

Proof: Let $j \in [d]$ be the ℓ^{th} constraint in the support of c. Let U_c^j denote the

usage of constraint j, when column c is considered (according to π). Then,

$$\begin{aligned}\mathbb{E}[U_c^j \mid a_c = x] &= \sum_{e=1}^{n} \Pr[\, a_e < x \text{ AND } e \text{ is probed}\,] \cdot \mu_e(j), \\ &= \sum_{e=1}^{n} \Pr[\, a_e < x\,] \cdot y_e \cdot \mu_e(j), \\ &= \sum_{e=1}^{n} x \cdot y_e \cdot \mu_e(j), \leq x \cdot b_j.\end{aligned}$$

Since $\mathbf{I}_{c,\ell} = \{U_c^j \geq b_j\}$, Markov's inequality implies that $\Pr[\mathbf{I}_{c,\ell} \mid a_c = x] \leq \frac{\mathbb{E}[U_c^j | a_c = x]}{b_j} \leq x$. □

Now using Inequality (9.3) and Claim 2, we have that

$$\begin{aligned}\Pr\left[\bigwedge_\ell (\neg \mathbf{I}_{c,\ell})\right] &= \int_0^1 \Pr\left[\bigwedge_\ell (\neg \mathbf{I}_{c,\ell}) \mid a_c = x\right] \mathrm{d}x \geq \int_0^1 \prod_\ell \Pr[(\neg \mathbf{I}_{c,\ell}) \mid a_c = x] \mathrm{d}x \\ &\geq \int_0^1 \prod_\ell (1-x) \mathrm{d}x \geq \int_0^1 (1-x)^k \mathrm{d}x = \frac{1}{1+k}\end{aligned}$$

This completes the proof of Lemma 15. □

Now, the probability of actually probing column c is at least $y_c \cdot \Pr[\wedge_\ell (\neg \mathbf{I}_{c,\ell})] \geq \frac{y_c}{k+1}$. Finally, by linearity of expectations (since the instantiation of item c is independent of the event that it is probed) the expected profit is at least $\frac{1}{k+1} \cdot \sum_{c=1}^{n} w_c \cdot y_c$. This proves the second part of Theorem 9.1.

9.2.2 Safe versus Unsafe policies

Here we show that our algorithm's policy (which is safe) achieves a good approximation even relative to the optimal unsafe policy. Recall that an item can be probed in a *safe* policy only if there is zero probability of violating any capacity constraint. Whereas an *unsafe* policy may probe an item even if there is positive probability of violating capacity—but if capacity is violated then no profit is re-

ceived from that item and the policy ends. For a given set of items (with their distributions) and capacity vector $b' \in \mathbb{Z}_+^d$, let $\mathsf{Safe}(b')$ (resp. $\mathsf{Unsafe}(b')$) denote the value of the optimal safe (resp. unsafe) policy with capacity b'; $\mathsf{LP}(b')$ the optimal value of (LP1) with right hand side in (9.1) being b'; and $\mathsf{ALG}(b')$ the value obtained by our algorithm. Let $b \in \mathbb{Z}_+^d$ denote the capacity vector for the given instance; i.e. $b_j \geq 1$ for all $j \in [d]$ (if b_j was allowed to be 0 then clearly any safe policy gets zero value from items participating in this constraint j, but an unsafe policy can get positive value). We have:

$$\mathsf{Unsafe}(b) \leq \mathsf{Safe}(b+\mathbf{1}) \leq \mathsf{LP}(b+\mathbf{1}) \leq 2 \cdot \mathsf{LP}\left(\left\lceil \frac{b+\mathbf{1}}{2} \right\rceil\right) \leq 2 \cdot \mathsf{LP}(b)$$

where $\mathbf{1}$ is the all-ones vector. The first inequality uses the fact that each size lies in $\{0,1\}$, the second is by Claim 1, the third is by scaling (since $\frac{b+\mathbf{1}}{2} \leq \lceil \frac{b+\mathbf{1}}{2} \rceil$), and the last inequality uses $b \in \mathbb{Z}_+^d$. Finally, the analysis in the previous subsections implies that $\mathsf{ALG}(b) \geq \frac{1}{2k} \cdot \mathsf{LP}(b)$ in general; and $\mathsf{ALG}(b) \geq \frac{1}{k+1} \cdot \mathsf{LP}(b)$ in case of monotone column-outcomes. Combined with the above inequality we have $\mathsf{ALG}(b) \geq \frac{1}{4k} \cdot \mathsf{Unsafe}(b)$, and $\mathsf{ALG}(b) \geq \frac{1}{2k+2} \cdot \mathsf{Unsafe}(b)$ in the monotone column-outcomes case.

9.3 Stochastic Matching

We consider the following stochastic matching problem. The input is an undirected graph $G = (V, E)$ with a weight w_e and a probability value p_e on each edge $e \in E$. In addition, there is an integer value t_v for each vertex $v \in V$ (called *patience parameter*). Initially, each vertex $v \in V$ has patience t_v. At each step in the algorithm, any edge $e(u, v)$ such that u and v have positive remaining patience can be probed. Upon probing edge e, one of the following happens: (1) with probability p_e, vertices u and v get *matched* and are removed from the graph (along with all adjacent edges), or (2) with probability $1 - p_e$, the edge e is removed and the remaining patience numbers of u and v get reduced by 1. An algorithm is an

adaptive strategy for probing edges: its performance is measured by the expected weight of matched edges. The *unweighted* stochastic matching problem is the special case when all edge-weights are uniform.

Consider the following linear program: as usual, for any vertex $v \in V$, $\partial(v)$ denotes the edges incident to v. Variable y_e denotes the probability that edge $e = (u, v)$ gets probed in the adaptive strategy, and $x_e = p_e \cdot y_e$ denotes the probability that u and v get matched in the strategy. (This LP is similar to the LP used for general stochastic packing problems by Dean, Goemans and Vondrák [58].)

$$\text{maximize} \sum_{e \in E} w_e \cdot x_e \tag{LP2}$$

subject to

$$\sum_{e \in \partial(v)} x_e \leq 1 \qquad \forall v \in V \tag{9.4}$$

$$\sum_{e \in \partial(v)} y_e \leq t_i \qquad \forall v \in V \tag{9.5}$$

$$x_e = p_e \cdot y_e \qquad \forall e \in E \tag{9.6}$$

$$0 \leq y_e \leq 1 \qquad \forall e \in E \tag{9.7}$$

9.3.1 Weighted Stochastic Matching: Bipartite Graphs

In this section, we consider the stochastic matching problem on bipartite graphs. In fact, the algorithm produces a *matching-probing strategy* whose expected value is a constant fraction of the optimal value of (LP2) (which was for edge-probing).

Algorithm. First, we find an optimal fractional solution (x, y) to (LP2) and round y to identify a set of interesting edges $\widehat{E}$. Then we use König's Theorem [158, Ch. 20] to partition $\widehat{E}$ into a small collection of matchings $M_1, \ldots, M_h$. Finally, these matchings are then probed in random order. If we are only interested in edge-

probing strategies, probing the edges in $\widehat{E}$ in random order would suffice. We will refer to this algorithm as round-color-probe:

1. $(x, y) \leftarrow$ optimal solution to (LP2)
2. $\widehat{y} \leftarrow$ round y to an integral solution using GKSP
3. $\widehat{E} \leftarrow \{e \in E : \widehat{y}_e = 1\}$
4. $M_1, \ldots, M_h \leftarrow$ optimal edge coloring of $\widehat{E}$
5. For each M in $\{M_1, \ldots, M_h\}$ in random order, do:
    a. probe $\{(u, v) \in M : u$ and v are unmatched$\}$

The algorithm above uses the GKSP procedure of Gandhi et al. [73], which we describe next.

The GKSP algorithm. We state some properties of the dependent rounding framework of Gandhi et al. [73] that are relevant in our context.

Theorem 9.5 ([73]). *Let $(A, B; E)$ be a bipartite graph and $z_e \in [0, 1]$ be fractional values for each edge $e \in E$. The GKSP algorithm is a polynomial-time randomized procedure that outputs values $Z_e \in \{0, 1\}$ for each $e \in E$ such that the following properties hold:*

P1. Marginal distribution. For every edge e, $\Pr[Z_e = 1] = z_e$.

P2. Degree preservation. For every vertex $u \in A \cup B$, $\sum_{e \in \partial(u)} Z_e \leq \left\lceil \sum_{e \in \partial u} z_e \right\rceil$.

P3. Negative correlation. For any vertex u and any set of edges $S \subseteq \partial(u)$:

$$\Pr[\bigwedge_{e \in S} (Z_e = 1)] \leq \prod_{e \in S} \Pr[Z_e = 1].$$

We note that the GKSP algorithm in fact guarantees stronger properties than the ones stated above. For the purpose of analyzing round-color-probe, however, the properties stated above will suffice.

9.3.1.0.1 Feasibility. Let us first argue that our algorithm outputs a feasible strategy. If we care about feasibility in the edge-probing model, we only need to show that each vertex u is not probed more than t_u times. The following lemma shows that:

Lemma 16. *For every vertex u,* ROUND-COLOR-PROBE *probes at most t_u edges incident on u.*

Proof: Vertex u is matched in $\left|\left\{e \in \partial_{\widehat{E}}(u)\right\}\right|$ matchings. This is an upper bound on the number of times edges incident on u probed. Hence we just need to show that this quantity is at most t_u. Indeed,

$$\left|\left\{e \in \partial_{\widehat{E}}(u)\right\}\right| = \sum_{e \in \partial(u)} \hat{y}_e \leq \left\lceil \sum_{e \in \partial(u)} y_e \right\rceil \leq t_u,$$

where the first inequality follows from the degree preservation property of Theorem 9.5 and the second inequality from the fact that y is a feasible solution to (LP2). □

Let us argue that the strategy is also feasible under the matching-probing model. Recall that in the latter model we are given an additional parameter k (which without loss of generality we can assume to be at most $\max_{v \in V} t_u$) and we can probe edges in k round, with each round forming a matching. Let $\widehat{E}$ be the set of edges in the support of $\widehat{y}$, i.e., $\widehat{E} = \{e \in E \mid \widehat{y}_e = 1\}$. Let $h = \max_{v \in V} \deg_{\widehat{E}}(v) \leq \max_{v \in V} t_v$. König's Theorem allows us to decomposed $\widehat{E}$ into h matchings. Therefore, the probing strategy devised by the algorithm is also feasible in the matching-probing model.

