University of Alberta

Top-k Ranking with Uncertain Data

by

Chonghai Wang

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Spring 2011
Edmonton, Alberta

Abstract

The goal of top-k ranking is to rank individuals so that the best k of them can be determined. Depending on the application domain, an individual can be a person, a product, an event, or just a collection of data or information for which an ordering makes sense. The problem of top-k ranking has profound commercial and social implications.

In the context of databases, top-k ranking has been studied in two distinct directions, depending on whether the stored information is certain or uncertain. In the former, since there is little dispute on what top-k ranking is, the past research has focused on efficient query processing. In the latter case, a number of semantics based on possible worlds have been proposed and computational mechanisms investigated for what are called *uncertain databases* or *probabilistic databases*, where a tuple is associated with a membership probability indicating the level of confidence on the stored information.

In this thesis, we study top-k ranking with uncertain data in two general areas. The first is on pruning for the computation of top-k tuples in a probabilistic database. We investigate the theoretical basis and practical means of pruning for the recently proposed, unifying framework based on *parameterized ranking functions*. As such, our results are applicable to a wide range of ranking functions. We show experimentally that pruning can generate orders of magnitude performance gains. In the second area of our investigation, we study the problem of top-k ranking for objects with multiple attributes whose values are modeled by probability distributions and constraints. We formulate a theory of top-k ranking for objects by a characterization of what constitutes the strength of an object, and show that a number of previous proposals for top-k ranking are special cases of our theory. We carry out a limited study on computation of top-k objects under our theory. We reveal the close connection between top-k ranking in this context and high-dimensional space studied in mathematics, in particular, the problem of computing the volumes of high-dimensional polyhedra expressed by linear inequations is a special case of top-k ranking of objects, and as such, the algorithms formulated for the former can be employed for the latter under the same conditions.

Acknowledgements

I would like to thank my supervisors: Dr. Li Yan Yuan and Dr. Jia-Huai You. Without your supervision in the last five and a half years, it is impossible for me to finish this thesis. From you I learned a lot and I will benefit from it for all my life. I want to thank Dr. Osmar R. Zaiane, Dr. Joerg Sander and Dr. Xi Chen for your kind help in my research work. I also want to thank Dr. Xuemin Lin for your service as the external examiner on my thesis defense.

I want to thank all the persons who gave me help during my PhD study at the University of Alberta. I am lucky to meet many nice people on campus. I will remember and treasure the time I have spent here.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the real world, we often need to rank things. For example, when a search engine searches something on the Internet, it often needs to rank a large number of web pages and return the most relevant ones as the result. When querying a database, there could be many tuples satisfying a given requirement, which need to be ranked and the most relevant ones returned. In general, when many things satisfy a given requirement, we are interested only in the most relevant ones; in particular, we typically do not care about the ranks of the rest.

The general problem of top-k ranking is to rank individuals so that the top-k of them can be determined. In an application, any top-k individual should be at least as desirable as anyone not in the top-k list. Here, individuals can be anything on which an ordering makes sense. They can be objects (concrete or abstract), events, tuples in a database; e.g., candidates for a job or an election, leads in a criminal case investigation, patients waiting for treatment, commercial products of a certain kind, popularity of actors/actresses, movies, songs, performance in a sport or artistic competition, etc.

Apparently, the problem of top-k ranking has profound commercial and social implications. It should also be clear that top-k ranking is often problem-specific, due to the fact that a reasonable ranking must heavily involve domain knowledge and ranking results are subject to interpretations.

When restricted to databases, that is, if we assume that the information for a particular ranking problem can be suitably represented as database objects, the problem of top-k ranking becomes manageable. In this context, an object may have one or more grades, or scores, one for each attribute. For example, an object may have a color grade to tell how red it is and a size grade to tell how large it is. Each object can be assigned an overall grade by combining the attribute grades using an aggregation function. Then, the top-k objects are the k objects with the highest overall grades. Here, the definition of top-k objects is clear,

and the main challenge is to compute top-k objects efficiently in a database context, e.g., by using the Threshold Algorithm [1] and its variants [2, 3].

The data above is certain. However, when the data in a database is uncertain, or difficult to be characterized quantitatively, the problem of top-k ranking presents an additional challenge - the semantics of top-k ranking. For instance, we may not know the exact price of a car, but a probability distribution of the prices may be available; we may be uncertain about a person's height which is known to be between $1.7m$ and $1.8m$; we may know that a professional is more experienced than another one, and so on. It is in general difficult to translate this kind of knowledge into quantities.

The general problem of top-k ranking with uncertain data is highly complex and challenging. It involves knowledge representation and reasoning with uncertain data; it may require the use of machine learning techniques to generate useful information, such as relations, from raw data; and the computational problem could be infeasible in general.

One recent approach in databases is to assume a limited form of uncertain data, represented by tuples, each with a membership probability to indicate the level of confidence of the stored information (in the literature sometimes this is also explained as the probability of the physical existence of the tuple in a database). This form of uncertain data often arises in real applications. For example, due to various factors, we may know the probability of an event, such as the probability of a certain speed of a certain car at a certain location. Information organized this way is called an *uncertain database*, or an *uncertain table*, or a *probabilistic database*. There are in general two problems involved. One is the semantics of top-k ranking, namely how top-k objects are determined semantically, and the other is how to compute the top-k tuples/objects according to the semantics.

Uncertain data can be described in different forms, e.g., by a probability distribution. For example, we may know the probability distribution of the heights of the group of persons in an application domain. Uncertain data may also be described by relations. For example, we may know that one actor is more popular than another one.

In this thesis, we study two important issues in top-k ranking. One is how to improve the computational efficiency of top-k ranking in probabilistic databases through pruning. This is studied in the context of *tuple ranking* where the semantics is defined by what are called *parameterized ranking functions*. A number of algorithms have been proposed for this semantics. We show how these algorithms can be improved by pruning, in a systematic fashion. We will also study *object ranking*, where objects are defined by one or more attributes whose values are expressed by probability distributions and constraints. In the

rest of this chapter, we will provide a more detailed account of these two topics.

1.1 Pruning for Top-k Ranking in Uncertain Databases

Uncertain databases, also called probabilistic databases, are proposed to deal with uncertainty in a variety of application domains, such as in sensor network, data cleaning, information retrieval, text analytics and social networks analysis [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. Some system are built to manage uncertain data [15, 16, 17, 18]. An uncertain database consists of a set of tuples each of which comes with a numeric value representing the *score* of the tuple and a *membership probability*. Uncertainty may be due to incompleteness of data, limitation of equipment, or loss in data transfer, etc. Different uncertain data models have been proposed for uncertain databases [19, 20, 21, 22, 23, 24, 25, 26, 27], some adopting the possible world semantics. As a reasonable approximation to the uncertain nature of data, the data model based on *x-tuples* is often adopted in the study of uncertain databases, where mutual exclusive correlations between tuples are specified in terms of *generation rules* – two tuples involved in the same generation rule cannot be true simultaneously in the real world, and thus should not appear in the same possible world.

The problem of top-k ranking in this context is to rank tuples in a database so that the best k of them can be determined. In uncertain databases, the interaction of the information associated with a tuple – the score representing the importance of the tuple, and the likelihood of a tuple representing the true information (or its existence), has made top-k ranking an intriguing issue.

Different semantics of top-k tuples in uncertain databases have been proposed [28, 29, 30, 31, 32]. In [29][1] however, a general framework of top-k ranking, based on *parameterized ranking function* (PRF) is formulated, which generalizes many of the previously proposed ranking functions. Two classes of parameterized ranking functions, under the names PRF^{ω} and PRF^{e}, are proposed, and their computational properties studied under the *probabilistic and/xor tree model* [33], of which x-tuples is a special but a dominating case. One can say that the framework of PRF is currently the most powerful one for top-k ranking in uncertain databases, where x-tuples serves as one of the most important data models.

Depending on the underlying ranking function, a top-k algorithm generally runs quadratic time or higher, which is considered too expensive for large databases. There are generally

[1] The paper won the best paper award of VLDB2009.

two ways to improve the performance. On the one hand, we may design approximation algorithms, and on the other we can try to safely omit the computation of the ranking function values of some tuples that are guaranteed not to be in top-k. The latter is called *pruning*. In general, pruning does not improve the complexity, since in the worst case nothing may be pruned. Some pruning techniques have been proposed in the past, but they are formulated only for some fixed semantics (e.g., [28]).[2] As a result, it is often not clear whether they are applicable to different ranking functions.

In [29], methods are proposed to compute the PRF value (for PRF^ω as well as PRF^e) of a tuple in an uncertain table. The algorithms based on these methods compute all PRF values of all the tuples and then choose the k tuples with largest PRF values to be the top-k tuples. In general, to get top-k tuples we may not need to compute the PRF value of every tuple. This is the case where we know that some upper bound of the PRF value of a tuple t is smaller than the kth largest PRF value found so far (so that it is guaranteed that t is not a top-k tuple). In this case, we say that the computation of the PRF value of t is pruned. Apparently, we need a mechanism to compute an upper bound of the PRF value of a tuple, with two distinct features. First, the cost of computing an upper bound of a tuple should be substantially smaller than the cost of the computing its PRF value. Second, such an upper bound should be sufficiently tight, as there can be useless, trivial upper bounds (e.g., some sufficiently large number). In this way, a top-k algorithm with pruning may improve the efficiency of computing top-k tuples.

In this thesis we present a general approach to pruning for the framework based on PRF^ω and PRF^e. We reveal a series of relationships among tuples and show how they can be used to derive efficient pruning methods. Our results are applicable to the x-tuple model for a wide range of ranking functions. As a result, it is now possible to augment a top-k algorithm for an instance of PRF^ω with pruning, in a systematic fashion. Our experiments show orders of magnitude speedup over algorithms without pruning, and substantial improvement over the existing pruning methods. The theoretical development of these results will be presented in Chapter 3 and the practical pruning methods presented in Chapter 4. The pruning method for PRF^e is studied in Chapter 5. The experiments are reported in Chapter 6.

[2]Pruning was also studied for a ranking function called *Expected Ranks* in [34], which does not belong to the family of PRFs, and therefore does not apply to the latter in general.

1.2 Theory of Top-k Ranking for Objects with Uncertain Data

The top-k ranking above is to rank tuples in an uncertain database. Even if in some cases a tuple may be viewed as an object,[3] there is only one attribute represented by *score*. In general, we would like to consider the general problem of top-k ranking for objects, where an object in this context is an entity which consists of a number of attributes, whose roles in the object are determined by an aggregation function. The problem of top-ranking in this case is conceptually simple for data that are complete and certain - the aggregation value of an object represents its strength and therefore its rank. For uncertain data, however, the problem becomes challenging, as the basis of such a theory is unclear.

As an example, let us consider the problem of renting an apartment from a group of the best k choices, based on a number of factors. For simplicity, let us consider two attributes, *prices* and *location*. The uncertainty of the former may be described by a probability distribution in a range of dollar values. The judgement of location could be fuzzy too; say we have 4 ranks for location: $excellent$, $good$, $fair$, $poor$, and we may know that a location is good or excellent but not sure which one it should be. Assuming that the user provides the weights of the factors on prices and location (i.e., an aggregation function), we should be able to generate the top-k apartments from the uncertain data.

In the presence of uncertain data, the central question is to how to determine the strength of an object, or what constitutes the strength of an object. For technical insights, let us consider a simple example. Suppose there are two objects, A and B, with only one attribute α, and we do not know the exact values of A and B under α.; we only know that the domain of the values of A under α is $\{2, 4\}$ and that of B under α is $\{1, 3\}$. Let us assume that the probability distribution of values of A and B under α are both uniform. That is,

$$Pr(A = 2) = 0.5,\ Pr(A = 4) = 0.5,\ Pr(B = 1) = 0.5,\ Pr(B = 3) = 0.5$$

At the first it appears unclear as how to rank A and B. If A takes value 4 and B takes value 1, then A should rank before B. If A takes value 2 and B takes value 3, then A should rank after B. The two different ranks are both possible. It looks like we cannot rank A and B in this case. But we may consider the likelihood of which one would get the most *support*, by

[3] In general, this may not the case, e.g., two different tuples may represent the same car at two different locations at the same time.

listing all the possible combinations of the values:

$$\{A = 2, B = 1\}$$
$$\{A = 2, B = 3\}$$
$$\{A = 4, B = 1\}$$
$$\{A = 4, B = 3\}$$

Since the probability distributions of values of A and B under α are both uniform, these four combinations have the same probability so that they have the same weights in the final ranking results. In all four possible combinations, A ranks before B in three and B ranks before A in one. Then we can say that A has more supports to be ranked ahead of B. So we can conclude that A is a better choice.

Constraints may change the ranking result. For example, suppose there are two objects A and B, with the domain of the values of A under attribute α being $\{2, 4, 6, 8\}$ and the domain of the values of B under attribute α being $\{1, 3, 5, 7\}$. Let us assume that the probability distributions of values of A and B under α are both uniform. If we list all the combinations, we can see that A has more supports to be ranked before B. Now let us assume a constraint between them: $B > A - 2$. In this case we can see that it is B that is supported by more combinations to be on top.