Performance guarantee. Let us focus our attention on some edge $e = (u, v) \in E$. Our goal is to show that there is good chance that the algorithm will indeed probe e. We first analyze the probability of e being probed conditioned on $\widehat{E}$. Notice that the algorithm will probe e if and only if all previous probes incident on u and

v were unsuccessful; otherwise, if there was a successful probe incident on u or v, we say that e was *blocked*.

Let π be a permutation of the matchings $M_1, \ldots, M_h$. We extend this ordering to the set $\widehat{E}$ by listing the edges within a matching in some arbitrary but fixed order. Let us denote by $B(e, \pi) \subseteq \widehat{E}$ the set of edges incident on u or v that appear before e in π. It is not hard to see that

$$\Pr[\, e \text{ was not blocked} \mid \widehat{E}\,] \geq \mathbb{E}_\pi\Big[\prod_{f \in B(e,\pi)} (1 - p_f) \mid \widehat{E}\Big]; \tag{9.8}$$

here we assume that $\prod_{f \in B(e,\pi)}(1 - p_f) = 1$ when $B(e, \pi) = \emptyset$.

Notice that in (9.8) we only care about the order of edges incident on u and v. Furthermore, the expectation does not range over all possible orderings of these edges, but only those that are consistent with some matching permutation. We call this type of restricted ordering *random matching ordering* and we denote it by π; similarly, we call an unrestricted ordering *random edge ordering* and we denote it by σ. Our plan is to study first the expectation in (9.8) over random edge orderings and then to show that the expectation can only increase when restricted to range over random matching orderings.

The following simple lemma is useful in several places.

Lemma 17. *Let r and p_{max} be positive real values. Consider the problem of minimizing $\prod_{i=1}^{t}(1 - p_i)$ subject to the constraints $\sum_{i=1}^{t} p_i \leq r$ and $0 \leq p_i \leq p_{\max}$ for $i = 1, \ldots, t$. Denote the minimum value by $\eta(r, p_{\max})$. Then,*

$$\eta(r, p_{\max}) = (1 - p_{\max})^{\lfloor \frac{r}{p_{\max}} \rfloor} \left(1 - (r - \left\lfloor \frac{r}{p_{\max}} \right\rfloor p_{\max})\right) \geq (1 - p_{\max})^{r/p_{\max}}.$$

Proof: Suppose the contrary that the quantity is minimized but there are two p_is that are strictly between 0 and $p_{\max}$. W.l.o.g, they are p_1, p_2 and $p_1 > p_2$ Let

$\epsilon = \min(p_{\max} - p_1, p_2)$. It is easy to see that

$$(1-(p_1+\epsilon))(1-(p_2-\epsilon))\prod_{i=3}^{t}(1-p_i) - \prod_{i=1}^{t}(1-p_i) = \epsilon(p_2-p_1-\epsilon)\prod_{i=3}^{t}(1-p_i) < 0.$$

This contradicts the fact the quantity is minimized. Hence, there is at most one p_i which is strictly between 0 and $p_{\max}$.

The last inequality holds since $1-b \geq (1-a)^{b/a}$ for any $0 \leq b \leq a \leq 1$. □

Let $\partial_{\widehat{E}}(e)$ be the set of edges in $\widehat{E}$ incident on either endpoint of e excluding e itself.

Lemma 18. *Let e be an edge in $\widehat{E}$ and let σ be a random edge ordering. Let $p_{\max} = \max_{f\in\widehat{E}} p_f$. Assume that $\sum_{f\in\partial_{\widehat{E}}(e)} p_f = r$. Then,*

$$\mathbb{E}_\sigma\Big[\prod_{f\in B(e,\sigma)}(1-p_f) \mid \widehat{E}\Big] \geq \int_0^1 \eta(xr, xp_{\max})\mathrm{d}x.$$

Proof: We claim that the expectation can be written in the following continuous form:

$$\mathbb{E}_\sigma\Big[\prod_{f\in B(e,\sigma)}(1-p_f) \mid \widehat{E}\Big] = \int_0^1 \prod_{f\in\partial_{\widehat{E}}(e)}(1-xp_f)\mathrm{d}x.$$

The lemma easily follows from this and Lemma 17.

To see the claim, we consider the following random experiment: For each edge $f \in \partial(e)$, we pick uniformly at random a real number a_f in $[0,1]$. The edges are then sorted according to these numbers. It is not difficult to see that the experiment produces uniformly random orderings. For each edge f, let the random variable

$A_f = 1 - p_f$ if $f \in B(e,\sigma)$ and $A_f = 1$ otherwise. Hence, we have

$$\begin{aligned}
\mathbb{E}_\sigma\Big[\prod_{f\in B(e,\sigma)}(1-p_f) \mid \widehat{E}\Big] &= \int_0^1 \mathbb{E}\Big[\prod_{f\in\partial_{\widehat{E}}(e)} A_f \mid a_e = x\Big]\mathrm{d}x \\
&= \int_0^1 \prod_{f\in\partial_{\widehat{E}}(e)} \mathbb{E}\Big[A_f \mid a_e = x\Big]\mathrm{d}x \\
&= \int_0^1 \prod_{f\in\partial_{\widehat{E}}(e)} \big(x(1-p_f)+(1-x)\big)\mathrm{d}x \\
&= \int_0^1 \prod_{f\in\partial_{\widehat{E}}(e)} (1-xp_f)\mathrm{d}x
\end{aligned}$$

The second equality holds since the A_f variables, conditional on $a_e = x$, are independent. □

Lemma 19. *Let $\rho(r, p_{\max}) = \int_0^1 \eta(xr, xp_{\max})\mathrm{d}x$. For any $r, p_{\max} > 0$, we have*

1. *$\rho\,(r, p_{\max})$ is convex and decreasing on r.*
2. *$\rho\,(r, p_{\max}) \geq \frac{1}{r+p_{\max}} \cdot \Big(1 - (1-p_{\max})^{1+\frac{r}{p_{\max}}}\Big) > \frac{1}{r+p_{\max}} \cdot \Big(1 - e^{-r}\Big)$*

Proof: To see the first part, let us consider the function values on discrete points $r = p_{\max}, 2p_{\max}, \ldots$. Let $F(x) = \frac{1}{x}(1-c^x)$ where $c = 1 - p_{\max}$. From Lemma 17, we can easily get that for integral t,

$$\rho\,(tp_{\max}, p_{\max}) = \int_0^1 (1-xp_{\max})^t \mathrm{d}x = \frac{1}{p_{\max}(t+1)}\left(1-c^{t+1}\right) = \frac{1}{p_{\max}}F(t+1).$$

The function $F(x)$ is a convex function for any $0 < c < 1$. Indeed, it is not hard to prove that $\frac{\mathrm{d}^2}{\mathrm{d}x^2}F(x) = \frac{2}{x^3} + c^x\left(-\frac{2}{x^3} + \frac{2\ln a}{x^2} - \frac{\ln^2 a}{x}\right) > 0$ for any $0 < c < 1$. However, $\rho\,(tp_{\max}, p_{\max})$ only coincides with $\frac{1}{p_{\max}}F(t+1)$ at integral values of t. Now, let us consider the value of $\rho(r, p_{\max})$ for $\gamma p_{\max} < r < (\gamma+1)p_{\max}$ (for some integer

$\gamma \geq 0$):

$$\rho\left(r, p_{\max}\right) = \int_0^1 (1 - x p_{\max})^{\gamma} \Big(1 - x(r - \gamma p_{\max})\Big) \mathrm{d}x \tag{9.9}$$

The key observation is that for fixed values of $p_{\max}$ and γ the right hand side of (9.9) is a just linear function of r. The dependency of ρ in terms of r then becomes clear: it is a piecewise linear function that takes the value $F(t+1)/p_{\max}$ at abscissa points $tp_{\max}$ for $t \in \mathbb{Z}_{\geq 0}$. Therefore, ρ is a convex decreasing function of r.

The second part follows easily from Lemma 17:

$$\begin{aligned}
\rho\left(r, p_{\max}\right) &= \int_0^1 \eta(xr, xp_{\max})\mathrm{d}x \geq \int_0^1 (1 - x p_{\max})^{r/p_{\max}} \mathrm{d}x \\
&= \frac{1}{r + p_{\max}} \cdot \Big(1 - (1 - p_{\max})^{1 + \frac{r}{p_{\max}}}\Big) \\
&\geq \frac{1}{r + p_{\max}} \cdot \Big(1 - e^{-r}\Big)
\end{aligned}$$

□

Lemma 20. *Let $e = (u, v) \in \widehat{E}$. Let π be a random matching ordering and σ be a random edge ordering of the edges adjacent to u and v. Then*

$$\mathbb{E}_{\pi}\Big[\prod_{f \in B(e,\pi)} (1 - p_f) \mid \widehat{E}\Big] \geq \mathbb{E}_{\sigma}\Big[\prod_{f \in B(e,\sigma)} (1 - p_f) \mid \widehat{E}\Big].$$

Proof: We can think of π as a permutation of bundles of edges: For each matching, if there are two edges incident on e, we bundle the edges together; if there is a single edge incident on e this edge is in a singleton bundle by itself. The random edge ordering σ can be thought as having all edges incident on e in singleton bundles.

Consider the same random experiment as in Lemma 18 except that we only pick one random number for each bundle. Let $G(e)$ be the set of all bundles incident

on e. Using the same argument as in Lemma 18, we have

$$\mathbb{E}_\pi\Big[\prod_{f\in B(e,\pi)}(1-p_f)\ \mid\ \widehat{E}\Big] = \int_0^1 \prod_{g\in G(e)}\Big(x\cdot\prod_{f\in g}(1-p_f)+(1-x)\Big)\mathrm{d}x.$$

But for any bundle $g \in G(e)$ and $0 \le x \le 1$, we claim that

$$x\cdot\prod_{f\in g}(1-p_f)+(1-x) \ge \prod_{f\in g}(1-xp_f).$$

For singleton bundles we actually have equality. For a bundle $g = \{f_1, f_2\}$, we have

$$\begin{aligned} x(1-p_{f_1})(1-p_{f_2})+(1-x) &= 1-xp_{f_1}-xp_{f_2}+xp_{f_1}p_{f_2} \\ &\ge 1-xp_{f_1}-xp_{f_2}+x^2p_{f_1}p_{f_2} \\ &= (1-xp_{f_1})(1-xp_{f_2}). \end{aligned}$$

This completes the proof. □

As we shall see shortly, if $\sum_{f\in\partial_{\widehat{E}}(e)} p_e$ is small then the probability that e is not blocked is large. Because of the marginal distribution property of the GKSP rounding procedure, we can argue that this quantity is small *in expectation* since $\sum_{f\in\partial(e)} p_e y_e \le 2$ due to the fact that y is a feasible solution to (LP2). This, however, is not enough; in fact, for our analysis to go through, we need a slightly stronger property.

Lemma 21. *For every edge e,*

$$\mathbb{E}\Big[\sum_{f\in\partial_{\widehat{E}}(e)} p_f\ \mid\ e\in\widehat{E}\Big] \le \sum_{f\in\partial(e)} p_f\, y_f.$$

Proof: Let u be an endpoint of e.

$$\begin{aligned}
\mathbb{E}\Big[\sum_{f\in\partial_{\widehat{E}}(u)-e} p_f \;\mid\; e\in\widehat{E}\Big] &= \sum_{f\in\partial(u)-e} \Pr[\,\widehat{y}_f = 1 \mid \widehat{y}_e = 1]\cdot p_f,\\
&\le \sum_{f\in\partial(u)-e} \Pr[\,\widehat{y}_f = 1\,]\cdot p_f, && \text{[by Theorem 9.5 P3]}\\
&= \sum_{f\in\partial(u)-e} y_f\, p_f. && \text{[by Theorem 9.5 P1].}
\end{aligned}$$

The same bound holds for the other endpoint of e. Adding the two inequalities we get the lemma. □

Everything is in place to derive a bound the expected weight of the matching found by our algorithm.

Theorem 9.6. *If G is bipartite then* ROUND-COLOR-PROBE *is a $1/\rho(2, p_{\max})$ approximation under the edge- and matching-probing model, where ρ is defined in Lemma 19. The worst ratio is attained at $p_{\max} = 1$, where it is 3. The ratio tends to $\frac{2}{1-e^{-2}}$ as $p_{\max}$ tends to 0.*

Proof: Recall that the optimal value of (LP2) is exactly $\sum_{e\in E} w_e y_e x_e$. The ex-

pected weight of the matching found by the algorithm is

$$\begin{aligned}
\mathbb{E}[\,\mathsf{ALG}\,] &= \sum_{e\in E} w_e\, p_e\, \Pr[\, e \in \widehat{E}\,] \cdot \Pr[\, e \text{ was not blocked } \mid e \in \widehat{E}\,] \\
&= \sum_{e\in E} w_e\, p_e\, y_e \cdot \Pr[\, e \text{ was not blocked } \mid e \in \widehat{E}\,] && \text{[by Theorem 9.5 P1]} \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e \cdot \mathbb{E}_\pi\Big[\prod_{f\in B(e,\pi)} (1-p_f) \mid e \in \widehat{E}\Big] && \text{[by (9.8)]} \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e \cdot \mathbb{E}_\sigma\Big[\prod_{f\in B(e,\sigma)} (1-p_f) \mid e \in \widehat{E}\Big] && \text{[by Lemma 20]} \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e \cdot \mathbb{E}\Big[\rho\Big(\sum_{f\in \partial_{\widehat{E}}(e)} p_f, p_{\max}\Big) \mid e \in \widehat{E}\Big] && \text{[by Lemma 18]} \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e \cdot \rho\Big(\mathbb{E}\Big[\sum_{f\in \partial_{\widehat{E}}(e)} p_f \mid e \in \widehat{E}\Big], p_{\max}\Big) && \text{[by Jensen's inequality]} \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e \cdot \rho\Big(\sum_{f\in \partial(e)} y_f\, p_f, p_{\max}\Big) && \text{[by Lemma 23]} \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e \cdot \rho(2, p_{\max}) && [y \text{ is feasible for (LP2)}].
\end{aligned}$$

Notice that we are able to use Jensen's inequality because, as shown in Lemma 19, $\rho(r, p_{\max})$ is a convex and decreasing function of r. The last two inequalities also use the fact that ρ is decreasing.

It can be checked directly that $\rho(2, p_{max})$ is minimized at $p_{max} = 1$ where it is $1/3$. Moreover $\rho(2, p_{max}) \to \frac{2}{1-e^{-2}}$ as p_{max} tends to 0. □

This proves the second parts of Theorem 9.2 and Theorem 9.3.

9.3.2 Weighted Stochastic Matching: General Graphs

We now present an algorithm for weighted stochastic matching in general graphs that builds on the algorithm for the bipartite case. The basic idea is to solve (LP2), randomly partition the vertices of G into two sets A and B, and then run ROUND-COLOR-PROBE on the bipartite graph induced by (A, B). For the analysis

to go through, it is crucial that we use the already computed fractional solution instead of solving again (LP2) for the new bipartite graph in the call to round-color-probe.

1. $(x, y) \leftarrow$ optimal solution to (LP2)
2. randomly partition vertices into A and B
3. run round-color-probe on the bipartite graph and the fractional solution induced by (A, B)

Theorem 9.7. *For general graphs there is a $2/\rho(1, p_{\max})$ approximation under the edge- and matching-probing model, where ρ is defined in Lemma 19. The worst ratio is attained at $p_{\max} = 1$, where it is 4. The ratio tends to $\frac{2}{1-e^{-1}}$ as $p_{\max}$ tends to 0.*

Proof: The analysis is very similar to the bipartite case. Essentially, conditional on a particular outcome for the partition (A, B), all the lemmas derived in the previous section hold. In other words, the same derivation done in the proof of Theorem 9.6 yields:

$$\mathbb{E}[\,\mathsf{ALG} \mid (A,B)\,] \geq \sum_{e\in(A,B)} w_e p_e y_e \cdot \rho\Big(\sum_{f\in\partial_{A,B}(e)} p_f\, y_f, p_{\max}\Big),$$

where $\partial_{A,B}(e) = \partial(e) \cap (A, B)$.

Hence, the expectation of algorithm's performance is:

$$\begin{aligned}
\mathbb{E}[\,\mathsf{ALG}\,] &\geq \sum_{e\in E} w_e\, p_e\, y_e\ \Pr[e \in (A,B)] \cdot \mathbb{E}\Big[\rho\Big(\sum_{f\in\partial_{A,B}(e)} p_f\, y_f,\ p_{\max}\Big) \mid e \in (A,B)\Big], \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e\, \frac{1}{2} \cdot \rho\Big(\mathbb{E}\Big[\sum_{f\in\partial_{A,B}(e)} p_f\, y_f \mid e \in (A,B)\Big],\ p_{\max}\Big), \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e\, \frac{1}{2} \cdot \rho\Big(\sum_{f\in\partial(e)} \frac{p_f\, y_f}{2},\ p_{\max}\Big), \\
&\geq \sum_{e\in E} w_e\, p_e\, y_e\, \frac{1}{2} \cdot \rho\big(1,\ p_{\max}\big),
\end{aligned}$$

where the second inequality follows from Jensen's inequality and the fact that $\rho(r, p_{\max})$ is a convex decreasing function of r. Finally, noting that $\sum_{e \in E} w_e \, p_e \, y_e$ is a lower bound on the value of the optimal strategy, the theorem follows. □

This proves the first parts of Theorem 9.2 and Theorem 9.3.

9.4 Stochastic *Online* Matching with Timeouts

As mentioned in the introduction, the stochastic online matching with timeouts is best imagined as selling a finite set of goods to buyers that arrive over time. The input to the problem consists of a bipartite graph $G = (A, B, A \times B)$, where A is the set of *items* that the seller has to offer, with exactly one copy of each item, and B is a set of *buyer types/profiles*. For each buyer type $b \in B$ and item $a \in A$, p_{ab} denotes the probability that a buyer of type b will like item a, and w_{ab} denotes the revenue obtained if item a is sold to a buyer of type b. Each buyer of type $b \in B$ also has a patience parameter $t_b \in \mathbb{Z}_+$. There are n buyers arriving online, with $e_b \in \mathbb{Z}$ denoting the expected number of buyers of type b, with $\sum e_b = n$. Let $\mathcal{D}$ denote the induced probability distribution on B by defining $\Pr_{\mathcal{D}}[b] = e_b/n$. All the above information is given as input.