In the literature, probability distributions have been employed in top-k ranking. In [34], the authors propose what is called *top-k ranking for attribute-level uncertainty model*, in which an uncertain database is a table of tuples, each possessing one attribute whose value is uncertain. Here, a tuple can be thought of as an object with one attribute. The value of an object under this attribute is represented by a discrete probability distribution. In [35], the authors propose to rank records with uncertain scores in databases. In some applications, the score of a record is modeled as a probability density function and we want to find the top-k records. Here, a record can be viewed as an object.

As illustrated earlier, uncertain information may be presented in forms different from probabilities, for example, by relations. A noticeable example is the practice of getting a *short list*. Consider a simple popularity contest: Given three contestants A, B, and C, suppose we know that A is more popular than B; but there is no information as how A is compared to C, neither B to C. Most observers will conclude that A is the top choice; however, the question of the top-2 contestants seems not so obvious.

Relations have been employed in top-k ranking. For example, in [36] a type of top-k queries is proposed, in which we do not know the exact value of an object, yet information about some relations between objects may be available. For instance, we may want to judge the popularity of some actors, but we do not know the score of popularity of an actor. We

only know some actors are preferred over some others. In [36] a method is proposed to find the top-k favorable objects based on preference relations.

As another example of the use of relations in top-k ranking, the well-known algorithm, PageRank [37], is to rank web pages on the Internet. The information used in ranking is the reference relation between web pages (i.e. linkages between web pages). The link structure can be captured by a system of linear equations, from which the page rank of a web page can be computed.

In this thesis, we study the semantics of top-k ranking for objects modeled by uncertain data, where the values of an object's attributes are expressed by probability distributions and constrained by some stated constraints. Under this setting, we present a theory of top-k ranking for objects so that their strengths can be determined in the presence of uncertain data. We show that top-k ranking for objects under some restrictions is closely related to high-dimensional space, in particular, the problem of computing volumes of a high-dimensional polyhedron represented by a system of inequations can be viewed as a subproblem of top-k object ranking in our theory. This ranking theory will be described in Chapter 7.

1.3 Organization

This thesis is organized as follows. Chapter 2 introduces the background on top-k ranking, including different top-k ranking semantics in uncertain databases and some other ranking problems. Chapter 3 develops a theory of how to generate an upper bound of PRF values in uncertain databases. We study the relations of PRF values of tuples and use these relations to compute an upper bound of a tuple's PRF value. Then we can use this upper bound for pruning. Chapter 4 derives practical pruning methods based on the upper bound theory of Chapter 3 and provides algorithm and complexity analysis for PRF^{ω}, followed by practical pruning method for PRF^{e} in Chapter 5. Chapter 6 reports experimental results for our pruning methods, which show a significant improvement on computational efficiency. Chapter 7 presents a ranking theory for objects with uncertain data. We present our theory in three stages. The first deals with discrete domains, which is extended ro include continuous domains. We further extend this theory to add weights to objects' positions and aggregation values in determination of ranking results. Chapter 8 concludes this thesis and comments on the future directions.

Here is a summary of the main contributions of this thesis:

- We formulate a general framework of pruning for top-k ranking in uncertain databases based on parameterized ranking functions. This formulation addresses the issues such as, what is an upper bound, how to determine it in principle, what is an optimal upper bound, and if an upper bound is not optimal how to improve it.

- Based on the general framework of pruning, we derive practical pruning methods and show experimentally that they generate significant improvement in the computation of top-k tuples. The experiments provide insights in the effectiveness of pruning in the process of determining top-k tuples.

- We formulate a theory of top-k ranking for objects with multiple attributes, and reveal that the problem studied in mathematics, namely the problem of determining the volumes of high-dimensional polyhedra expressed by linear inequations is a special case of top-k ranking for objects in our theory.

Here is a summary of publications related to this thesis:

- The work in Chapters 3,4,5, and 6 has been submitted to VLDB2011 and we are invited to submit a revised version for the 2nd round review. We have submitted the revised version.

- The work in Chapter 7 has been accepted by the Journal of Computers and Mathematics with Applications.

Chapter 2

Background

This chapter provides a brief review of the previous work on the topic of top-k ranking and familiarizes the reader with the main concepts and terminologies needed in the rest of this thesis. Much of the material in this chapter can be read briefly, without affecting the understanding of the main results of the thesis.

Here is a road map for how this chapter may be read. Section 2.1 is about top-k ranking for objects with complete data, which can be processed by the well-known Threshold algorithm of Fagin and its variants. The concept of object here is relevant, as it motivated us to formulate top-k ranking for objects with uncertain data later in this thesis. Section 2.2 introduces the notion of possible world, based on which various semantics of top-k ranking are defined. The concept of possible world and related terminologies are important, but the reader need not understand the full detail of every semantics, e.g., Subsections 2.2.2 and 2.2.3 can be safely skipped in the first reading. The definition of parameterized ranking functions is essential, both for discrete domains (Section 2.2.4) and continuous domains (Section 2.3). But the latter becomes technically relevant only in Chapter 7, so it can be read as needed. Then, one can go through the later sections of this chapter briefly.

2.1 Threshold Algorithm

This is the case of top-k ranking with data that are complete and certain in a database context.

We assume that the objects in a database possess the same attributes and the availability of an aggregation function. Then a top-k query is to determine the top-k objects, i.e., the k objects with highest aggregation values. In [1], Fagin proposes the Threshold Algorithm (TA) to compute the top-k objects. For each attribute, the algorithm maintains a sorted list of the values for all the objects. In each round, the algorithm retrieves a value from

each attribute and calculate the overall grade of the corresponding object. The algorithm maintains a virtual object. The value of each attribute of this virtual object is the lowest value retrieved from the sorted list for each attribute. Fagin proves that the overall grade of this virtual object is greater than the overall grade of any object which has not been retrieved. After some rounds, when the algorithm finds k objects with higher overall grades than the overall grade of the virtual object in the last round, it halts. The threshold algorithm provides an efficient way to find top-k objects in a database.

Many variants of TA have been proposed and their properties studied [2, 3, 38, 39, 40, 41, 42, 43, 44, 45].

2.2 Ranking in Uncertain Databases

Under the x-tuple model, an *uncertain database* (or an *uncertain table*) T contains a set of tuples, each t of which is associated with a membership probability, denoted by $Pr(t)$, such that $Pr(t) > 0$. Each tuple t is associated with a score, $score(t)$, which is determined by a scoring function: $T \rightarrow \Re$. A *generation rule* r of an uncertain database T is an exclusive relation of one or more tuples in T, written as $r = t_1 \oplus t_2 \oplus ... \oplus t_e$ (as an alternative notation, we will also use r as the set of these tuples), and the sum of the membership probabilities of the involved tuples, denoted by $Pr(r)$, is less than or equal to 1. We assume each tuple in T appears in one and only one generation rule. When $Pr(r) < 1$, to represent the probability of the missing information, we define $Pr(\bar{r}) = 1 - Pr(r)$. A tuple involved in a single-tuple generation rule is called an *independent tuple*. Table 2.1 below shows an uncertain table with eight tuples. The score of a tuple is "Drifted Days" and the membership probability of a tuple is shown on the last column. We will explain the intended application shortly.

	Location	Time	Drifted Days	Member Prob
t_1	a_1	b_1	22	0.5
t_2	a_1	b_1	9	0.5
t_3	a_2	b_2	16	0.5
t_4	a_2	b_2	15	0.5
t_5	a_3	b_3	18	0.6
t_6	a_3	b_3	10	0.4
t_7	a_4	b_4	14	0.3
t_8	a_4	b_4	11	0.7

Table 2.1: An uncertain table

The notion of generation rules intuitively corresponds to events that cannot take place simultaneously. For example, in the real world, the same car cannot appear in two distant places at the same time. But different sensor equipments may report the spotting of the car at different locations at about the same time, each with some certainty which is represented by a probability.

Let T be an uncertain database. A *possible world* W is a set of tuples in T, such that for each generation rule r on T, W consists of exactly one tuple in r if $Pr(r) = 1$, and zero or one tuple in r if $Pr(r) < 1$. The *probability* of W, denoted by $Pr(W)$, is the product of the membership probabilities of all the tuples in W and all of $Pr(\bar{r})$, for each r where W contains no tuples from it. Note that $Pr(W) > 0$. Let Λ be the set of all the possible worlds. It is clear that $\sum_{W \in \Lambda} Pr(W) = 1$, where Λ denotes the set of all possible worlds for T. Table 2.2 shows the possible worlds for the uncertain table in Table 2.1 .

Let us consider the following example of an uncertain database.

Example 2.2.1 Consider a database that stores the sighting information about drifted icebergs. We can get the sighting information of icebergs from different sources: radar, visual, satellite, etc. Each source has different confidence. Each tuple in this database contains the sighting time, sighting location, number of drifted days and the confidence. If some tuples have the same time and location, we will think they refer to the same iceberg and these tuples are involved in the same generation rule. Table 2.1 is such an uncertain database. There are 8 tuples in the table. The generation rules are:

$$r_1 = t_1 \oplus t_2$$
$$r_2 = t_3 \oplus t_4$$
$$r_3 = t_5 \oplus t_6$$
$$r_4 = t_7 \oplus t_8$$

The possible worlds and their probabilities are given in Table 2.2.

The tuples of an uncertain database may have many attributes. We are interested in ranking one of its numerical attributes. Top-k ranking for certain data of one attribute is obvious. The k tuples with highest values of this numerical attribute are top-k tuples. But in an uncertain world, there has been no such definitive agreement. The results of the top-k ranking depend on both the values of the numerical attribute and the membership probabilities of tuples.

Top-k ranking has been studied for uncertain databases with different semantics proposed [28, 29, 30, 31, 34, 46]. We note that, except the top-k ranking semantics proposed

Possible World	Prob
$PW^1 = \{t_1, t_3, t_5, t_7\}$	0.045
$PW^2 = \{t_1, t_3, t_5, t_8\}$	0.105
$PW^3 = \{t_1, t_3, t_6, t_7\}$	0.03
$PW^4 = \{t_1, t_3, t_6, t_8\}$	0.07
$PW^5 = \{t_1, t_4, t_5, t_7\}$	0.045
$PW^6 = \{t_1, t_4, t_5, t_8\}$	0.105
$PW^7 = \{t_1, t_4, t_6, t_7\}$	0.03
$PW^8 = \{t_1, t_4, t_6, t_8\}$	0.07
$PW^9 = \{t_2, t_3, t_5, t_7\}$	0.045
$PW^{10} = \{t_2, t_3, t_5, t_8\}$	0.105
$PW^{11} = \{t_2, t_3, t_6, t_7\}$	0.03
$PW^{12} = \{t_2, t_3, t_6, t_8\}$	0.07
$PW^{13} = \{t_2, t_4, t_5, t_7\}$	0.045
$PW^{14} = \{t_2, t_4, t_5, t_8\}$	0.105
$PW^{15} = \{t_2, t_4, t_6, t_7\}$	0.03
$PW^{16} = \{t_2, t_4, t_6, t_8\}$	0.07

Table 2.2: Possible worlds for uncertain table in Table 2.1

in [29], all the other semantics are defined under the x-tuple model. In this section, we will introduce the core formulations of top-k ranking and associated algorithms.

2.2.1 U-Topk and U-kRanks

In [30], the authors define top-k query answer for uncertain databases based on possible worlds semantics. To motivate their approach, they provide the following example [1].

Example 2.2.2 Consider a radar-controlled traffic, where car speed readings are stored in a database. Radar units detect speed automatically, while car identification, e.g., by plate number, is usually performed by a human operator. In this database, multiple sources of errors (uncertainty) exist. E.g., radar readings can be interfered by high voltage lines, close-by cars cannot be precisely distinguished, or human operators might make identification mistakes. Suppose Table 2.3 is a snapshot of a radar database in the last hour. Each reading is associated with a confidence field "Conf" indicating its membership probability. Based on radar locations, the same car cannot be detected by radars at two different locations within 1 hour interval. This constraint is captured by the exclusiveness rules: $(t2 \oplus t3), (t4 \oplus t5), (t6 \oplus t7)$. Table 2.4 lists all the possible worlds and their probabilities.

[1] We slightly change the example.

	Time	Radar	Car Model	Plate No	Speed	Prob
t_1	11:45	L1	Honda	X-123	120	1.0
t_2	11:50	L2	Toyota	Y-245	130	0.7
t_3	11:35	L3	Toyota	Y-245	95	0.3
t_4	12:10	L4	Mazda	W-541	90	0.4
t_5	12:25	L5	Mazda	W-541	110	0.6
t_6	12:15	L6	Chevy	L-105	105	0.5
t_7	12:20	L7	Chevy	L-105	85	0.4

Table 2.3: A sample uncertain database

World	Prob
$PW^1 = \{t_1, t_2, t_4, t_6\}$	0.14
$PW^2 = \{t_1, t_2, t_4, t_7\}$	0.112
$PW^3 = \{t_1, t_2, t_4\}$	0.028
$PW^4 = \{t_1, t_2, t_5, t_6\}$	0.21
$PW^5 = \{t_1, t_2, t_5, t_7\}$	0.168
$PW^6 = \{t_1, t_2, t_5\}$	0.042
$PW^7 = \{t_1, t_3, t_4, t_6\}$	0.06
$PW^8 = \{t_1, t_3, t_4, t_7\}$	0.048
$PW^9 = \{t_1, t_3, t_4\}$	0.012
$PW^{10} = \{t_1, t_3, t_5, t_6\}$	0.09
$PW^{11} = \{t_1, t_3, t_5, t_7\}$	0.072
$PW^{12} = \{t_1, t_3, t_5\}$	0.018

Table 2.4: Possible worlds for sample uncertain database in Table 2.3

Two different definitions of top-k query answer are given in [30]. One is called *Uncertain Top-k Query (U- Topk)*, and it is defined as follows.