The stochastic online model is the following: At each point in time, a buyer arrives, where her type $\mathbf{b} \in_{\mathcal{D}} B$ is an i.i.d. draw from $\mathcal{D}$. The algorithm now shows her *up to* $t_{\mathbf{b}}$ *distinct items one-by-one*: the buyer likes each item $a \in A$ shown to her independently with probability p_{ab}. The buyer purchases the first item that she is *offered and likes*; if she buys item a, the revenue accrued is w_{ab}. If she does not like any of the items shown, she leaves without buying. The objective is to maximize the expected revenue.

We get the stochastic online matching problem of Feldman et al. [68] if we have $w_{ab} = p_{ab} \in \{0, 1\}$, in which case we need only consider $t_b = 1$. Their focus was on beating the $1 - 1/e$-competitiveness known for worst-case models [114, 109, 133, 32, 80]; they gave a 0.67-competitive algorithm that works for the unweighted case

with high probability. On the other hand, our results are for the weighted case (with preference-uncertainty and timeouts), but only in expectation. Furthermore, in our extension, due to the presence of timeouts (see Section **??**), any algorithm that provides a guarantee whp must necessarily have a high competitive ratio.

By making copies of buyer types, we may assume that $e_b = 1$ *for all* $b \in B$*, and* $\mathcal{D}$ *is uniform over* B*.* For a particular run of the algorithm, let $\hat{B}$ denote the actual set of buyers that arrive during that run. Let $\hat{G} = (A, \hat{B}, A \times \hat{B})$, where for each $a \in A$ and $\hat{b} \in \hat{B}$ (and suppose its type is some $b \in B$), the probability associated with edge $(a, \hat{b})$ is p_{ab} and its weight is w_{ab}. Moreover, for each $\hat{b} \in \hat{B}$ (with type, say, $b \in B$), set its patience parameter to $t_{\hat{b}} = t_b$. We will call this the *instance graph*; the algorithm sees the vertices of $\hat{B}$ in random order, and has to adaptively find a large matching in $\hat{G}$.

It now seems reasonable that the algorithm of Section 9.2 (specialized to stochastic matching) should work here. But the algorithm does not know $\hat{G}$ (the actual instantiation of the buyers) up front, it only knows G, and hence some more work is required to obtain an algorithm. Further, as was mentioned in the preliminaries, we use OPT to denote the optimal adaptive strategy (instead of the optimal offline matching in $\hat{G}$ as was done in [68]), and compare our algorithm's performance with this OPT.

The Linear Program. For a graph $H = (A, C, A \times C)$ with each edge (a, c) having a probability p_{ac} and weight w_{ac}, and vertices in C having patience parameters t_j, consider the LP(H):

$$\text{maximize} \sum_{a \in A,\, c \in C} w_{ac} \cdot x_{ac} \tag{LP3}$$

subject to

$$\sum_{c \in C} x_{ac} \leq 1 \qquad \forall a \in A \tag{9.10}$$

$$\sum_{a \in A} x_{ac} \leq 1 \qquad \forall c \in C \tag{9.11}$$

$$\sum_{a \in A} y_{ac} \leq t_c \qquad \forall c \in C \tag{9.12}$$

$$x_{ac} = p_{ac} \cdot y_{ac} \qquad \forall a \in A,\, c \in C \tag{9.13}$$

$$y_{ac} \in [0, 1] \qquad \forall a \in A,\, c \in C \tag{9.14}$$

Note that this LP is very similar to the one in Section 9.3, but the vertices in A do not have timeout values. Let $\mathsf{LP}(H)$ denote the optimal value of this LP.

The algorithm:

1. Before any buyers arrive, solve the LP on the expected graph G to get values y.
2. $\widehat{y} \leftarrow$ round y to an integral solution using GKSP
3. $\widehat{E} \leftarrow \{e \in E : \widehat{y}_e = 1\}$
4. When any buyer $\hat{b}$ (of type b) arrives online:
   a. If $\hat{b}$ is the first buyer of type b, consider the set of items $\{a \in A \mid (a, b) \in \widehat{E}\}$ in a uniformly random order. One by one, offer each item a (that is still unsold) to $\hat{b}$; stop if either t_b offers are made or $\hat{b}$ purchases any item.
   b. If $\hat{b}$ is not the first arrival of type b, do not offer any items to $\hat{b}$.

In the following, we prove that our algorithm achieves a constant approximation to stochastic online matching with timeouts. The first lemma show that the expected value obtained by the best online adaptive algorithm is bounded above by $\mathbb{E}[\mathsf{LP}(\hat{G})]$.

Lemma 22. *The optimal value* OPT *of the given instance is at most* $\mathbb{E}[\mathsf{LP}(\hat{G})]$*, where the expectation is over the random draws to create* $\hat{G}$*.*

Proof: Consider an algorithm that is allowed to see the instantiation $\hat{B}$ of the buyers before deciding on the selling strategy—the expected revenue of the best such algorithm is clearly an upper bound on OPT. Given any instantiation $\hat{B}$, the expected revenue of the optimal selling strategy is at most $\mathsf{LP}(\hat{G})$ (see e.g. Claim 1). The claim follows by taking an expectation over $\hat{B}$. □

For any buyer-type $b \in B$, in the following, $\hat{b}$ refers to the first type-b buyer (if any). For each $b \in B$, let random variable $T_b \in [n] \cup \{\infty\}$ denote the earliest arrival time of a type-b buyer; if there is no type-b arrival then $T_b = \infty$.

Let $\mathcal{A}_b \equiv (T_b < \infty)$ denote the event that there is some type-b arrival in the instantiation $\hat{B}$. Since each arrival is i.i.d. from the uniform distribution over B,

$$\Pr[\mathcal{A}_b] = 1 - (1 - \frac{1}{n})^n \geq 1 - \frac{1}{e}.$$

Recall $\partial_{\widehat{E}}(e)$ is the set of edges in $\widehat{E}$ incident on either endpoint of e.

Lemma 23. *For every edge* $e = (a, b)$*,*

$$\mathbb{E}\Big[\sum_{f \in \partial_{\widehat{E}}(e) \cap \hat{G}} p_f \;\mid\; \mathcal{A}_b, e \in \widehat{E} \Big] \leq \sum_{f \in \partial(b)} p_f \, y_f + (1 - \frac{1}{e}) \sum_{f \in \partial(a)} p_f \, y_f.$$

Proof:

$$\begin{aligned}
\mathbb{E}\Big[\sum_{f \in \hat{G} \cap \partial_{\widehat{E}}(a) - e} p_f \;\mid\; \mathcal{A}_b, e \in \widehat{E} \Big] &= \sum_{(a,u) \in \cap \partial(a) - e} \Pr[\, \widehat{y}_{au} = 1 \;\mid\; \mathcal{A}_b, \widehat{y}_e = 1] \cdot \Pr[\mathcal{A}_u] \cdot p_{au}, \\
&\leq (1 - \frac{1}{e}) \sum_{f \in \partial(a) - e} \Pr[\, \widehat{y}_f = 1 \,] \cdot p_f, \\
&= (1 - \frac{1}{e}) \sum_{f \in \partial(a) - e} y_f \, p_f.
\end{aligned}$$

The same bound holds for the other endpoint b except we do not have the extra factor $(1 - \frac{1}{e})$. Adding the two inequalities we get the lemma. □

Lemma 24. *Our expected revenue is at least* $\frac{2e^2-e-1}{6e^2} \cdot \mathsf{LP}(G)$.

Proof:

Note that our algorithm obtains positive revenue only for buyers $\{\hat{b} \mid b \in B, T_b < \infty\}$; let R_b denote the revenue obtained from buyer $\hat{b}$ (if any). The expected revenue of the algorithm is $\mathbb{E}[\sum_{b \in B} R_b]$. We now estimate $\mathbb{E}[R_b]$ for a fixed $b \in B$.

In the following, we condition on $\mathcal{A}_b$ and $(a, b) \in \widehat{E}$ and bound $\mathbb{E}[R_b \mid \widehat{E}, \mathcal{A}_b]$. Similar to the argument in Section 9.3.1, we can see that

$$\begin{aligned}\Pr[\,\text{item } a \text{ offered to } \hat{b} \mid \widehat{E}, \mathcal{A}_b\,] &= \Pr\,[\,e = (a, \hat{b}) \text{ was not blocked } \mid\ \widehat{E}, \mathcal{A}_b\,] \\ &\geq \mathbb{E}_\pi\Big[\prod_{f \in B(e,\pi)} (1 - p_f)\ \mid\ \widehat{E}, \mathcal{A}_b\Big];\end{aligned}$$

Therefore, we have that

$$\begin{aligned}\Pr[\,\text{item } a \text{ offered to } \hat{b} \mid \mathcal{A}_b\,] &= \Pr[(a, b) \in \widehat{E}] \cdot \mathbb{E}_\pi\Big[\prod_{f \in B(e,\pi)} (1 - p_f)\ \mid\ \widehat{E}, \mathcal{A}_b\Big]; \\ &\geq y_{a,b} \cdot \rho\Big(\sum_{f \in \partial(b)} p_f\, y_f + (1 - \frac{1}{e}) \sum_{f \in \partial(a)} p_f\, y_{f\cdot}, 1\Big) \\ &\geq y_{a,b} \cdot \rho(2 - \frac{1}{e}, 1) = \frac{2e+1}{6e} y_{a,b}.\end{aligned}$$

The first inequality follows from the same argument as in the proof of Theorem 9.6. This implies:

$$\mathbb{E}[R_b \mid \mathcal{A}_b] = \sum_{a \in A} w_{ab} \cdot p_{ab} \cdot \Pr[\,\text{item } a \text{ offered to } \hat{b} \mid \mathcal{A}_b\,] \geq \frac{2e+1}{6e} \sum_{a \in A} w_{ab} \cdot x_{ab}.$$

Since $\Pr[\,\mathcal{A}_b\,] \geq 1 - \frac{1}{e}$, we also have

$$\mathbb{E}[R_b] \geq \frac{e-1}{e} \cdot \frac{we+1}{6e} \sum_{a \in A} w_{ab} \cdot x_{ab} \geq \frac{2e^2 - e - 1}{6e^2} \sum_{a \in A} w_{ab} \cdot x_{ab}.$$

Finally, the expected revenue obtained by the algorithm is:

$$\sum_{b \in B} \mathbb{E}[R_b] \geq \frac{2e^2 - e - 1}{6e^2} \mathsf{LP}(G).$$

This proves Lemma 24. □

Note that we have shown that $\mathbb{E}[\mathsf{LP}(\hat{G})]$ is an upper bound on OPT, and that we can get a constant fraction of $\mathsf{LP}(G)$. The final lemma relates these two, namely the LP-value of the expected graph G (computed in Step 1) to the expected LP-value of the instantiation $\hat{G}$; the proof uses a simple but subtle duality-based argument.