Definition 2.2.3 Uncertain Top-k Query (U-Topk): *Let D be an uncertain database with possible worlds space $PW = \{PW^1, ..., PW^n\}$. Let $T = \{T^1, ..., T^m\}$ be a set of k-length tuple vectors, where for each $T^i \in T$: (1) Tuples of T^i are ordered according to their scores, and (2) T^i is the top-k answer for a non-empty set of possible worlds $PW(T^i) \subseteq PW$. A U-Topk query, based on scores, returns $T^* \in T$, where $T^* = argmax_{T^i \in T}(\sum_{w \in PW(T^i)}(Pr(w)))$.*

An U-Topk query answer is a tuple vector with the maximum aggregated probability of being top-k across all possible worlds. The U-Top2 query answer for the example above is $\{t_2, t_1\}$ with probability 0.7.

Another notion of top-k query answer is called *Uncertain k Ranks Query (U-kRanks)*. The definition is as follows.

Definition 2.2.4 Uncertain k Ranks Query (U-kRanks): *Let D be an uncertain database with possible worlds space $PW = \{PW^1, ..., PW^n\}$. For $i = 1...k$, let$\{x_i^1, ..., x_i^m\}$ be a set of tuples, where each tuple x_i^j appears at rank i in a non empty set of possible worlds $PW(x_i^j) \subseteq PW$ based on scores. A U-kRanks query , based on scores, returns $\{x_i^*; i = 1...k\}$, where $x_i^* = argmax_{x_i^j}(\sum_{w \in PW(x_i^j)}(Pr(w)))$.*

The tuples of U-kRanks query answer may not form a possible world. But each tuple is the winner in its position. The U-2Ranks query answer for the example above is $\{t_2, t_1\}$.

In [30] algorithms are designed to find top-k tuples for each of the definitions above. They create a state graph for the problem and use an A^*-like search algorithm to find the result. The complexity of the algorithm is exponential. In [31], with the internal structure of the problem, new algorithms are designed with polynomial complexity to find top-k tuples following the same definition given in [30].

2.2.2 PT-k query answer

In [28], the concept of *probability threshold top-k query* (PT-k query) is introduced. First, it defines the top-k probability of a tuple t_i to be the sum of the probabilities of all the possible worlds in which t_i is one of the top-k tuples (k tuples with highest scores). Given a probability threshold p, the answer set of a PT-k query is the set of all tuples whose top-k probability values are at least p. Then, exact and approximate algorithms are proposed to find the answer set of a PT-k query. For Example 2.2.1, the PT-2 query answer is t_1 and t_5.

In [28], a polynomial time algorithm is proposed to compute the PT-k query answer. Pruning is also used in [28] to improve the performance of the proposed algorithms.

2.2.3 Expected rank

In [34], the authors propose a definition of top-k ranking called *expected rank* for attribute-level uncertainty model and tuple-level uncertain model in uncertain databases.

The tuple-level uncertain model is actually the x-tuple model which we have introduced before. In attribute-level uncertain model, the uncertain database is a table of N tuples. Each tuple has one attribute whose value is uncertain. The values of the uncertain attribute of a tuple are described by a discrete probability distribution. A possible world consists of N tuples and each of them takes one value for the uncertain attribute according to its probability distribution. The probability of a possible world is the product of the probabilities of all the values of the tuples in this possible world. The attribute-level uncertain model has many practical applications [47, 48, 49, 50, 51].

The rank of a tuple in a possible world W is defined to be the number of tuples whose values are higher than this tuple. The rank of a tuple t_i in W

$$rank_W(t_i) =\mid \{t_j \in W \mid v_j > v_i\} \mid$$

Let Ω be the set of all the possible worlds. The expected rank of t_i in attribute-level uncertain model

$$r(t_i) = \sum_{W \in \Omega, t_i \in W} Pr(W) \cdot rank_W(t_i)$$

For tuple-level uncertain model, we define $rank_W(t_i) = |W|$ when a possible world W does not contain t_i. The expected rank of t_i in tuple-level uncertain model

$$r(t_i) = \sum_{t_i \in W} Pr(W) \cdot rank_W(t_i) + \sum_{t_i \notin W} Pr(W) \cdot |W| = \sum_{W \in \Omega} Pr(W) \cdot rank_W(t_i)$$

Then the top-k tuples are the k tuples with lowest expected ranks.

In [34], polynomial time algorithms are designed for finding top-k tuples for both attribute-level uncertain model and tuple-level uncertain model. Some pruning methods are applied to improve the performance of the algorithms. In [34], the notion of expected rank is defined formally only for one attribute under ranking. The authors argue that the definition of expected rank can be extended to multiple attributes as well.

2.2.4 Parameterized ranking functions

In [29], the authors propose a definition of top-k ranking in uncertain databases based on the notion of parameterized ranking functions. The *parameterized ranking function (PRF)* for a given tuple t is defined as:

$$\Upsilon(t) = \sum_{W \in PW(t)} \omega(t, \beta_W(t)) \times Pr(W)$$

where $PW(t)$ is the set of all the possible worlds containing t, $\beta_W(t)$ is the *position* of t in the possible world W (according to $score(t)$), and $\omega(t, i)$ is a *weight function*: $T \times N \rightarrow C$ (C is the set of complex numbers). We will call $\Upsilon(t)$ *the* PRF^ω *value of* t.

In [29], a special class of PRF^ω functions, named PRF^e, is proposed. A PRF^e function requires the weight function to be $\omega(i) = \alpha^i$, where α is a constant and may be a real or a complex number.

The approach based on PRF provides a general definition to cover a wide range of ranking functions; e.g., some top-k ranking definitions introduced earlier are special cases.

When we set $\omega(t, i) = score(t)$, this is the *Expected Score* defined in [34]. The top-k tuples under the definition of expected score is the k tuples with highest expected scores.

When we set

$$\omega(i) = \begin{cases} 1 & \text{if } i \leq k \\ 0 & \text{if } i > k \end{cases}$$

this is almost the PT-k query answer defined in [28][2].

When we set

$$\omega_j(i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

for some $1 \leq j \leq k$, we can see that the tuple with largest $\Upsilon_{\omega_j}(t)$ value is the rank-j answer in U-kRanks defined in [30].

In [29], algorithms are formulated for the computation of top-k tuples under the definition of PRF^{ω} and PRF^{e}, respectively. These algorithm are designed for the probabilistic and/xor tree model which is a data model for uncertain databases. The x-tuple model is a special case of probabilistic and/xor tree model. So the algorithms in [29] can be applied to x-tuple model. Under the x-tuple model, two algorithms are proposed for the computation of top-k tuples for PRF^{ω}, one of which runs $O(n^3)$ time and the other $O(n^2 log^2 n)$ time. Under the x-tuple model, the complexity of the algorithm to compute top-k tuples for PRF^{e} is $O(n \log n)$. When all the tuples have been sorted in a decreasing order of their scores, the complexity reduces to $O(n)$.

2.3 Ranking Continuous Probabilistic Datasets

For the top-k ranking we introduced in Section 2.2, the domain of uncertain data is discrete. However continuous domain of uncertain data arises in many areas [52, 53, 54, 55]. In [56], the authors propose a definition of parameterized ranking function to rank tuples with uncertain scores which are captured by a continuous probability distribution. Let us assume we are given a probabilistic dataset consisting of n tuples, each of which has an uncertain score which is described by a continuous probability distribution. For a tuple t_i, let us denote by s_i the random variable corresponding to its score, and by u_i the probability density function of s_i. We define the *support* of u_i as the set of reals where u_i is nonzero. The cumulative density function of s_i is:

$$\rho_i(l) = Pr(s_i \leq l) = \int_{-\infty}^{l} u_i(x)dx$$

[2]The definition here is slightly different from the definition in [28]. We return k tuples with highest PRF values. In [28], they return all the tuples with PRF values higher than a threshold.

A tuple is also associated with an existence probability to represent the probability of existence of this tuple in the dataset (a tuple may or may not exist in the dataset). The uncertain tuples and attribute scores are independent of each other.

A *possible world* consists of some tuples each of which has a value. The domain of a tuple may be infinite (from a continuous domain), so there can be an infinite number of possible worlds. We use $r(t)$ to denote the position of tuple t in a possible world. If a tuple does not exist in a possible world, we denote its position by ∞. $Pr(r(t) = j)$ is the probability that t is ranked at position j. $\omega : T \times N \rightarrow C$ is a weight function that maps a tuple-rank pair to a complex number. The parameterized ranking function is defined as:

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

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

When the probability distribution of the scores is uniform or piecewise polynomial, polynomial-time exact algorithms are given to compute top-k tuples. When the probability distribution is uniform and we do not consider tuple uncertainty, the complexity of the algorithm is $O(\sum_j m_j^3)$. When the probability distribution is piecewise polynomial and we do not consider tuple uncertainty, the complexity of the algorithm is $O(\gamma^2 \sum_j m_j^3)$. Here, γ is the maximum degree of the piecewise polynomials. We introduce m_j below. When considering tuple uncertainty, the complexities of the algorithms are still polynomial.

For a uniform distribution, we assume a tuple t_i's support interval is $[l_i, u_i]$. The probability density function u_i of s_i is uniform over $[l_i, u_i]$. We assume there are n tuples. For simplicity, we assume all the l_i,u_i are different. So there are $2n$ points which partition the real line into $2n + 1$ small intervals. For each of this small interval I_j $(1 \leq j \leq 2n + 1)$, let M_j be the set of tuples whose score interval contains I_j. Let $m_j = |M_j|$. We can see that the maximal value of m_j is n. But in general, it is much smaller than n, and the maximal number of j is $2n + 1$.

For a piecewise polynomial distribution, a tuple t_i's support interval can be divided into some small parts. We assume the maximal number of pieces of a tuple's support interval is τ. Then there are at most $n(\tau+1)$ points which partition the real line into $n(\tau+1)+1$ small intervals. For each of this small interval I_j, let M_j be the set of tuples whose score interval contains I_j. Let $m_j = |M_j|$. We can see that the maximal value of m_j is n. Generally, it is much smaller than n, and the maximal number of j is $n(\tau + 1) + 1$.

2.4 Top-k Ranking with Uncertain Score

In [35], the authors propose to rank records with uncertain scores in databases. In some applications, the score of a record t_i is modeled as a probability density function f_i defined on a score interval $[lo_i, up_i]$. The interval-based score representation can induce a partial order over database records. If a record's lower bound is higher than another record's upper bound, we can order this record ahead of the other record. Otherwise, there is no order between these two records. Then we get a partial order among these records. The linear extensions of the partial order are all the possible total orders that are consistent with this partial order. In this paper, each linear extension is associated with a probability. The paper considers different ranking queries. An *UTop-Rank(i,j)* query reports the most probable record to appear at any rank $i...j$ in possible linear extensions. An l*-UTop-Rank*(i, j) query reports the l most probable records to appear at a rank $i...j$. An *UTop-Prefix(k)* query reports the most probable linear extension prefix of k records. An *UTop-set(k)* query reports the most probable set of top-k records of linear extensions. Approximation algorithms are designed to compute answers to queries.

2.5 Ranking Objects with Relations

Agrawal et al. [36] propose a method for ranking tuples when we only know some relations between tuples. We can treat tuples as objects. The paper gives an example of actors, and we want to rank their popularity. But we do not have exact values to represent the degree of popularity. Instead, we know some preferences between actors, such as A is more popular than B and C, and B is more popular than D. In the paper, it is assumed that this order can be any order, e.g., it doesn't have to satisfy antisymmetry or transitivity. A simple method is then used to find the k most popular actors. Assume there are m different sets of preferences each of which is specified by a partial order on actors. Given a partial order, the method in the paper finds a total order that satisfies the partial order. Then this total order is updated by putting the actor without any preferences with other actors at the end of the order. Then for each actor, a score $n - i + 1$ is assigned, where n is the number of actors and i the position of the actor in this order. Then for each set of preferences, each actor has a score. An actor thus has m scores. The paper gives an aggregation function designed by the authors themselves to get an aggregation score which combines the m scores for each actor. Then the top-k actors are the k actors with highest aggregation scores.

2.6 Ranking in World Wide Web

PageRank [37] is a link analysis algorithm to measure the relative importance of the elements in a linked graph, such as World Wide Web. Generally speaking, this algorithm tries to capture the relations between pages based on analyzing the link structure of the web. These relations can be described by a group of linear equations:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where p_i is a page, $PR(p_i)$ is the page rank of the page p_i, $M(p_i)$ is the set of pages that link to p_i, d is the *damping factor*, $L(p_j)$ is the number of outbound links on page p_j, and N is the total number of pages.

This is a system of linear equations over N variables (pages) and N equations (each page has an equation). So we can compute the page rank $PR(p_i)$ from the system of linear equations.