Lemma 25. $\mathsf{LP}(G) \geq \mathbb{E}[\mathsf{LP}(\hat{G})]$.

Proof: Consider the dual of the linear program (LP3).

$$\min \sum_{a \in A} \alpha_a + \sum_{c \in C} (\alpha_c + t_c \cdot \beta_c) + \sum_{a \in A,\, c \in C} z_{ac} \tag{9.15}$$

$$z_{ac} + p_{ac} \cdot (\alpha_a + \alpha_c) + \beta_c \geq w_{ac} \cdot p_{ac} \qquad \forall a \in A,\, c \in C \tag{9.16}$$

$$\alpha, \beta, z \geq 0 \tag{9.17}$$

Let (α, β, z) denote the optimal dual solution corresponding to graph G; note that its objective value equals $\mathsf{LP}(G)$ by strong duality. For any instantiation $\hat{G}$, define dual solution $(\hat{\alpha}, \hat{\beta}, \hat{z})$ as follows:

1. For all $a \in A$, $\hat{\alpha}_a = \alpha_a$.
2. For each $c \in \hat{B}$ (of type b), $\hat{\alpha}_c = \alpha_b$ and $\hat{\beta}_c = \beta_b$.
3. For each $a \in A$ and $c \in \hat{B}$ (of type b), $\hat{z}_{ac} = z_{ab}$.

Note that $(\hat{\alpha}, \hat{\beta}, \hat{z})$ is a feasible dual solution corresponding to the LP on $\hat{G}$: there is constraint for each $a \in A$ and $c \in \hat{B}$, which reduces to a constraint for (α, β, z) in the dual corresponding to G. By weak duality, the objective value for $(\hat{\alpha}, \hat{\beta}, \hat{z})$ is an upper-bound on $\mathsf{LP}(\hat{G})$. For each $b \in B$, let N_b denote the number of type b buyers in the instantiation $\hat{B}$; note that $\mathbb{E}[N_b] = 1$ by definition of distribution $\mathcal{D}$. Then the dual objective for $(\hat{\alpha}, \hat{\beta}, \hat{z})$ satisfies:

$$\sum_{a \in A} \alpha_a + \sum_{b \in B} N_b \cdot (\alpha_b + t_b \cdot \beta_b) + \sum_{a \in A,\, b \in B} N_b \cdot z_{ab} \quad \geq \quad \mathsf{LP}(\hat{G}).$$

Taking an expectation over $\hat{B}$, we obtain:

$$\begin{aligned} \mathbb{E}[\mathsf{LP}(\hat{G})] &\leq \sum_{a \in A} \alpha_a + \sum_{b \in B} \mathbb{E}[N_b] \cdot \left(\alpha_b + t_b \cdot \beta_b + \sum_{a \in A} z_{ab} \right) \\ &= \sum_{a \in A} \alpha_a + \sum_{b \in B} (\alpha_b + t_b \cdot \beta_b) + \sum_{a \in A,\, b \in B} z_{ab} = \mathsf{LP}(G). \end{aligned}$$

This proves the lemma. □

Applying Lemmas 22, 24 and 25 completes Theorem 9.4's proof.

Chapter 10

Conclusion

Managing large-scale uncertain data and solving decision-making problems over them have become increasingly important in computer science and other disciplines. This is in part due to the rapid increase in the volume of uncertain data automatically generated by modern data gathering or integration systems. In this thesis, we studied three important problems in decision making under uncertainty: ranking under uncertainty, utility maximization under uncertainty, and matching under uncertainty.

In the first half of the thesis, we considered the problem of ranking and top-k query processing over probabilistic datasets. We observed that several prior ranking proposals, while all seem to be natural, behaved in drastically diverse, even conflicting manners. This observation led us to contend that a single, specific ranking function may not suffice for probabilistic datasets. Hence, instead of proposing yet another specific ranking function, we proposed using two parameterized ranking functions, called PRF^ω and PRF^e that allow the user to control their behavior by properly setting the parameters. We presented novel exact or approximate algorithms for computing PRF functions, even if the datasets exhibit complex correlations, modeled using probabilistic and/xor trees or Markov networks, or the probability distributions are continuous. Our algorithms match or improve the time complexities of several algorithms developed for previous ranking functions. We also developed an approach for approximating a ranking function using a linear combination of PRF^e functions thus enabling highly efficient, albeit approximate computation, and also for learning a ranking function from user preferences. Moreover, we proposed the notion of a consensus answer (CON) which,

roughly speaking, is a deterministic answer that is "closest in expectation" to the possible answers over a probabilistic database. Under this framework, we obtained optimal or approximation algorithms for computing the consensus top-k answers, under different distance metrics. We also showed a close relationship between PRF and the consensus top-k answer semantics.

The second set of problems we studied are the stochastic versions of a broad class of combinatorial optimization problems, including shortest paths, spanning trees, matchings and knapsacks. An instance of the problem consists of a set of ground elements (edges, items, etc.) and a feasible solution is a subset of the elements satisfying some property. The weight of each element is a random variable and the probability distribution is part of the input. We could formulate the problem as minimizing the expected total weight of the solution – this is perhaps the first problem formulation that comes to our mind. However, we observed that the expected value is inadequate in capturing different types of *risk-averse* or *risk-prone* behaviors. To resolve this issue, we adopted the expected utility theory and considered a more general objective which is to maximize the *expected utility* of the solution for some given utility function. We presented a polynomial time approximation algorithm with *additive error* ϵ for any $\epsilon > 0$, under certain conditions. Our result generalizes and improves several prior results on stochastic shortest path, stochastic spanning tree, and stochastic knapsack. A key ingredient in our algorithm is to the Fourier series based technique for decomposing the utility function into a short exponential sum, which may find other applications in stochastic optimization. Our technique works only in settings where the solution is a fixed set. It would be interesting to see if such technique can handle more general stochastic models, e.g., the non-adaptive (or adaptive) setting considered in Dean et al. [59].

The last part of the thesis is devoted to the stochastic matching problem, which is motivated by interesting applications in online dating, kidney exchange and online ad assignment. In this problem, we are given a probabilistic graph where each

possible edge is present independently with some probability. However, the presence of each edge can be only found out by probing the edge. The goal is to design a probing strategy to maximize the expected total weight of the matching. We obtained constant approximations for the weighted stochastic matching problems by using an LP rounding approach. This resolved the main open question from Chen et al. [40]. We also obtained constant approximations for the more restricted matching-probing model, significantly improving on the previous logarithmic approximation ratio by Chen et al. [40].

A common, sometimes confusing, yet very important issue in decision making under uncertainty is to choose the right problem formulation. Recall that in our first and second problem, we spent significant amount of space to motivate and justify our problem formulations. From illustrating diverse behavior of several prior ranking functions to criticizing expected values via the St. Petersberg paradox, all roads lead to Rome – we end up adopting some form of the expected utility theory (both PRF and CON can be cast into an expected utility maximization problem, while EUM is a direct adoption of the theory). The theory has been quite successful in economics and game theory. We expect that more decision making problems under uncertainty that araise from computer science, especially in stochastic combinatorial optimization and probabilistic databases, can be formulated and studied under the framework of the expected utility theory. Typically, optimizing the expected utility for general utility functions is a hard task since it generalizes the problem of optimizing the expected value and the overflow probability. For many problems with the new objective, much less is known and new computational techniques are required. We believe it is a fruitful direction for further research.

Appendix A

Expanding Polynomials

A.1 Expanding a Nested Formula

We consider the the general question how fast can we expand a nested expression of a uni-variable polynomial (with variable x) into the standard form $\sum c_i x^i$. Here a nested expression refers to a formula that only involves constants, the variable x, addition $+$, multiplication $\times$, and parenthesis (and), for example, $f(x) = ((1 + x + x^2)(x^2 + 2x^3) + x^3(2 + 3x^4))(1 + 2x)$. Formally, we define recursively an *expression* to be either (1) A constant or the variable x, or (2) The sum of two expressions, or (3) The product of two expressions.

We assume the degree of the polynomial and the length of the expression are of sizes $O(n)$. The naïve method runs in time $O(n^3)$ by expanding each subformula. If we use the divide-and-conquer method for expanding each subformula, we can easily achieve $O(n^2 \log^2 n)$. We omit the details. Now, we sketch two improved algorithms with running time $O(n^2)$.

Algorithms 1

1. Choose $n + 1$ different numbers $x_0, ..., x_n$.

2. Evaluate the polynomial at these points, i.e., compute $f(x_i)$. It is easy to see that each evaluation takes linear time (bottom-up over the tree). So this step takes $O(n^2)$ time in total.

3. Use any $O(n^2)$ polynomial interpolation algorithm to find the coefficient. In fact, the interpolation reduces to finding a solution for the following linear

system:

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

The commonly used Gaussian elimination for inverting a matrix requires $O(n^3)$ operations. The matrix we used is a special type of matrix and is commonly referred to as a Vandermonde matrix. There exists numerical algorithms that can invert a Vandermonde matrix in $O(n^2)$ time, for example [33].