2.7 Other Research on Ranking

There are some other works on ranking. In [38], an automated ranking method for the query results in a database is proposed. It mimics the idea of TF-IDF (term frequency-inverse document frequency) in Information Retrieval to rank tuples. In [57], new methods are proposed to derive a ranking function for a database query. They adapt and apply principles of probabilistic model from Information Retrieval to structured data. Machine learning methods can also be used for ranking [58, 59, 60].

Chapter 3

Pruning for Top-k Ranking: Theoretical Development

In this chapter, we investigate the theoretical basis for pruning for top-k ranking based on parameterized ranking functions. There are two key ideas in our work. The first is that, given a tuple t, the definition of $PRF^{\omega}(t)$ can be equivalently reformulated as the sum of some partial PRF^{ω} values of t, using a structure devised from the symmetry property of tuple sets. This reformulation tells us what part of computation of PRF^{ω} might be pruned. This is followed by a derivation of a generic method for pruning, which serves as a model for developing concrete pruning methods.

This chapter involves a number of nontrivial technical results and here is a road map for it. The next section provides an outline of computing top-k tuples. That is, given a top-k algorithm, we sketch how pruning may be added into it. This is intended to give the reader an overall picture of the role of pruning, and how it may be integrated into an existing algorithm, in the computation of top-k tuples. The reader can also benefit from the hints on the problems that need to be resolved. In Section 3.2, we study the symmetry property of tuple sets, which forms the basis of the mathematical development that leads to a new representation of PRF^{ω} values, which is presented in Section 3.3. Then, in Section 3.4 we formulate a general method for pruning. Finally in Section 3.5 we provide a summary.

3.1 Outline of Computing Top-k Tuples

Given an algorithm for computing top-k tuples under the definition of PRF, we can combine it with our pruning methods. Below, we outline this process for the algorithms given in [29] for PRF^{ω}, where the tuples of an uncertain database are sorted in a descending order based on their scores, and are retrieved one by one.

We stress the assumption that the tuples stored in an uncertain database are sorted in a descending order of their scores and are retrieved one by one in that order - this is a common assumption in the literature, and an assumption in our work as well. From the (worst case) complexity point of view, sorting tuples takes $O(n \log n)$ time, which is lower than the overall complexity of computing top-k tuples for PRF^ω.[1]

In the combined algorithm, we maintain a heap L_k of the k tuples with the highest PRF^ω values retrieved so far in the descending order of their PRF^ω values. For the first k tuples, they are stored in L_k along with their computed PRF^ω values. For a subsequently retrieved tuple, its PRF^ω value may or may not be computed, depending on the computed upper bound of its PRF^ω value.

We maintain a special tuple, called t_{lowest}, from the retrieved tuples (the details on how to choose it will be given later). The tuple t_{lowest} will be used in computing the upper bound of the next retrieved tuple. Thus, after the first k tuples, we retrieve a tuple t_{new} and compute its upper bound. If this upper bound is less than or equal to $\Upsilon(t)$, for any t in L_k, then it is guaranteed that t_{new} is not a top-k tuple, hence $\Upsilon(t_{new})$ is not computed. Otherwise, $\Upsilon(t_{new})$ is computed and t_{lowest} updated if necessary. If $\Upsilon(t_{new})$ is higher than the lowest PRF^ω value in L_k, we replace a tuple in L_k with the lowest PRF^ω value with t_{new}. After all the tuples are retrieved, L_k holds the top-k tuples.

The most important missing detail in this outline is how to compute an upper bound of $\Upsilon(t_{new})$. Related questions include:

- What could be an upper bound of $\Upsilon(t_{new})$,
- In what sense it is a tight upper bound, and
- Is there a simple way to determine such an upper bound for ranking functions of PRF^e?

These questions will be answered in the rest of this chapter. But first, let us reveal a crucial property in our theoretical development - the symmetry property among tuple sets.

3.2 Symmetry among Tuple Sets

In this section, we study a fundamental property among tuple sets in uncertain databases under the x-tuple model, which is called *symmetry*. A tuple set is just a collection of tuples.

[1] As a related remark, later in this thesis we will often see the phrases like "given two tuples t_1 and t_2 such that $score(t_1) \geq score(t_2)$". This implies that t_1 is retrieved before t_2 in an algorithmic context.

For each generation rule $r_i \in T$, if $Pr(r_i) < 1$, for technical convenience, we create a tuple $\oslash_i$, called *the virtual tuple* of r_i. Virtual tuples do not participate in ranking, hence their scores are irrelevant. Thus we assume a virtual tuple has no score and is not involved in any comparison of scores. But a virtual tuple has a probability, $Pr(\oslash_i) = 1 - Pr(r_i)$. To distinguish, we call the other tuples of this generation rule the *real tuples* of the generation rule. A virtual tuple has the same exclusive relation with other tuples of the same generation rule. Note that the probability of a possible world is the product of the membership probabilities of all the tuples in it, one from each generation rule, be it a real tuple or a virtual tuple. In addition, a tuple set may contain virtual tuples if not said otherwise explicitly.

In the sequel, by abuse of notation, we also use a generation rule r as a set, which consists of all the tuples (including both real and virtual ones) that are involved in r.

Definition 3.2.1 *Let $\eta = \{t_1, ..., t_m\}$ and $\eta' = \{s_1, ..., s_m\}$ be two tuple sets. η and η' are said to be* symmetric *iff*

1. *tuples in η (resp. η') are from different generation rules, and*
2. *for each $t \in \eta$ there is exactly one $s \in \eta'$ from the same generation rule, and vice versa.*

Consider Example 2.2.2. The tuple sets $\{t_2, t_4\}$ and $\{t_3, t_5\}$ are symmetric, so are the tuple sets $\{t_2, t_4\}$ and $\{t_2, t_5\}$.

Symmetric tuple sets possess some interesting properties. The following lemma says that symmetric tuple sets are closely related to symmetric possible worlds. Its proof is routine and omitted

Lemma 3.2.2 *If two tuple sets η and η' are symmetric, then for every possible world W such that $\eta \subset W$, there exists a possible world $W' = (W - \eta) \cup \eta'$, and vice versa.*

The possible worlds W and W' in Lemma 3.2.2 are then said to be *symmetric* w.r.t. η and η'. That is, W' can be obtained by replacing η in W with η', and vice versa.

Note that two symmetric tuple sets may induce more than one pair of symmetric possible worlds. By this lemma, the one-to-one correspondence holds for all induced possible worlds.

An interesting property of symmetric possible worlds is that the ratio of the probabilities of two symmetric possible worlds is the same as that of the product of the membership probabilities of all the tuples in these two tuple sets that induce them.

Lemma 3.2.3 *If two possible worlds W and W' are symmetric w.r.t. η and η', then* $\frac{Pr(W)}{Pr(W')} = \frac{\prod_{t\in\eta} Pr(t)}{\prod_{t\in\eta'} Pr(t)}$.

Proof. Let $W = A \cup \eta$ and $W' = A \cup \eta'$. If $A \neq \emptyset$, we have

$$Pr(W) = \prod_{t\in W} Pr(t) = \prod_{t\in A} Pr(t) \prod_{t\in\eta} Pr(t)$$
$$Pr(W') = \prod_{t\in W'} Pr(t) = \prod_{t\in A} Pr(t) \prod_{t\in\eta'} Pr(t)$$

So we can get $\frac{Pr(W)}{Pr(W')} = \frac{\prod_{t\in\eta} Pr(t)}{\prod_{t\in\eta'} Pr(t)}$. If $A = \emptyset$, we have

$$Pr(W) = \prod_{t\in W} Pr(t) = \prod_{t\in\eta} Pr(t)$$
$$Pr(W') = \prod_{t\in W'} Pr(t) = \prod_{t\in\eta'} Pr(t)$$

In this case we still have $\frac{Pr(W)}{Pr(W')} = \frac{\prod_{t\in\eta} Pr(t)}{\prod_{t\in\eta'} Pr(t)}$. □

Let $Pr(\eta)$ be the sum of the probabilities of the possible worlds in $PW(\eta)$. We then have

Lemma 3.2.4 *For a tuple set η in which all tuples are involved in different generation rules, we have* $Pr(\eta) = \prod_{t_i\in\eta} Pr(t_i)$.

Proof. Let $\eta = (t_1, t_2, ..., t_l)$ and t_i be involved in the generation rule r_{s_i}. Let $r_{v_1}, r_{v_2}, ..., r_{v_h}$ be all the generation rules except $r_{s_1}, r_{s_2}, ..., r_{s_l}$. For each generation rule r_{v_i}, we assume $t_{v_i1}, t_{v_i2}, ..., t_{v_ip_{v_i}}$ are all the tuples involved in it. Note that $\sum_{j=1}^{p_{v_i}} Pr(t_{v_ij}) = 1$.

A possible world in $PW(\eta)$ must contain a tuple t_i from r_{s_i}. Clearly, such a possible world also contains one of the tuples t_{v_ij} from r_{v_i}. Thus, we have

$$\begin{aligned}
&Pr(\eta)\\
&= \prod_{t_i\in\eta} Pr(t_i) \times \sum_{j_1=1}^{p_{v_1}} ... \sum_{j_h=1}^{p_{v_h}} Pr(t_{v_1j_1}) \times ... \times Pr(t_{v_hj_h})\\
&= \prod_{t_i\in\eta} Pr(t_i) \times \sum_{j_1=1}^{p_{v_1}} ... \sum_{j_{h-1}=1}^{p_{v_{h-1}}} Pr(t_{v_1j_1}) \times ... \times Pr(t_{v_{h-1}j_{h-1}}) \times \sum_{j_h=1}^{p_{v_h}} Pr(t_{v_hj_h})\\
&= \prod_{t_i\in\eta} Pr(t_i) \times \sum_{j_1=1}^{p_{v_1}} ... \sum_{j_{h-1}=1}^{p_{v_{h-1}}} Pr(t_{v_1j_1}) \times ... \times Pr(t_{v_{h-1}j_{h-1}})\\
&= ...\\
&= \prod_{t_i\in\eta} Pr(t_i) \times \sum_{j_1=1}^{p_{v_1}} Pr(t_{v_1j_1}) = \prod_{t_i\in\eta} Pr(t_i).
\end{aligned}$$

□

3.3 A New Representation of Υ^ω

Our goal is to develop a general approach to generating an upper bound of $\Upsilon(t)$, for a given tuple t. For simplicity, we may just call it *an upper bound of* t. A key finding in our work is a new representation of PRF^ω values, which is presented in this section.

Warning: The results presented in this section are nontrivial, and sometimes their development could be quite involved, due to various mathematical manipulations of possible worlds. We will give intuitions whenever possible, but the notations could be overwhelming at the first. The reader is advised to go through each concept and notation carefully.

Throughout this section and the rest of this chapter, given an uncertain database T, we consider a set of q tuples $Q = \{t_1, t_2, ..., t_q\}$. As we are interested in the upper bound of a tuple $t \in Q$ and in our method we will use at least one another tuple as a reference, we assume $q \geq 2$. Given such a Q, there is a set of generation rules $R = \{r_1, r_2, ..., r_l\}$ associated with Q, i.e., every tuple in Q is in some generation rule in R and every $r_i \in R$ contains at least one tuple in Q. Clearly, $l \leq q$.

For any $t \in Q$, our interest is to find an upper bound of it. For this, we want to find some real numbers c_i such that

$$\sum_{i=1}^{q} c_i \Upsilon(t_i) \geq 0 \tag{3.1}$$

Note that one of the $t_i \in Q$ is t. Let us denote its coefficient in the above inequation by c. If $c < 0$, the inequation in (3.1) above can be transformed to

$$\Upsilon(t) \leq \sum_{t_i \in Q, t_i \neq t} -\frac{c_i}{c} \Upsilon(t_i) \tag{3.2}$$

That is, the value of $\Upsilon(t)$ cannot be higher than the right hand side of (3.2), which is thus an upper bound of t. The lower such an upper bound, the better (tighter). Since this expression is essential in this thesis and frequently referred to, let us give it a special term - *the upper bound expression*. In particular, the theoretical development for pruning in this chapter is centered around the question of how to assign the coefficients of the upper bound expression such that the inequation in (3.2) holds.

The upper bound expression above, i.e., the inequation in (3.2) does not tell us how to compute such an upper bound, for two reasons:

- we may not have computed all of the $\Upsilon(t_i)$ values of the other tuples in Q, and
- we were not told how to choose c_i so that inequation (3.2) is guaranteed to hold *for any given q tuples*.

The theoretical development of this section aims at the methods that resolve these questions.

3.3.1 A novel representation of PRF^{ω}

We introduced the *parameterized ranking function* in Subsection 2.2.4. That is, such a function Υ on a tuple t is defined as:

$$\Upsilon(t) = \sum_{W \in PW(t)} \omega(t, \beta_W(t)) \times Pr(W) \tag{3.3}$$

where $PW(t)$ is the set of all the possible worlds containing t, $\beta_W(t)$ is the *position* of t in the possible world W (according to $score(t)$), and $\omega(t, i)$ is a *weight function*: $T \times N \rightarrow C$ (C is the set of complex numbers).

In our work, we restrict $\omega(t, i)$ to $\omega(i)$, which means the weight function is independent of t, and the values of $\omega(i)$ to real numbers. Further, we assume $\omega(i)$ is *monotonically non-increasing*. This means $\omega(i) \geq \omega(i+1)$, for all i with $1 \leq i \leq n-1$, assuming n tuples. This assumption is reasonable, since in normal cases a higher position is at least as desirable as those behind it and thus should be given a higher weight. For PRF^{e}, we assume α is a constant real number and $0 < \alpha < 1$. We make this assumption because only when $0 < \alpha < 1$, $\omega(i) = \alpha^{i}$ is monotonically non-increasing (we will not consider the trivial cases where $\alpha = 0$ or 1).

Given a tuple $t_i \in Q$ (note that Q contains only real tuples), its PRF^{ω} value $\Upsilon(t_i)$ is defined above as:

$$\Upsilon(t_i) = \sum_{W \in PW(t_i)} \omega(\beta_W(t_i)) \times Pr(W) \tag{3.4}$$

Note that the t_i here is the t in equation (3.3). This notational change is due to the fact that later we need to refer to each of the elements in Q, and talk about some relationships between different elements in Q, which are distinguished by their subscripts.

We will derive a new representation of $\Upsilon(t_i)$, which serves as a critical element in our work. The idea is to divide all the possible worlds containing t_i into l groups, such that $\Upsilon(t_i)$ can be expressed by the sum of the *part PRF^{ω} values* of t_i, each of which is obtained from the possible worlds in a group. That is, each group contributes to a part of the $\Upsilon(t_i)$ value.

We now give the details. Suppose, among l generation rules $R = \{r_1, ..., r_l\}$, $t_i \in r_d$, for some $r_d \in R$. Consider a tuple set η of size l, such that $t_i \in \eta$ and each tuple in η is from a distinct generation rule in R (thus all tuples in η are from different generation rules). Let us write it in the form

$$\{t_{s_1}, t_{s_2}, ..., t_{s_{d-1}}, t_i, t_{s_{d+1}}, ..., t_{s_l}\}$$

where $t_{s_j} \in r_j$. Let us denote by Δ_i the set of all such tuple sets. Note that the tuple sets in Δ_i are *symmetric* to each other. Recall that symmetry means, assuming $\eta_1, \eta_2 \in \Delta_i$, for every possible world W_1 such that $\eta_1 \subset W_1$, there exists a possible world $W_2 = (W_1 - \eta_1) \cup \eta_2$, and vice versa. That is, W_2 can be obtained by replacing η_1 in W_1 with η_2, and vice versa.

We now divide Δ_i into l sets S_{ij}, for $0 \leq j \leq l-1$:

$$S_{ij} = \{S \in \Delta_i \mid \text{ there are exactly } j \text{ real tuples in } S \text{ s.t. for each such tuple } t, \\ score(t) > score(t_i)\} \tag{3.5}$$

We call S_{ij} the *j-rank set of t_i w.r.t. Q*. The elements in S_{ij} are "equivalent" in the sense that each of them contains *exactly j tuples whose scores are higher than $score(t_i)$*.

Suppose $\eta \in S_{ij}$. Let us define $\Upsilon_\eta(t_i)$ to be the PRF^ω value of t_i obtained from all the possible worlds in $PW(\eta)$, where $PW(\eta)$ is the set of all the possible worlds containing η, i.e.,

$$\Upsilon_\eta(t_i) = \sum_{W \in PW(\eta)} \omega(\beta_W(t_i)) \times Pr(W) \tag{3.6}$$

This is the representation of *part PRF^ω value of t_i* that we talked about above.

Recall that $Pr(\eta)$ denotes the sum of the probabilities of the possible worlds in $PW(\eta)$. We then can show

Theorem 3.3.1 *For any two tuple sets $\eta_1, \eta_2 \in S_{ij}$, we have $\frac{\Upsilon_{\eta_1}(t_i)}{Pr(\eta_1)} = \frac{\Upsilon_{\eta_2}(t_i)}{Pr(\eta_2)}$.*

That is, the ratio between $\Upsilon_\eta(t_i)$ and $Pr(\eta)$ is the same for all tuple sets $\eta \in S_{ij}$. This is an important property in our pruning theory.

Proof. Since $\eta_1, \eta_2 \in S_{ij}$, they are symmetric. We assume $W_1, W_2, ..., W_e$ are all the possible worlds which contain all the tuples in η_1. From Lemma 3.2.2, there exist possible worlds $W'_1, W'_2, ..., W'_e$ which contain all the tuples in η_2, W_v and W'_v $(1 \leq v \leq e)$ are symmetric w.r.t. η_1 and η_2, $W_v = A \cup \eta_1$ and $W'_v = A \cup \eta_2$ ($A \subseteq T$, $A \cap \eta_1 = \emptyset$, and $A \cap \eta_2 = \emptyset$). As $\eta_1, \eta_2 \in S_{ij}$, η_1 and η_2 both contain j tuples that have a higher score than t_i. So we know that t_i has the same position in W_v and W'_v. So $\beta_{W_v}(t_i) = \beta_{W'_v}(t_i)$ and $\omega(\beta_{W_v}(t_i)) = \omega(\beta_{W'_v}(t_i))$. From equation (3.6), we have

$$\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) \times Pr(W_v)$$
$$\Upsilon_{\eta_2}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W'_v}(t_i)) \times Pr(W'_v)$$

From Lemma 3.2.3, we have

$$\frac{Pr(W_v)}{Pr(W'_v)} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)}$$

So we have

$$\begin{aligned}\Upsilon_{\eta_1}(t_i) &= \sum_{v=1}^{e} \omega(\beta_{W'_v}(t_i)) \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times Pr(W'_v) \\ &= \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \sum_{v=1}^{e} \omega(\beta_{W'_v}(t_i)) \times Pr(W'_v) \\ &= \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \Upsilon_{\eta_2}(t_i)\end{aligned}$$

It follows that

$$\frac{\Upsilon_{\eta_1}(t_i)}{\Upsilon_{\eta_2}(t_i)} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)}$$

From Lemma 3.2.4, we have

$$\begin{aligned}Pr(PW(\eta_1)) &= \prod_{t_u \in \eta_1} Pr(t_u) \\ Pr(PW(\eta_2)) &= \prod_{t_u \in \eta_2} Pr(t_u)\end{aligned}$$

So we have

$$\frac{\Upsilon_{\eta_1}(t_{x_i})}{\Upsilon_{\eta_2}(t_i)} = \frac{Pr(PW(\eta_1))}{Pr(PW(\eta_2))}$$

That is

$$\frac{\Upsilon_{\eta_1}(t_i)}{Pr(PW(\eta_1))} = \frac{\Upsilon_{\eta_2}(t_i)}{Pr(PW(\eta_2))}$$

This completes the proof. □

The fact that the ratio between $\Upsilon_\eta(t_i)$ and $Pr(\eta)$ is the same for all tuple sets $\eta \in S_{ij}$ may not a total surprise, as the tuple sets in S_{ij} are in a sense equal (they form an equivalent relation). As this result is critical in our theory of pruning, let us give it a special notation: Given a non-empty S_{ij} and $\eta \in S_{ij}$, define *the PRF^ω value ratio of S_{ij}*, denoted U_{ij}, as

$$U_{ij} = \frac{\Upsilon_\eta(t_i)}{Pr(\eta)} \tag{3.7}$$

Intuitively, one can think of U_{ij} as some kind of average PRF^ω value; more precisely (unfortunately by a somewhat awkward sentence), it is the *average PRF^ω value of t_i per unit value of the probabilities of the possible worlds containing η.*

We now show an example of the concepts of S_{ij} and U_{ij}.

Example 3.3.2 *Consider Example 2.2.2 again. For convenience, let us provide the uncertain table here in Table 3.1.*

Assume $Q = \{t_2, t_5\}$. Let us use Δ_1, S_{10} and S_{11} for t_2, and similarly Δ_2, S_{20} and S_{21} for t_5. That is, these two tuples are identified by their positions in Q: the tuple t_2 is indexed by subscript 1 and t_5 by subscript 2.

	Time	Radar	Car Model	Plate No	Speed	Prob
t_1	11:45	L1	Honda	X-123	120	1.0
t_2	11:50	L2	Toyota	Y-245	130	0.7
t_3	11:35	L3	Toyota	Y-245	95	0.3
t_4	12:10	L4	Mazda	W-541	90	0.4
t_5	12:25	L5	Mazda	W-541	110	0.6
t_6	12:15	L6	Chevy	L-105	105	0.5
t_7	12:20	L7	Chevy	L-105	85	0.4

Table 3.1: An example uncertain table

For example, Δ_1 is the set of the tuple sets each of which consists of tuples from distinct generation rules involved in Q, i.e., $\Delta_1 = \{\{t_2, t_4\}, \{t_2, t_5\}\}$. In both of these two tuple sets there is no tuple with a higher score than t_2. So, $S_{10} = \{\{t_2, t_4\}, \{t_2, t_5\}\}$ and $S_{11} = \emptyset$. Similarly, we have $\Delta_2 = \{\{t_5, t_2\}, \{t_5, t_3\}\}$, and $S_{20} = \{\{t_5, t_3\}\}$ and $S_{21} = \{\{t_5, t_2\}\}$.

Assume the weight function is $\omega(i) = 5 - i$. By Theorem 3.3.1, we know

$$U_{10} = \frac{\Upsilon_{\{t_2,t_4\}}(t_2)}{Pr(PW(\{t_2, t_4\}))} = \frac{\Upsilon_{\{t_2,t_5\}}(t_2)}{Pr(PW(\{t_2, t_5\}))} = 4$$

$$U_{20} = \frac{\Upsilon_{\{t_5,t_3\}}(t_5)}{Pr(PW(\{t_5, t_3\}))} = 3$$

$$U_{21} = \frac{\Upsilon_{\{t_5,t_2\}}(t_5)}{Pr(PW(\{t_5, t_2\}))} = 2$$

□

The tuple set η in (3.6) is just one in S_{ij}. We are interested in all η in S_{ij}. Let $PW(S_{ij}) = \cup_{\eta \in S_{ij}} PW(\eta)$, and define

$$\Upsilon_{S_{ij}}(t_i) = \sum_{\eta \in S_{ij}} \Upsilon_{\eta}(t_i)$$

That is, $\Upsilon_{S_{ij}}(t_i)$ is the part of the PRF^{ω} value obtained from the possible worlds in $PW(S_{ij})$.

Now, let us define the following notation:

$$Pr(S_{ij}) = \sum_{\eta \in S_{ij}} Pr(\eta)$$

That is, $Pr(S_{ij})$ is the sum of the probabilities of all possible worlds that contain all the tuples of a tuple set in S_{ij}. We call $Pr(S_{ij})$ the probability of S_{ij}. Note that $Pr(S_{ij}) = 0$ when S_{ij} is empty.

It is easy to check that

$$\begin{aligned}\Upsilon(t_i) &= \textstyle\sum_{j=0}^{l-1} \Upsilon_{S_{ij}}(t_i) \\ &= \textstyle\sum_{j=0}^{l-1} \sum_{\eta \in S_{ij}} \Upsilon_{\eta}(t_i) \\ &= \textstyle\sum_{j=0}^{l-1} \sum_{\eta \in S_{ij}} U_{ij} \times Pr(\eta) \qquad [by\ (3.7)] \\ &= \textstyle\sum_{j=0}^{l-1} U_{ij} \times Pr(S_{ij})\end{aligned}$$

This is to say that we have arrived at a new representation of $\Upsilon(t_i)$:

$$\Upsilon(t_i) = \sum_{j=0}^{l-1} U_{ij} \times Pr(S_{ij}) \tag{3.8}$$

Equation (3.8) above is actually quite intuitive: for each j, $U_{ij} \times Pr(S_{ij})$ is the part of the PRF^{ω} value of t_i obtained from the possible worlds in $PW(S_{ij})$; then the sum of all these PRF^{ω} values of t_i is $\Upsilon(t_i)$. This is how all *part PRF^{ω} values of* t_i added together gives us the the PRF^{ω} values of t_i.

Here, $\Upsilon(t_i)$ is expressed in terms of U_{ij} and $Pr(S_{ij})$. We will compute the latter but not the former (if we do compute both, we then have computed $\Upsilon(t_i)$ - there is no pruning).

Notice that the equation

$$Pr(t_i) = \sum_{j=0}^{l-1} Pr(S_{ij}) \tag{3.9}$$

holds, since both sides equal the sum of the probabilities of all the possible worlds containing t_i. This equation will be used frequently later.

For the computation of $Pr(S_{ij})$, we can show that, for each tuple t_i, we need $O(l^2 + l\tau)$ time to compute $Pr(S_{ij})$, where τ is the maximum number of real tuples in a generation rule. Hence for q tuples, the complexity is $O(ql^2 + ql\tau)$. In real applications, τ is usually a very small number compared with the number of tuples in an uncertain database. The details of the algorithm and the complexity analysis are given below.