A small drawback of the above algorithm is that the algorithms used to invert a Vandermonde matrix is nontrivial to implement. The next algorithm does not need to invert a matrix, is much simpler to implement and has the same running time of $O(n^2)$.

Algoirthm 2

Instead of picking arbitrary $n+1$ real points $x_0, \dots x_n$ to evaluate the polynomial, we pick $n+1$ complex points $1, u, u^2, \dots, u^n$ where Let $u = e^{-\frac{2\pi \jmath}{n+1}}$ be the $n+1^{\text{th}}$ root of unit. The Vandermonde matrix formed by these points, i.e.,

$$\mathbf{F} = \begin{bmatrix} u^{0\cdot 0} & u^{0\cdot 1} & \dots & u^{0\cdot n} \\ u^{1\cdot 0} & u^{1\cdot 1} & \dots & u^{1\cdot n} \\ \vdots & \vdots & \ddots & \vdots \\ u^{n\cdot 0} & u^{n\cdot 1} & \dots & u^{n\cdot n} \end{bmatrix}$$

has a very nice property that

$$\mathbf{F}^{-1} = \frac{1}{n+1}\mathbf{F}^*$$

where $\mathbf{F}^*$ is the conjugate of $\mathbf{F}$ (This can be verified easily). Therefore, we can

obtain $\mathbf{F}^{-1}$ for free. The coefficients can be simply obtained by

$$(c_0, \ldots, c_n)^T = \frac{1}{n+1}\mathbf{F}^* \left(f(u^n), \ldots, f(u^0)\right)^T.$$

Bibliography

[1] "St. Petersburg paradox," http://en.wikipedia.org/wiki/St._Petersburg_paradox.

[2] M. Abramowitz and I. A. Stegun, Eds., *Handbook of Mathematical Functions*. National Bureau of Standards, Applied Mathematics Series - 55, 1972.

[3] H. Ackermann, A. Newman, H. Röglin, and B. Vöcking, "Decision making based on approximate and smoothed pareto curves," *International Symposium on Algorithm and Computation*, pp. 675–684, 2005.

[4] M. Adamczyk, "Improved analysis of the greedy algorithm for stochastic matching," *Information Processing Letters*, vol. 111, no. 15, pp. 731–737, 2011.

[5] P. Agrawal and J. Widom, "Continuous Uncertainty in Trio," in *MUD*, 2009.

[6] S. Agrawal, A. Saberi, and Y. Ye, "Stochastic Combinatorial Optimization under Probabilistic Constraints," *Arxiv preprint arXiv:0809.0460*, 2008.

[7] N. Ailon, "Aggregation of partial rankings, p-ratings and top-m lists," in *ACM-SIAM Symposium on Discrete algorithms*, 2007, pp. 415–424.

[8] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: Ranking and clustering," in *Journal of the ACM*, vol. 55(5), 2008.

[9] S. Alimov, R. Ashurov, and A. Pulatov, "Multiple fourier series and fourier integrals, in commutative harmonic analysis. IV: Harmonic analysis in $\mathbb{R}_n$," *Encyclopedia of Mathematical Science*, vol. 42, 1992.

[10] N. Alon and J. H. Spencer, *The probabilistic method*. Wiley-Interscience, 2008.

[11] P. Andritsos, A. Fuxman, and R. J. Miller, "Clean answers over dirty databases," in *IEEE International Conference on Data Engineering*, 2006.

[12] L. Antova, C. Koch, and D. Olteanu, "From complete to incomplete information and back," in *ACM SIGMOD International Conference on Management of Data*, 2007.

[13] S. Arumugam, R. Jampani, L. Perez, F. Xu, C. Jermaine, and P. Haas, "MCDB-R: Risk Analysis in the Database," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010.

[14] M. Atallah and Y. Qi, "Computing all skyline probabilities for uncertain data," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, 2009, pp. 279–287.

[15] B. Bahmani and M. Kapralov, "Improved bounds for online stochastic matching," in *European Symposia on Algorithms*. Springer, 2010, pp. 170–181.

[16] N. Bansal, A. Gupta, J. Li, J. Mestre, V. Nagarajan, and A. Rudra, "When LP is the Cure for Your Matching Woes: Improved Bounds for Stochastic Matchings," *European Symposia on Algorithms*, pp. 218–229, 2010.

[17] ——, "When LP is the Cure for Your Matching Woes: Improved Bounds for Stochastic Matchings," *Algorithmica*, pp. 1–30, 2011. [Online]. Available: http://dx.doi.org/10.1007/s00453-011-9511-8

[18] N. Bansal, N. Korula, V. Nagarajan, and A. Srinivasan, "On k-column sparse packing programs," *Integer Programming and Combinatorial Optimization*, pp. 369–382, 2010.

[19] F. Barahona and W. Pulleyblank, "Exact arborescences, matchings and cycles," *Discrete Applied Mathematics*, vol. 16, no. 2, pp. 91–99, 1987.

[20] D. Barbara, H. Garcia-Molina, and D. Porter, "The management of probabilistic data," *IEEE Transactions of Knowledge Data Engineering*, vol. 4, no. 5, pp. 487–502, 1992.

[21] J. Bard and J. Bennett, "Arc reduction and path preference in stochastic acyclic networks," *Management Science*, vol. 37, no. 2, pp. 198–215, 1991.

[22] D. Bernoulli, "Exposition of a new theory on the measurement of risk," *Econometrica*, vol. 22, no. 1, pp. 22–36, 1954, originally published in 1738; translated by Dr. Lousie Sommer.

[23] G. Beskales, M. Soliman, and I. IIyas, "Efficient search for the top-k probable nearest neighbors in uncertain databases," *International Conference on Very Large Data Bases*, 2008.

[24] G. Beylkin and L. Monzón, "On Generalized Gaussian Quadratures for Exponentials and Their Applications* 1," *Applied and Computational Harmonic Analysis*, vol. 12, no. 3, pp. 332–373, 2002.

[25] ——, "On approximation of functions by exponential sums," *Applied and Computational Harmonic Analysis*, vol. 19, no. 1, pp. 17–48, 2005.

[26] ——, "On approximation of functions by exponential sums," *Applied and Computational Harmonic Analysis*, vol. 19, pp. 17–48, 2005.

[27] ——, "Approximation by exponential sums revisited," *Applied and Computational Harmonic Analysis*, vol. 28, no. 2, pp. 131–149, 2010.

[28] A. Bhalgat, 2011, personal Communication.

[29] A. Bhalgat, A. Goel, and S. Khanna, "Improved approximation results for stochastic knapsack problems," in *ACM-SIAM Symposium on Discrete algorithms*, 2011.

[30] S. Bhattacharya, G. Goel, S. Gollapudi, and K. Munagala, "Budget constrained auctions with heterogeneous items," in *ACM Symposium on Theory of Computing*. ACM, 2010, pp. 379–388.

[31] J. R. Birge and F. Louveaux, "Introduction to stochastic programming," 1997.

[32] B. E. Birnbaum and C. Mathieu, "On-line bipartite matching made simple," *SIGACT News*, vol. 39, no. 1, pp. 80–87, 2008.

[33] A. Bjorck and V. Pereyra, "Solution of vandermonde systems of equations," *Mathematics of Computation*, vol. 24, no. 112, pp. 893–903, 1970.

[34] J. C. Borda, "Mémoire sur les élections au scrutin," *Histoire de l'Académie Royale des Sciences*, 1781.

[35] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML*, 2005.

[36] R. Carraway, R. Schmidt, and L. Weatherford, "An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns," *Naval research logistics*, vol. 40, no. 2, pp. 161–173, 1993.

[37] L. Chang, J. Yu, L. Qin, and X. Lin, "Probabilistic ranking over relations," in *International Conference on Extending Database Technology*. ACM, 2010, pp. 477–488.

[38] C. Chekuri and S. Khanna, "A PTAS for the multiple knapsack problem," in *ACM-SIAM Symposium on Discrete algorithms*, 2000, pp. 213–222.

[39] J. Chen and K. Yi, "Dynamic structures for top-k queries on uncertain data." in *International Symposium on Algorithm and Computation*, 2007, pp. 427–438.

[40] N. Chen, N. Immorlica, A. R. Karlin, M. Mahdian, and A. Rudra, "Approximating matches made in heaven," in *International Colloquium on Automata, Languages and Programming*, 2009, pp. 266–278.

[41] W. Cheney and W. Light, *A Course in Approximation Theory*. Brook/Cole Publishing Company, 2000.

[42] R. Cheng, J. Chen, M. Mokbel, and C. Chow, "Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data," in *IEEE International Conference on Data Engineering*, 2008.

[43] R. Cheng, J. Chen, and X. Xie, "Cleaning uncertain data with quality guarantees," in *International Conference on Very Large Data Bases*, 2008, pp. 722–735.

[44] R. Cheng, L. Chen, J. Chen, and X. Xie, "Evaluating probability threshold k-nearest-neighbor queries over uncertain data," in *International Conference on Extending Database Technology*, 2009.

[45] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *ACM SIGMOD International Conference on Management of Data*, 2003.

[46] M. J. Condorcet, *Éssai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*, 1785.

[47] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2001.

[48] G. Cormode, F. Li, and K. Yi, "Semantics of ranking queries for probabilistic data and expected ranks," in *IEEE International Conference on Data Engineering*, 2009.

[49] G. Cormode and A. McGregor, "Approximation algorithms for clustering uncertain data," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2008, pp. 09–12.

[50] G. Cormode and M. Garofalakis, "Histograms and wavelets on probabilistic data," in *IEEE International Conference on Data Engineering*, 2009.

[51] G. Cormode and A. McGregor, "Approximation algorithms for clustering uncertain data," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2008.

[52] P. Dagum, R. Karp, M. Luby, and S. Ross, "An optimal algorithm for Monte Carlo estimation," in *Annual IEEE Symposium on Foundations of Computer Science*, 1995.

[53] N. Dalvi, K. Schnaitter, and D. Suciu, "Computing query probability with incidence algebras," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, 2010, pp. 203–214.

[54] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal*, 2006.

[55] ——, "Management of probabilistic data: Foundations and challenges," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2007.

[56] G. Dantzig, "Linear programming under uncertainty," *Management Science*, vol. 1, no. 3, pp. 197–206, 1955.

[57] C. de Boor, *A Practical Guide to Spline*. Springer, 2001.

[58] B. Dean, M. Goemans, and J. Vondrák, "Adaptivity and approximation for stochastic packing problems," in *ACM-SIAM Symposium on Discrete algorithms*, 2005, pp. 395–404.

[59] ——, "Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity," *Mathematics of Operations Research*, vol. 33, no. 4, pp. 945–964, 2008.

[60] O. Dekel, C. Manning, and Y. Singer, "Log-linear models for label-ranking," in *Advances in Neural Information Processing Systems*, 2004.

[61] A. Deshpande, C. Guestrin, and S. Madden, "Using probabilistic models for data management in acquisitional environments," in *CIDR*, 2005.

[62] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks." in *International Conference on Very Large Data Bases*, 2004, pp. 588–599.

[63] X. L. Dong, A. Y. Halevy, and C. Yu, "Data integration with uncertainty," in *International Conference on Very Large Data Bases*, 2007.

[64] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *International World Wide Web Conference*, 2001.

[65] ——, "Rank aggregation revisited," in *Manuscript*, 2001.

[66] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing top k lists," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 1, pp. 134–160, 2003.

[67] ——, "Comparing top k lists," in *ACM-SIAM Symposium on Discrete algorithms*, 2003.

[68] J. Feldman, A. Mehta, V. S. Mirrokni, and S. Muthukrishnan, "Online stochastic matching: Beating $1-1/e$," in *Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2009, pp. 117–126.

[69] J. Finn and J. Frank, "Optimal junction trees," in *Uncertainty in Artificial Intelligence*, 1994.

[70] P. Fishburn, *Utility Theory and Decision Making*. John Wiley & Sons, Inc, 1970.

[71] N. Fuhr and T. Rolleke, "A probabilistic relational algebra for the integration of information retrieval and database systems," *ACM Transactions on Information System*, 1997.

[72] Z. Füredi, J. Kahn, and P. Seymour, "On the Fractional Matching Polytope of a Hypergraph," *Combinatorica*, vol. 13, no. 2, pp. 167–180, 1993.

[73] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, "Dependent rounding and its applications to approximation algorithms," *Journal of the ACM*, vol. 53, no. 3, p. 360, 2006.

[74] M. Garey and D. Johnson, *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*. WH Freeman and Company, San Francisco, California, 1979.

[75] M. Garofalakis and D. Suciu, Eds., *IEEE Data Engineering Bulletin Special Issue on Probabilistic Data Management*, March 2006.

[76] T. Ge, S. Zdonik, and S. Madden, "Top-k queries on uncertain data: on score distribution and typical answers," in *ACM SIGMOD International Conference on Management of Data*, 2009.

[77] S. Geetha and K. Nair, "On stochastic spanning tree problem," *Networks*, vol. 23, no. 8, pp. 675–679, 1993.

[78] A. Goel, S. Guha, and K. Munagala, "Asking the right questions: Model-driven optimization using probes," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2006, pp. 203–212.

[79] A. Goel and P. Indyk, "Stochastic load balancing and related problems," in *Annual IEEE Symposium on Foundations of Computer Science*, 1999, p. 579.

[80] G. Goel and A. Mehta, "Online budgeted matching in random input models with applications to adwords," in *ACM-SIAM Symposium on Discrete algorithms*, 2008, pp. 982–991.

[81] D. Gottlieb and C. Shu, "On the Gibbs phenomenon and its resolution," *SIAM review*, vol. 39, no. 4, pp. 644–668, 1997.

[82] V. Goyal and R. Ravi, "Chance constrained knapsack problem with random item sizes," *Operation Research Letter*, 2009.

[83] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Addison-Wesley, 1994.

[84] G. Grahne, "Horn tables - an efficient tool for handling incomplete information in databases," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 1989.

[85] T. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2007, pp. 31–40.

[86] T. Green and V. Tannen, "Models for incomplete and probabilistic information," in *International Conference on Extending Database Technology*, 2006.

[87] S. Guha and K. Munagala, "Adaptive Uncertainty Resolution in Bayesian Combinatorial Optimization Problems," *ACM Transactions on Algorithms*, 2008.

[88] ——, "Exceeding expectations and clustering uncertain data," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2009, pp. 269–278.

[89] ——, "Approximation algorithms for partial-information based stochastic control with markovian rewards," in *Annual IEEE Symposium on Foundations of Computer Science*, 2007, pp. 483–493.

[90] ——, "Multi-armed bandits with metric switching costs," in *International Colloquium on Automata, Languages and Programming*, 2009, pp. 496–507.

[91] J. Guiver and E. Snelson, "Learning to rank with softrank and gaussian processes," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008.

[92] A. Gupta, R. Krishnaswamy, M. Molinaro, and R. Ravi, "Approximation algorithms for correlated knapsacks and non-martingale bandits," 2011, coRR, abs/1102.3749.

[93] A. Gupta, M. Pál, R. Ravi, and A. Sinha, "Boosted sampling: approximation algorithms for stochastic optimization," in *ACM Symposium on Theory of Computing*. ACM, 2004, pp. 417–426.

[94] E. Hazan, S. Safra, and O. Schwartz, "On the complexity of approximating k-set packing," *Computational Complexity*, vol. 15, no. 1, pp. 20–39, 2006.

[95] M. Henig, "Risk criteria in a stochastic knapsack problem," *Operations Research*, vol. 38, no. 5, pp. 820–825, 1990.

[96] R. Herbrich, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer, "Learning preference relations for information retrieval," in *ICML-98 Workshop: Text Categorization and Machine Learning*, 1998.

[97] J. Hodge and R. E. Klima, *The mathematics of voting and elections: a hands-on approach*. AMS, 2000.

[98] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *ACM SIGMOD International Conference on Management of Data*, 2008.

[99] C. Hurkens and A. Schrijver, "On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for

packing problems," *SIAM Journal on Discrete Mathematics*, vol. 2, no. 1, pp. 68–72, 1989.

[100] I. Ilyas, G. Beskales, and M. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, 2008.

[101] T. Imielinski and W. Lipski, Jr., "Incomplete information in relational databases," *Journal of the ACM*, 1984.

[102] H. Ishii, S. Shiode, and T. Nishida Yoshikazu, "Stochastic spanning tree problem," *Discrete Applied Mathematics*, vol. 3, no. 4, pp. 263–273, 1981.

[103] R. Jampani, F. Xu, M. Wu, L. Perez, C. Jermaine, and P. Haas, "MCDB: a monte carlo approach to managing uncertain data," in *ACM SIGMOD International Conference on Management of Data*. ACM, 2008, pp. 687–700.

[104] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems*, vol. 20, no. 4, 2002.

[105] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee, "Estimating statistical aggregates on probabilistic data streams," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2007.

[106] J.F.Hauer, C. Demeure, and L.L.Scharf, "Initial results in prony analysis of power system response signals," *IEEE Transactions on Power Systems*, vol. 5, no. 1, pp. 80–89, 1990.

[107] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin, "Sliding-window top-k queries on uncertain streams," in *International Conference on Very Large Data Bases*, 2008, pp. 301–312.

[108] T. Joachims, "Optimizing search engines using click-through data," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2002, pp. 133–142.

[109] B. Kalyanasundaram and K. Pruhs, "Online weighted matching," *Journal of Algorithms*, vol. 14, no. 3, pp. 478–488, 1993.

[110] B. Kanagal and A. Deshpande, "Efficient query evaluation over temporally correlated probabilistic streams," in *ICDE*, 2009.

[111] ——, "Indexing correlated probabilistic databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 2009, pp. 455–468.

[112] ——, "Lineage processing over correlated probabilistic databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 2010, pp. 675–686.

[113] O. Kariv and S. Hakimi, "An algorithmic approach to network location problems. II: The p-medians," *SIAM Journal on Applied Mathematics*, vol. 37, no. 3, pp. 539–560, 1979.

[114] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for online bipartite matching," in *ACM Symposium on Theory of Computing*. ACM, 1990, pp. 352–358.

[115] I. Katriel, C. Kenyon-Mathieu, and E. Upfal, "Commitment under uncertainty: Two-stage stochastic matching problems," *Theoretical Computer Science*, vol. 408, no. 2-3, pp. 213–223, 2008.

[116] J. G. Kemeny, "Mathematics without numbers," *Daedalus*, vol. 88, pp. 571–591, 1959.

[117] M. Kendall, "A new measure of rank correlation," *Biometrica*, vol. 30, pp. 81–89, 1938.

[118] B. Kimelfeld and C. Ré, "Transducing markov sequences," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2010, pp. 15–26.

[119] J. Kleinberg, Y. Rabani, and É. Tardos, "Allocating bandwidth for bursty connections," in *ACM Symposium on Theory of Computing*, 1997, p. 673.

[120] C. Koch, "MAYBMS: A System for Managing Large Probabilistic Databases," *Managing and Mining Uncertain Data*, pp. 149–183, 2009.

[121] H. Kriegel, P. Kunath, and M. Renz, "Probabilistic nearest-neighbor query on uncertain objects," in *International Conference on Database Systems for Advanced Applications*, 2007.