We adopt the idea of generating function in [29] for the computation of $Pr(S_{ij})$. Suppose tuple t_i is involved in the generation rule r_d. For each of the generation rules $r_1, r_2, ..., r_{d-1}, r_{d+1}, ..., r_l$, it divides the tuples involved in it into two parts. Some tuples have higher scores than t_i and others have lower or equal scores than t_i. We define b_{ih} as the sum of the probabilities of the tuples involved in $r_h (1 \leq h \leq l$ and $h \neq d)$ which have higher scores than t_i. Here we assume a virtual tuple has lower score than any real tuple. From the definition of b_{ih}, we know that $1 - b_{ih}$ is the sum of the probabilities of the tuples involved in r_h which have equal or lower scores than t_i. It is easy to see that we need $O(l\tau)$ time to compute all the b_{ih} for t_i. For each tuple t_i, we define a set $\theta_i = \{b_{ih}\}$ $(1 \leq h \leq l$ and $h \neq d)$.

Let us consider $\eta \in \Delta_i$. We assume $\eta = \{t_{s_1}, t_{s_2}, ..., t_{s_{d-1}}, t_i, t_{s_{d+1}}, ..., t_{s_l}\}$ where $t_{s_h} \in r_h$. We construct a generating vector $\gamma = \langle \gamma_1, \gamma_2, ..., \gamma_{d-1}, \gamma_{d+1}, ..., \gamma_l \rangle$ for η. If $score(t_{s_h}) > score(t_i)$, $\gamma_h = 1$; otherwise, $\gamma_h = 0$. Some tuple sets in Δ_i may have the same generating vector.

We assume γ is a generating vector for some tuple sets in Δ_i. We define Φ_γ as the set of all the tuple sets in Δ_i that have the generating vector γ. Let $Pr(\Phi_\gamma)$ be the sum of the probabilities of all the tuple sets in Φ_γ. It is easy to see that

$$Pr(\Phi_\gamma) = Pr(t_i) \prod_{h:\gamma_h=1} b_{ih} \prod_{h:\gamma_h=0} (1 - b_{ih})$$

We notice that all the tuple sets in Φ_γ belong to the same S_{ij}. S_{ij} is the union of some Φ_γ. Let $|\gamma|$ be the number of 1 in γ. The condition that Φ_γ belongs to S_{ij} is $|\gamma| = j$. We can compute $Pr(S_{ij})$ as follows.

$$Pr(S_{ij}) = Pr(t_i) \sum_{|\gamma|=j} \prod_{h:\gamma_h=1} b_{ih} \prod_{h:\gamma_h=0} (1 - b_{ih})$$

.

Let us see a function: $F(x) = \prod_{i=1}^{n}(a_i + b_i x)$. The coefficient of x^j in $F(x)$ is given by: $\sum_{|\beta|=j} \prod_{i:\beta_i=0} a_i \prod_{i:\beta_i=1} b_i$ where $\beta = \langle \beta_1, ..., \beta_n \rangle$ is a boolean vector.

Now consider the following generating function: $F^i(x) = Pr(t_i) \times \prod_{b \in \theta_i}(1 - b + b \times x) = \sum_{j=0}^{l-1} a_j x^j$.

We can see that the coefficient a_j of x^j in the expansion of $F^i(x)$ is $Pr(S_{ij})$. We can extend $F^i(x)$ to get c_j in $O(l^2)$ time. Therefore, for each tuple t_i, the complexity of computing $Pr(S_{ij})$ is $O(l^2 + l\tau)$.

3.3.2 A key theorem

We have gone through some fairly involved technical development above, and it is time to take a pause. Now, the reader may review the main symbols used so far and their meanings, in Table 3.2, before moving to a key theorem.

The following is a key insight in our pruning methods.

Theorem 3.3.3 *Suppose S_{ij_1}, S_{ij_2}, $S_{i_1 j}$, and $S_{i_2 j}$ are non-empty, where $(1 \leq i, i_1, i_2 \leq q)$ and $(0 \leq j, j_1, j_2 \leq l - 1)$. Then we have*

(i) if $j_1 \leq j_2$, then $U_{ij_1} \geq U_{ij_2}$;

(ii) if $score(t_{i_1}) \geq score(t_{i_2})$, then $U_{i_1 j} \geq U_{i_2 j}$.

Table of Symbols	
Q	$\{t_1, ..., t_q\}$ ($q \geq 2$), the set of real tuples considered
R	$\{r_1, ..., r_l\}$ ($l \leq q$), the set of relevant generation rules
τ	the maximum number of real tuples in a generation rule in R
Δ_i	the set of tuple sets of size l, s.t. $\forall \eta \in \Delta_i$, $t_i \in \eta$ and all tuples in η are from distinct gen. rules in R
S_{ij}	$S_{ij} \subseteq \Delta_i$ s.t. $\forall \eta \in S_{ij}$ there are exactly j tuples with scores higher than $score(t_i)$
U_{ij}	the PRF^ω value ratio of S_{ij}
$PW(t)$	the set of possible worlds (PWs) that contain t
$PW(\eta)$	the set of PWs that contain all tuples in η
$PW(S_{ij})$	the union of $PW(\eta)$, for each $\eta \in S_{ij}$
$Pr(\eta)$	the sum of the probabilities of the PWs containing all tuples in η
$Pr(S_{ij})$	the sum of the probabilities of all PWs in $PW(S_{ij})$

Table 3.2: A summary of symbols and their meanings in theoretical development

Before we present a proof, let us remark that there are good intuitions of why the claims given in theorem hold. In (i), for a tuple $t_i \in \{t_1, ..., t_q\}$, the larger the j value, the lower the U_{ij}, since higher j means more tuples "ahead" in a possible world containing η. In (ii), with the same j, the higher the score the higher the U_{ij}, as higher score contributes to higher average PRF^ω value per unit value of the probabilities of the possible world containing $\eta \in S_{ij}$.

Example 3.3.4 *Consider Example 3.3.2 again. As an example of part (i) of Theorem 3.3.3, one can see that* $U_{20} > U_{21}$*, and as an example of part (ii),* $U_{10} > U_{20}$.

We now proceed to present a proof of the theorem.

Proof. By the assumption that these S_{ij} are non-empty, we suppose $\eta_1 \in S_{ij_1}$ and $\eta_2 \in S_{ij_2}$. We assume $W_1, W_2, ..., W_e$ are all the possible worlds containing all the tuples in η_1 and $W'_1, W'_2, ..., W'_e$ are all the possible worlds containing all the tuples in η_2. Here, W_v and W'_v ($1 \leq v \leq e$) are symmetric w.r.t. η_1 and η_2. Let $W_v = A \cup \eta_1$ and $W'_v = A \cup \eta_2$ where $A \subseteq T$, $A \cap \eta_1 = \emptyset$, and $A \cap \eta_2 = \emptyset$. As $\eta_1 \in S_{ij_1}$, η_1 contains j_1 tuples that have a score higher than t_i. Since $\eta_2 \in S_{ij_2}$, η_2 contains j_2 tuples with a score higher than t_i. Since $j_1 \leq j_2$, compared with W'_v, W_v must contain a less or equal number of tuples that have a higher score than t_i. So t_i has an equal or higher position in W_v than W'_v. As we have assumed that the weight function is monotonically non-increasing, we get

$$\omega(\beta_{W_v}(t_i)) \geq \omega(\beta_{W'_v}(t_i))$$

From Lemma 3.2.3, we have

$$\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) Pr(W_v) \Upsilon_{\eta_2}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W'_v}(t_i)) Pr(W'_v)$$
$$\frac{Pr(W_v)}{Pr(W'_v)} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)}$$

from which we derive

$$\begin{aligned}
&\Upsilon_{\eta_1}(t_i) \\
&= \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times Pr(W'_v) \\
&= \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) \times Pr(W'_v) \\
&\geq \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \sum_{v=1}^{e} \omega(\beta_{W'_v}(t_i)) \times Pr(W'_v) \\
&= \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \Upsilon_{\eta_2}(t_i)
\end{aligned}$$

So

$$\frac{\Upsilon_{\eta_1}(t_i)}{\prod_{t_u \in \eta_1} Pr(t_u)} \geq \frac{\Upsilon_{\eta_2}(t_i)}{\prod_{t_u \in \eta_2} Pr(t_u)}$$

From Lemma 3.2.4, we get

$$Pr(PW(\eta_1)) = \prod_{t_u \in \eta_1} Pr(t_u)$$
$$Pr(PW(\eta_2)) = \prod_{t_u \in \eta_2} Pr(t_u)$$

which leads to

$$\frac{\Upsilon_{\eta_1}(t_i)}{Pr(PW(\eta_1))} \geq \frac{\Upsilon_{\eta_2}(t_i)}{Pr(PW(\eta_2))}$$

That is, $U_{ij_1} \geq U_{ij_2}$. □

Now we present a generalization of Theorem 3.3.3 by removing the condition that S_{ij} be non-empty. When S_{ij} is empty, it is meaningless to define U_{ij}. For technical convenience in our later exploration, we can define U_{ij} to be a real number for any empty S_{ij} so that Theorem 3.3.3 holds when any of S_{ij} is empty.

Before setting up U_{ij} for empty S_{ij}, we point out some properties of S_{ij}. From the definition of S_{ij}, we know that for a fixed i, the non-empty sets S_{ij} must be consecutive. This means that it is impossible that $S_{ih} (0 \leq h \leq l-3)$ and $S_{i(h+2)}$ are non-empty sets and $S_{i(h+1)}$ is an empty set. From the definition of S_{ij}, we also know there exist at least one non-empty S_{ij} for a fixed i. Notationally, given $t_i \in Q$, let us denote the consecutive sequence of all non-empty sets (w.r.t. t_i) by

$$S_{ij_1}, S_{i(j_1+1)}, ..., S_{ij_2}$$

where $0 \leq j_1 \leq j_2 \leq l-1$. Let us call S_{ij_1} the first non-empty S_{ij} for t_i and S_{ij_2} the last non-empty S_{ij} for t_i. We can see that $S_{i0}, S_{i1}, ..., S_{i(j_1-1)}$ and $S_{i(j_2+1)}, S_{i(j_2+2)}, ..., S_{i(l-1)}$ are all empty sets.

For any $1 \leq i_1, i_2 \leq q$, let $S_{i_1 j_1}$ be the first non-empty S_{ij} for t_{i_1}, $S_{i_1 j_2}$ be the last non-empty S_{ij} for t_{i_2}, $S_{i_2 j_3}$ be the first non-empty S_{ij} for t_{i_2} and $S_{i_2 j_4}$ be the last non-empty S_{ij} for t_{i_2}. From the definition of S_{ij}, we know that if $score(t_{i_1}) \geq score(t_{i_2})$, then $j_1 \leq j_3$ and $j_2 \leq j_4$.

Now we set up U_{ij} for empty S_{ij}. For a tuple $t_i \in Q$ $(1 \leq i \leq q)$, let S_{ij_1} be the first non-empty S_{ij} for t_i and S_{ij_2} be the last non-empty S_{ij} for t_i. For an empty S_{ij}, we set

$$U_{ij} = \begin{cases} U_{ij_1} & \text{if } j < j_1 \\ U_{ij_2} & \text{if } j > j_2 \end{cases}$$

It is easy to check that after the set up, all U_{ij} for S_{ij} (empty or non-empty) satisfy the conclusions in Theorem 3.3.3.

The argument above in fact constitutes a proof that Theorem 3.3.3 can be generalized without the assumption that the given S_{ij} be non-empty. We state this result formally below.

Theorem 3.3.5 *Given S_{ij_1}, S_{ij_2}, $S_{i_1 j}$, and $S_{i_2 j}$, where $(1 \leq i, i_1, i_2 \leq q)$ and $(0 \leq j, j_1, j_2 \leq l-1)$, we have*

(i) if $j_1 \leq j_2$, then $U_{ij_1} \geq U_{ij_2}$;

(ii) if $score(t_{i_1}) \geq score(t_{i_2})$, then $U_{i_1 j} \geq U_{i_2 j}$.

If $\omega(\beta(t)) \geq 0$ for any position $\beta(t)$ (in this case, we say that the weight function ω is *non-negative*), then according to (3.6) and (3.7), all U_{ij} for non-empty S_{ij} must be non-negative. So in this situation, we also set U_{ij} for empty S_{ij} to be non-negative and make all the U_{ij} satisfy the conditions above.

3.4 A General Upper Bound Method

Recall that, given a set of real tuples $Q = \{t_1, ..., t_q\}$, our goal is to compute an upper bound of a tuple $t \in Q$, expressed in form of (3.2). In this section we present a generic method, which is independent of the number of tuples in Q, as long as there are at least two.

We know that for each tuple $t_i \in Q$, its PRF^ω value can be expressed in form of (3.8). For this equation, we can multiply both sides with a constant c_i to get

$$c_i \Upsilon(t_i) = c_i \sum_{j=0}^{l-1} U_{ij} \times Pr(S_{ij})$$

For each tuple in Q we have such an equation, so we have q equations. Let us add them together to get

$$\sum_{i=1}^{q} c_i \Upsilon(t_i) = \sum_{i=1}^{q} \sum_{j=0}^{l-1} c_i \times U_{ij} \times Pr(S_{ij}) \tag{3.10}$$

Recall that to get the upper bound expression in (3.2), all we need is to establish the inequation in (3.1) (of course, with the assumption $c < 0$), which can be obtained from equation (3.10) if we make its right hand side non-negative. This is our focus in the following exploration.