[122] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian, "Probview: a flexible probabilistic database system," *ACM Transactions on Database Systems*, vol. 22, no. 3, 1997.

[123] F. Li, K. Yi, and J. Jestes, "Ranking distributed probabilistic data," in *ACM SIGMOD International Conference on Management of Data*, 2009.

[124] J. Li and A. Deshpande, "Consensus answers for queries over probabilistic databases," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2009.

[125] ——, "Ranking continuous probabilistic datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010.

[126] ——, "Maximizing expected utility for stochastic combinatorial optimization problems," in *Annual IEEE Symposium on Foundations of Computer Science*, 2011.

[127] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," in *International Conference on Very Large Data Bases*, 2009.

[128] ——, "A unified approach to ranking in probabilistic databases," *The VLDB Journal*, vol. 20, no. 2, pp. 249–275, 2011.

[129] R. Loui, "Optimal paths in graphs with stochastic or multidimensional weights," *Communications of the ACM*, vol. 26, no. 9, pp. 670–676, 1983.

[130] M. Mahdian, H. Nazerzadeh, and A. Saberi, "Allocating online advertisement space with unreliable estimates," in *ACM Conference on Electronic Commerce*, 2007, pp. 288–294.

[131] V. H. Manshadi, S. O. Gharan, and A. Saberi, "Online stochastic matching: Online actions based on offline statistics," in *ACM-SIAM Symposium on Discrete algorithms*, 2011.

[132] R. Martin, "The St. Petersburg Paradox," *The Stanford Encyclopedia of Philosophy*, 2004, http://plato.stanford.edu/archives/fall2004/entries/paradox-stpetersburg.

[133] A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani, "Adwords and generalized on-line matching," in *Annual IEEE Symposium on Foundations of Computer Science*, 2005, pp. 264–273.

[134] S. Micali and V. V. Vazirani, "An $o(\sqrt{|v||e|})$ algoithm for finding maximum matching in general graphs," in *Annual IEEE Symposium on Foundations of Computer Science*, 1980, pp. 17–27.

[135] S. Mittal and A. Schulz, "A general framework for designing approximation schemes for combinatorial optimization problems with many objectives combined into one," *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pp. 179–192, 2008.

[136] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[137] I. Murthy and S. Sarkar, "Exact algorithms for the stochastic shortest path problem with a decreasing deadline utility function," *European Journal of Operational Research*, vol. 103, no. 1, pp. 209–229, 1997.

[138] ——, "Stochastic shortest path problems with piecewise-linear concave utility functions," *Management Science*, vol. 44, no. 11, pp. 125–136, 1998.

[139] E. Nikolova, "Approximation Algorithms for Reliable Stochastic Combinatorial Optimization," *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pp. 338–351, 2010.

[140] E. Nikolova, M. Brand, and D. Karger, "Optimal route planning under uncertainty," in *Proceedings of International Conference on Automated Planning and Scheduling*, 2006.

[141] E. Nikolova, J. Kelner, M. Brand, and M. Mitzenmacher, "Stochastic shortest paths via quasi-convex maximization," in *European Symposia on Algorithms*, 2006, pp. 552–563.

[142] F. Oberhettinger, *Fourier transforms of distributions and their inverses: a collection of tables.* Academic press, 1973.

[143] D. Olteanu, J. Huang, and C. Koch, "Approximate confidence computation in probabilistic databases," in *IEEE International Conference on Data Engineering*, 2010, pp. 145–156.

[144] M. R. Osborne and G. K. Smyth, "A modified prony algorithm for fitting sums of exponential functions," *SIAM Journal of Scientific Computing*, 1995.

[145] C. Papadimitriou and M. Yannakakis, "On the approximability of trade-offs and optimal access of web sources," in *Annual IEEE Symposium on Foundations of Computer Science*, 2000.

[146] M. Patil, R. Shah, and S. Thankachan, "Fully Dynamic Data Structure for Top-k Queries on Uncertain Data," *Arxiv preprint arXiv:1007.5110*, 2010.

[147] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *International Conference on Very Large Data Bases.* VLDB Endowment, 2007, pp. 15–26.

[148] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "k-Nearest Neighbors in Uncertain Graphs," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010.

[149] M. J. D. Powell, *Approximation theory and methods.* Cambridge University Press, 1981.

[150] A. Ralston and R. Rabinowitz, *A First Course in Numerical Analysis*, 2001.

[151] C. Re, N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic data," in *IEEE International Conference on Data Engineering*, 2007.

[152] C. Ré, J. Letchner, M. Balazinska, and D. Suciu, "Event queries on correlated probabilistic streams," in *ACM SIGMOD International Conference on Management of Data*, 2008.

[153] C. Ré and D. Suciu, "Efficient evaluation of HAVING queries on a probabilistic database," in *DBPL*, 2007.

[154] A. Roth, T. Sonmez, and U. Unver, "Kidney exchange," *Quarterly Journal of Economics*, vol. 119, pp. 457–488, 2004.

[155] ——, "Pairwise kidney exchange," *Journal of Economic Theory*, vol. 125, pp. 151–188, 2005.

[156] H. Safer, J. B. Orlin, and M. Dror, "Fully polynomial approximation in multi-criteria combinatorial optimization," 2004, mIT Working Paper.

[157] A. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working models for uncertain data," in *IEEE International Conference on Data Engineering*, 2006.

[158] A. Schrijver, *Combinatorial Optimization*. Springer-Verlag, 2003.

[159] P. Sen and A. Deshpande, "Representing and querying correlated tuples in probabilistic databases," in *IEEE International Conference on Data Engineering*. IEEE, 2007, pp. 596–605.

[160] P. Sen, A. Deshpande, and L. Getoor, "Exploiting shared correlations in probabilistic databases," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 809–820, 2008.

[161] ——, "Prdb: managing and exploiting rich correlations in probabilistic databases," *The VLDB Journal*, vol. 18, no. 5, pp. 1065–1090, 2009.

[162] H. Shachnai and A. Srinivasan, "Finding large independent sets in graphs and hypergraphs," *SIAM Journal on Discrete Mathematics*, vol. 18, no. 3, pp. 488–500, 2005.

[163] D. Shmoys and C. Swamy, "An approximation scheme for stochastic linear programming and its application to stochastic integer programs," *Journal of the ACM*, vol. 53, no. 6, pp. 978–1012, 2006.

[164] C. Sigal, A. Pritsker, and J. Solberg, "The stochastic shortest route problem," *Operations Research*, vol. 28, no. 5, pp. 1122–1129, 1980.

[165] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang, "A sampling-based approach to optimizing top-k queries in sensor networks," in *IEEE International Conference on Data Engineering*. IEEE, 2006, p. 68.

[166] M. Soliman, I. Ilyas, D. Martinenghi, and M. Tagliasacchi, "Ranking with uncertain scoring functions: Semantics and sensitivity measures," in *ACM SIGMOD International Conference on Management of Data*, 2011.

[167] M. Soliman, I. Ilyas, and C. Chang, "Top-k query processing in uncertain databases," in *IEEE International Conference on Data Engineering*. IEEE, 2007, pp. 896–905.

[168] M. A. Soliman and I. F. Ilyas, "Ranking with uncertain scores," in *IEEE International Conference on Data Engineering*, 2009, pp. 317–328.

[169] E. Stein and R. Shakarchi, *Fourier analysis: an introduction*. Princeton University Press, 2003.

[170] A. H. Stroud and D. Secrest, *Gaussian Quadrature Formulas*. Prentice-Hall Inc., 1966.

[171] C. Swamy, "Risk-Averse Stochastic Optimization: Probabilistically-Constrained Models and Algorithms for Black-Box Distributions." *ACM-SIAM Symposium on Discrete algorithms*, 2010.

[172] C. Swamy and D. Shmoys, "Approximation algorithms for 2-stage stochastic optimization problems," *ACM SIGACT News*, vol. 37, no. 1, p. 46, 2006.

[173] P. Talukdar, M. Jacob, M. Mehmood, K. Crammer, Z. Ives, F. Pereira, and S. Guha, "Learning to create data-integrating queries," in *International Conference on Very Large Data Bases*. VLDB Endowment, 2008, pp. 785–796.

[174] M. Taylor, J. Guiver, S. Robertson, and T. Minka, "Softrank: optimizing non-smooth rank metrics," in *WSDM*, 2008.

[175] T. Tran, L. Peng, B. Li, Y. Diao, and A. Liu, "PODS: A new model and processing algorithms for uncertain data streams," in *ACM SIGMOD International Conference on Management of Data*. ACM, 2010, pp. 159–170.

[176] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, 2nd ed. Princeton Univ. Press, 1947.

[177] Y. Wakabayashi, "The complexity of computing medians of relations," in *Resenhas*, vol. 3(3), 1998, pp. 323–349.

[178] D. Wang, E. Michelakis, M. Garofalakis, and J. M. Hellerstein, "BayesStore: Managing large, uncertain data repositories with probabilistic graphical models," in *International Conference on Very Large Data Bases*, 2008.

[179] M. Wick, A. McCallum, and G. Miklau, "Scalable Probabilistic Databases with Factor Graphs and MCMC," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010.

[180] J. Widom, "Trio: A system for integrated management of data, accuracy, and lineage," in *The Conference on Innovative Data Systems Research*, 2005.

[181] K. Yi, F. Li, G. Kollios, and D. Srivastava, "Efficient processing of top-k queries in uncertain databases with x-relations," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1669–1682, 2008.

[182] K. Yi, F. Li, D. Srivastava, and G. Kollios, "Efficient processing of top-k queries in uncertain databases," in *IEEE International Conference on Data Engineering*, 2008.

[183] X. Zhang and J. Chomicki, "On the semantics and evaluation of top-k queries in probabilistic databases," in *DBRank*, 2008.