Since $Pr(S_{ij})$ in (3.10) is to be computed, we are left with U_{ij} and a choice of the values of c_i. As we want to avoid the computation of U_{ij}, we will utilize the relation between them developed in Section 3.3.1. The idea is to transform the right hand side of (3.10) to a form in terms of U_{ij}, which is guaranteed to be non-negative without actually computing U_{ij}. We identify this form to be a summation of m terms, where $m \geq 1$. Let us write it as

$$\sum_{k=1}^{m} a_k (U_{ij} - U_{i'j'}) \tag{3.11}$$

where $a_k > 0$ is a real number, and each term involves a pair of distinct U_{ij} and $U_{i'j'}$ such that $U_{ij} \geq U_{i'j'}$, where $1 \leq i, i' \leq q$, $0 \leq j, j' \leq l-1$.

In general, there is no guarantee that there exists an assignment of c_i so that such a transformation is possible. However, whenever such an assignment exists (3.11) is guaranteed to be non-negative.

Example 3.4.1 *Assume two tuples t_1 and t_2, $score(t_1) \geq score(t_2)$, and we are interested in an upper bound of t_2. Suppose the PRF^ω values of the two tuples, expressed in the form (3.8), are*

$$\Upsilon(t_1) = 0.03U_{10} + 0.06U_{11}$$
$$\Upsilon(t_2) = 0.2U_{20} + 0.7U_{21}$$

From Theorem 3.3.5, we know that

$$U_{10} \geq U_{11},\ U_{20} \geq U_{21},\ U_{10} \geq U_{20},\ U_{11} \geq U_{21}$$

If we set $c_1 = 1$ and $c_2 = -0.1$, we will get

$$\begin{aligned}&\Upsilon(t_1) - 0.1\Upsilon(t_2)\\ &= 0.03U_{10} + 0.06U_{11} - 0.02U_{20} - 0.07U_{21}\\ &= 0.02(U_{10} - U_{20}) + 0.06(U_{11} - U_{21}) + 0.01(U_{10} - U_{21})\end{aligned}$$

The last expression is in the form (3.11), which involves three pairs of PRF^ω value ratios with coefficients $a_1 = 0.02$, $a_2 = 0.06$, and $a_3 = 0.01$. Clearly, the value of this expression

is non-negative, which guarantees $\Upsilon(t_1) - 0.1\Upsilon(t_2) \geq 0$. *So an upper bound of* t_2 *is obtained by* $\Upsilon(t_2) \leq 10\Upsilon(t_1)$.

Sometimes it may not be possible to transform the right hand side of (3.10) to (3.11), in which case we can relax the condition by allowing an extra expression, as in

$$\sum_{k=1}^{m_1} a_k(U_{ij} - U_{i'j'}) + \sum_{k=1}^{m_2} b_k U_{ij} \tag{3.12}$$

where $m_1 \geq 0$ and $m_2 \geq 1$. The first summation is similar to (3.11). The second involves a subset of PRF^{ω} value ratios, for $1 \leq i \leq q$ and $0 \leq j \leq l-1$, with coefficients $b_k > 0$. In this situation, if $\omega(\beta(t)) \geq 0$ for any position $\beta(t)$, all U_{ij} must be non-negative. Then (3.12) must be non-negative. It follows that the right hand side of (3.10) must be non-negative.

Example 3.4.2 *Assume two tuples* t_1 *and* t_2 *with* $score(t_1) \geq score(t_2)$, *and suppose*

$$\Upsilon(t_1) = 0.02U_{10} + 0.08U_{11}$$
$$\Upsilon(t_2) = 0.2U_{20} + 0.6U_{21}$$

Let c_1 *and* c_2 *be the coefficients of* t_1 *and* t_2 *respectively, as in equation (3.10). Assume the weight function is non-negative. From Theorem 3.3.5, we know that* $U_{10} \geq U_{11}, U_{20} \geq U_{21}, U_{10} \geq U_{20}$, *and* $U_{11} \geq U_{21}$. *It can be shown that there does not exist an assignment that leads to a transformation to (3.11). However, a transformation to (3.12) is possible. If we set* $c_1 = 1$ *and* $c_2 = -0.1$, *we will get*

$$\begin{aligned}&\Upsilon(t_1) - 0.1\Upsilon(t_2)\\ &= 0.02U_{10} + 0.08U_{11} - 0.02U_{20} - 0.06U_{21}\\ &= 0.02(U_{10} - U_{20}) + 0.06(U_{11} - U_{21}) + 0.02U_{11}\end{aligned}$$

The second line above is in the form of the right hand side of (3.10), and the last expression is in the form (3.12), whereas the term $0.02U_{11}$ *corresponds to the second summation. Thus, we conclude* $\Upsilon(t_1) - 0.1\Upsilon(t_2) \geq 0$.

Theorem 3.4.3 *Let* $Q = \{t_1, ..., t_q\}$. *Assume* $t \in Q$ *and there exists a tuple* $s \in Q$ *such that* $s \neq t$ *and* $score(s) \geq score(t)$. *Then, there exists at least one assignment* θ *of* c_i *such that the right hand side of (3.10) can be transformed to an expression in form of (3.11), and if not, to an expression in form of (3.12).*

Proof. For notational convenience, let s and t in the theorem be also named as t_a and t_b, respectively. Because $score(t_a) \geq score(t_b)$, we know $U_{aj} \geq U_{bj}$. We construct an

assignment θ of c_i w.r.t. Q as follows: set $c_i(i \neq a, b) = 0$ in θ. Then the right hand side of (3.10) can be written as:

$$\sum_{j=0}^{l-1} c_a \times Pr(S_{aj}) \times U_{aj} + c_b \times Pr(S_{bj}) \times U_{bj}$$

Here, $c_a > 0$ and $c_b < 0$. Clearly, we just need to set c_a large enough and the absolute value of c_b small enough, so that $\sum_{j=0}^{l-1} c_a \times Pr(S_{aj}) \times U_{aj} + c_b \times Pr(S_{bj}) \times U_{bj}$ can be transformed to an expression in the form (3.11), and if not, to an expression in the form (3.12). □

Note that if there does not exist a tuple $s \in Q$ such that $score(s) \geq score(t)$, then it is not difficult to show that no assignment can lead to a successful transformation to either (3.11) or (3.12). This is why we need at least one tuple ahead of tuple t, as a reference.

Notice also that for the expression in (3.12) to be non-negative, the underlying weight function shall be non-negative.

In the sequel, given any q (real) tuples $Q = \{t_1, ..., t_q\}$, we say that *an assignment θ (of c_i w.r.t. Q) induces an upper bound of t*, where $t \in Q$, if the right hand side of (3.10), with θ substituted, can be transformed to an expression either in the form of (3.11) or in the form of (3.12). We note that the notion of "an assignment inducing an upper bound of t" is important in the rest of this chapter. Essentially, such an assignment θ makes the upper bound expression in (3.2) hold, and that is why it induces an upper bound of t.

Questions arise. Under the method of transformation from (3.10) to either (3.11) or (3.12), is there a notion of optimal upper bound for a tuple t w.r.t. a given Q, where $t \in Q$? This question seems hard for a Q with any size. But when the size of Q is 2, we can give a positive answer.

Theorem 3.4.4 *Let T be an uncertain table, $Q = \{t', t\}$ be a set of tuples from T. The upper bound u of t, induced by any assignment w.r.t. Q, satisfies $u \geq \frac{Pr(t)}{Pr(t')}\Upsilon(t')$.*

Proof. If $score(t') < score(t)$, it is easy to know that no assignment w.r.t. Q can induce an upper bound of t. For an assignment w.r.t. Q, if the coefficient of t' is not greater than 0, it is also easy to see that this assignment cannot induce an upper bound of t. So if an assignment w.r.t. Q induces an upper bound, we then must have $score(t') \geq score(t)$ and the coefficient of t' is greater than 0.

Let ϕ be any assignment w.r.t. Q which induces an upper bound u of t. Let the coefficients of t', t in ϕ be c_1, c_2, respectively. We know that $c_1 > 0$ and $c_2 < 0$. From equation

(3.10), we have

$$c_1\Upsilon(t') + c_2\Upsilon(t)$$
$$= \sum_{j=0}^{l-1} c_1 Pr(S_{1j})U_{1j} + \sum_{j=0}^{l-1} c_2 Pr(S_{2j})U_{2j}$$

To make the transformation from the right hand side of the equation above to (3.11) or (3.12), the sum of the positive coefficients of U_{ij} must not be smaller than the sum of the absolute value of the negative coefficients of U_{ij}. Notice that $\sum_{j=0}^{l-1} Pr(S_{1j}) = Pr(t')$ and $\sum_{j=0}^{l-1} Pr(S_{2j}) = Pr(t)$. So we get

$$c_1 Pr(t') \geq -c_2 Pr(t)$$

and this is

$$c_1 \geq -c_2 \frac{Pr(t)}{Pr(t')}$$

We know the upper bound u of t induced from ϕ is

$$u = \frac{c_1\Upsilon(t')}{-c_2}$$

So we can get

$$u \geq \frac{Pr(t)}{Pr(t')}\Upsilon(t')$$

□

Theorem 3.4.4 shows that, if an assignment w.r.t. $Q = \{t', t\}$ can induce an upper bound u of t, then this upper bound u is not lower than $\frac{Pr(t)}{Pr(t')}\Upsilon(t')$.

In other words, $\frac{Pr(t)}{Pr(t')}\Upsilon(t')$ is the lowest possible upper bound of t, induced by any assignment w.r.t. $Q = \{t', t\}$. In general, it may or may not be an upper bound of t. When it is, let us call it the *optimal upper bound of* t *w.r.t.* Q. We stress that the notion of optimality here is relative, i.e., it is the lowest upper bound one can possibly get within the framework of transforming the right hand side of (3.10) to either (3.11) or (3.12). Note also that an optimal bound is defined w.r.t. Q. For different Q's, optimal upper bounds may well be different.

Intuitively, one can think of the practical scenario in which some tuples with higher score than t have been retrieved and we want to know the upper bound of t. Let the retrieved tuples with higher scores than t be P. When using two tuple relation (this means the size of Q is 2), from Theorem 3.4.4, we know the upper bound of t is not lower than $\frac{Pr(t)}{Pr(t')}\Upsilon(t')$ for any $t' \in P$. Let $t_{lowest} \in P$ such that $\frac{\Upsilon(t_{lowest})}{Pr(t_{lowest})} \leq \frac{\Upsilon(t')}{Pr(t')}$, for any $t' \in P$. Then the upper bound of t is not lower than $\frac{Pr(t)}{Pr(t_{lowest})}\Upsilon(t_{lowest})$ when using two tuple relation to get the upper bound of T from tuples in P.

If we want to improve an upper bound, we may consider additional tuples. This is easy to understand as with more tuples involved, we have more information to use. So we may get a lower upper bound. Let us see an example.

Example 3.4.5 *Consider Example 2.2.2, where t_6 and t_4 are from different generation rules and $score(t_6) > score(t_4)$. Assume*

$$\Upsilon(t_6) = Pr(S_{10})U_{10} + Pr(S_{11})U_{11} = 0.2U_{10} + 0.3U_{11}$$
$$\Upsilon(t_4) = Pr(S_{20})U_{20} + Pr(S_{21})U_{21} = 0.2U_{20} + 0.2U_{21}$$

Consider an upper bound of t_4. By Theorem 3.4.4, we know that the upper bound of t_4 induced from any assignment w.r.t. $\{t_6, t_4\}$ is no smaller than $\frac{Pr(t_4)}{Pr(t_6)}\Upsilon(t_6)$. But there does not exist an assignment w.r.t. $\{t_6, t_4\}$, which induces this upper bound. Now let us introduce t_3. We then have

$$\begin{aligned}\Upsilon(t_6) &= Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11} + Pr(S_{12}) \times U_{12}\\ &= 0.06U_{10} + 0.23U_{11} + 0.21U_{12}\\ \Upsilon(t_4) &= Pr(S_{31}) \times U_{31} + Pr(S_{32}) \times U_{32}\\ &= 0.2U_{31} + 0.2U_{32}\end{aligned}$$

If we set $c_1 = \frac{1}{Pr(t_6)} = 2$ and $c_2 = -\frac{1}{Pr(t_4)} = -2.5$, we will get

$$\begin{aligned}&\tfrac{\Upsilon(t_6)}{Pr(t_6)} - \tfrac{\Upsilon(t_4)}{Pr(t_4)}\\ &= 0.04(U_{10} - U_{31}) + 0.46(U_{11} - U_{31}) + 0.08(U_{10} - U_{32}) + 0.42(U_{12} - U_{32})\\ &\geq 0\end{aligned}$$

So we can get an upper bound of t_4 as $\frac{Pr(t_4)}{Pr(t_6)}\Upsilon(t_6)$. This upper bound is induced from an assignment w.r.t. $\{t_6, t_4, t_3\}$.

The above example shows that there are cases where by using an additional tuple we can get a better upper bound.

3.5 Summary

In this chapter, we study inter-connections of PRF^ω values of tuples, and find that symmetric tuple sets possess some interesting properties. Using these properties, we derive a new representation of PRF^ω value, in the form:

$$\Upsilon(t_i) = \sum_{j=0}^{l-1} U_{ij} \times Pr(S_{ij})$$

This result allows us to locate the part of the computation of $\Upsilon(t_i)$ that may be pruned, namely the computations of U_{ij}'s. We further find that there exist some interesting relations between different U_{ij}'s. Then we utilize these relations to derive a general method of

pruning, in the form of a scheme. We show some properties of this general method. In addition, we study the possibility of using more tuples in order to improve an upper bound. To our knowledge, the theoretical insights revealed in this chapter are new, and results enhance our understanding of the inter-relationships among tuples in the possible world context.

Chapter 4

Deriving Practical Pruning Methods

In the last chapter, we have developed an abstract schema to determine an upper bound of the PRF^{ω} value of a tuple. Based on this scheme, in this chapter, we will derive some practical methods for pruning.

This chapter is organized as follows. We introduce the concrete computational methods of upper bounds of PRF^{ω} values of tuples in Section 4.1. Section 4.2 gives the details of a combined algorithm and its complexity analysis. Section 4.3 compares our pruning with the pruning methods in the literature. Section 4.4 comments on the assumptions in our work.

4.1 Deriving Practical Methods

The method given in the last chapter is only a scheme, which does not tell us how to choose the coefficients c_i so that the described transformations are possible. In this section, we show how to instantiate this scheme to generate practical upper bound methods.

Briefly, if two tuples t_1 and t_2 ($score(t_1) \geq score(t_2)$) are from the same generation rule, there is a choice of c_i such that the optimal upper bound of t_2 w.r.t. $\{t_1, t_2\}$ is guaranteed. In the case that two tuples are from different generation rules, if some condition is satisfied, the optimal bound is guaranteed, and if it is not we provide a conservative estimate. That is, an upper bound is generated in any case. We continue this exercise to three tuples for possibly improving non-optimal upper bounds. It is then clear that theoretically speaking, this process can continue for any n tuples.

4.1.1 Two tuples from the same generation rule

Assume two real tuples t_1 and t_2 from the same generation rule, with $score(t_1) \geq score(t_2)$. As they are involved in the same generation rule, $l = 1$. So according to (3.8), we have

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10} \quad \Upsilon(t_2) = Pr(S_{20}) \times U_{20}$$

where, as $l = 1$, $Pr(t_1) = Pr(S_{10})$ and $Pr(t_2) = Pr(S_{20})$ (cf. equation (3.9)). It's also clear that $U_{10} \geq U_{20}$. Then, by setting $c_1 = \frac{1}{Pr(t_1)}$ and $c_2 = -\frac{1}{Pr(t_2)}$ the right hand side of equation (3.10) can be transformed to

$$\frac{\Upsilon(t_1)}{Pr(t_1)} - \frac{\Upsilon(t_2)}{Pr(t_2)} = U_{10} - U_{20} \geq 0$$

which is an instance of (3.11). Then we have $\Upsilon(t_2) \leq \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$. This is to say that if we know the PRF^{ω} value of t_1, we can compute the optimal upper bound of t_2 w.r.t. $\{t_1, t_2\}$.

The above actually constitutes a proof for the following theorem.

Theorem 4.1.1 *Let T be an uncertain table, and t_1 and t_2 be two real tuples in T that are involved in the same generation rule, with $score(t_1) \geq score(t_2)$. Then, we have $\Upsilon(t_2) \leq \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$.*

Note that the time complexity to compute the upper bound of t_2 in the theorem is $O(1)$.

4.1.2 Two tuples from different generation rules

Assume two real tuples t_1 and t_2 from different generation rules and $score(t_1) \geq score(t_2)$. As they belong to different generation rules, $l = 2$. So, according to (3.8), we have

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11}$$
$$\Upsilon(t_2) = Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21}$$

Let us set $c_1 = \frac{1}{Pr(t_1)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. We can show

Theorem 4.1.2 *Let T be an uncertain table, and t_1 and t_2 be two real tuples in T belonging to different generation rules. Assume $score(t_1) \geq score(t_2)$. If $\frac{Pr(S_{10})}{Pr(t_1)} \geq \frac{Pr(S_{20})}{Pr(t_2)}$, then $\Upsilon(t_2) \leq \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$.*

Proof. We know that $U_{10} \geq U_{20}, U_{11} \geq U_{21}, U_{10} \geq U_{11}$. By the definition of U_{ij}, it follows immediately that $U_{10} \geq U_{21}$. By definition, we also have

$$Pr(t_1) = Pr(S_{10}) + Pr(S_{11})$$
$$Pr(t_2) = Pr(S_{20}) + Pr(S_{21})$$

So we can get

$$\frac{Pr(S_{10})}{Pr(t_1)} - \frac{Pr(S_{20})}{Pr(t_2)} = \frac{Pr(S_{21})}{Pr(t_2)} - \frac{Pr(S_{11})}{Pr(t_1)}$$

Let us set $c_1 = \frac{1}{Pr(t_1)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. Then, we have

$$\begin{aligned}
&\tfrac{1}{Pr(t_1)} \times \Upsilon(t_1) - \tfrac{1}{Pr(t_2)} \times \Upsilon(t_2) \\
&= \tfrac{Pr(S_{10})}{Pr(t_1)} \times U_{10} + \tfrac{Pr(S_{11})}{Pr(t_1)} \times U_{11} - \tfrac{Pr(S_{20})}{Pr(t_2)} \times U_{20} - \tfrac{Pr(S_{21})}{Pr(t_2)} \times U_{21} \\
&= \tfrac{Pr(S_{20})}{Pr(t_2)} \times (U_{10} - U_{20}) + \tfrac{Pr(S_{11})}{Pr(t_1)} \times (U_{11} - U_{21}) + (\tfrac{Pr(S_{10})}{Pr(t_1)} - \tfrac{Pr(S_{20})}{Pr(t_2)}) \times U_{10} \\
&-(\tfrac{Pr(S_{21})}{Pr(t_2)} - \tfrac{Pr(S_{11})}{Pr(t_1)}) \times U_{21} \\
&= \tfrac{Pr(S_{20})}{Pr(t_2)} \times (U_{10} - U_{20}) + \tfrac{Pr(S_{11})}{Pr(t_1)} \times (U_{11} - U_{21}) + (\tfrac{Pr(S_{10})}{Pr(t_1)} - \tfrac{Pr(S_{20})}{Pr(t_2)}) \times (U_{10} - U_{21})
\end{aligned}$$

Clearly, if $\frac{Pr(S_{10})}{Pr(t_1)} \geq \frac{Pr(S_{20})}{Pr(t_2)}$, then

$$\frac{1}{Pr(t_1)} \times \Upsilon(t_1) \geq \frac{1}{Pr(t_2)} \times \Upsilon(t_2)$$

So we get

$$\Upsilon(t_2) \leq \frac{Pr(t_2)}{Pr(t_1)} \Upsilon(t_1)$$

□

For some insights, in the proof of Theorem 4.1.2, if two tuples t_1 and t_2 are from different generation rules and the condition $\frac{Pr(S_{10})}{Pr(t_1)} \geq \frac{Pr(S_{20})}{Pr(t_2)}$ is satisfied, we can choose c_1 and c_2 so that the right hand side of (3.10) can be transformed to (3.11), hence we get the optimal upper bound of t_2 w.r.t. $\{t_1, t_2\}$.

In Theorem 4.1.2, we need to compute $Pr(S_{ij})$. As discussed in Section 3.3.1, the time complexity of computing $Pr(S_{ij})$ is $O(ql^2 + ql\tau)$.[1] Here $q = 2$ and $l = 2$. So the time complexity of computing $Pr(S_{ij})$ is $O(\tau)$, so is the cost of computing the upper bound of t_2.

Example 4.1.3 *Consider Example 2.2.2. Suppose the weight function is $\omega(i) = 5 - i$. Tuples t_2 and t_5 are involved in different generation rules.*

$$\Upsilon(t_2) = Pr(S_{10})U_{10} = 0.7U_{10}$$
$$\Upsilon(t_5) = Pr(S_{20})U_{20} + Pr(S_{21})U_{21} = 0.18U_{20} + 0.42U_{20}$$

Since

$$\frac{Pr(S_{10})}{Pr(t_2)} = 1 > \frac{Pr(S_{20})}{Pr(t_5)} = 0.3$$

we have

$$\frac{\Upsilon(t_2)}{Pr(t_2)} = 4 \geq \frac{\Upsilon(t_5)}{Pr(t_5)} = 2.3$$

In Theorem 4.1.2, the condition $\frac{Pr(S_{10})}{Pr(t_1)} \geq \frac{Pr(S_{20})}{Pr(t_2)}$ must be satisfied. If this condition is not satisfied, question arises as whether there is a reasonable way to estimate an upper bound of t_2. The answer is yes. For this, let us set $c_1 = \frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. We can show

Theorem 4.1.4 *Let T be an uncertain table, and t_1 and t_2 be two real tuples in T from different generation rules, with $score(t_1) \geq score(t_2)$. If $\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$ and the weight function is non-negative, we have $\Upsilon(t_2) \leq \frac{Pr(S_{20})}{Pr(S_{10})} \Upsilon(t_1)$.*

[1] Recall that τ is the maximum number of real tuples in a generation rule.

Proof. Because

$$\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$$

we get

$$\frac{Pr(S_{10})}{Pr(S_{10}) + Pr(S_{11})} < \frac{Pr(S_{20})}{Pr(S_{20}) + Pr(S_{21})}$$

From this, we have

$$Pr(S_{20}) \times Pr(S_{11}) - Pr(S_{10}) \times Pr(S_{21}) > 0$$

Here we assume the weight function $\omega(\beta(t)) \geq 0$ for any position so that any U_{ij} will be equal to or greater than 0.

Since $score(t_1) \geq score(t_2)$ and the tuples are involved in different generation rules, $Pr(S_{10})$ must be greater than 0. Let us set $c_1 = \frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. We then have

$$\begin{aligned}
&\frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)} \times \Upsilon(t_1) - \frac{1}{Pr(t_2)} \times \Upsilon(t_2) \\
&= \frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)}(Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11}) \\
&- \frac{1}{Pr(t_2)}(Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21}) \\
&= \frac{Pr(S_{20})}{Pr(t_2)} \times (U_{10} - U_{20}) + \frac{Pr(S_{21})}{Pr(t_2)} \times (U_{11} - U_{21}) \\
&+ \frac{Pr(S_{20}) \times Pr(S_{11}) - Pr(S_{10}) \times Pr(S_{21})}{Pr(S_{10}) \times Pr(t_2)} \times U_{11} \\
&\geq 0
\end{aligned}$$

So when $\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$ and the weight function is always greater than or equal to 0, we have $\frac{Pr(S_{20})}{Pr(S_{10})} \times \Upsilon(t_1) \geq \Upsilon(t_2)$. □

Theorem 4.1.4 says that if the condition in Theorem 4.1.2 is not satisfied (i.e., its negation $\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$ is satisfied), we still can get an upper bound of t_2, which is $\Upsilon(t_1)$ multiplying the factor $\frac{Pr(S_{20})}{Pr(S_{10})}$. Obviously, this upper bound is higher $\frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$.

The cost of computing the upper bound above is $O(\tau)$.

Example 4.1.5 *Consider Example 2.2.2. Assume the weight function $\omega(i) = 5 - i$. Tuples t_6 and t_4 are involved in different generation rules and $score(t_6) > score(t_4)$.*

$$\begin{aligned}
\Upsilon(t_6) &= Pr(S_{10})U_{10} + Pr(S_{11})U_{11} = 0.2U_{10} + 0.3U_{11} \\
\Upsilon(t_4) &= Pr(S_{20})U_{20} + Pr(S_{21})U_{21} = 0.2U_{20} + 0.2U_{21}
\end{aligned}$$

Since

$$\frac{Pr(S_{10})}{Pr(t_6)} = 0.4 < \frac{Pr(S_{20})}{Pr(t_4)} = 0.5$$

we have

$$\frac{Pr(S_{20})}{Pr(S_{10})}\Upsilon(t_6) = 0.85 \geq \Upsilon(t_4) = 0.6$$

4.1.3 Three tuples

When the condition in Theorem 4.1.2 is not satisfied, it is possible to improve the non-optimal upper bound given in Theorem 4.1.4.

Suppose t_1, t_2, and t_3 are three real tuples from three different generation rules, with $score(t_1) \geq score(t_2) \geq score(t_3)$. From expression (3.8), we get

$$\begin{aligned}
\Upsilon(t_1) &= Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11} + Pr(S_{12}) \times U_{12} \\
\Upsilon(t_2) &= Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21} + Pr(S_{22}) \times U_{22} \\
\Upsilon(t_3) &= Pr(S_{30}) \times U_{30} + Pr(S_{31}) \times U_{31} + Pr(S_{32}) \times U_{32}
\end{aligned}$$

Theorem 4.1.6 *Let T be an uncertain table, and t_1, t_2, and t_3 be three real tuples in T from three different generation rules, with $score(t_1) \geq score(t_2) \geq score(t_3)$. Let $a, b \in \{1, 2, 3\}$ and $a < b$. If $\frac{Pr(S_{a0})}{Pr(t_a)} \geq \frac{Pr(S_{b0})}{Pr(t_b)}$ and $\frac{Pr(S_{a2})}{Pr(t_a)} \leq \frac{Pr(S_{b2})}{Pr(t_b)}$, then we have $\frac{1}{Pr(t_a)} \times \Upsilon(t_a) \geq \frac{1}{Pr(t_b)} \times \Upsilon(t_b)$*

Proof. Let d be the remaining number in $\{1, 2, 3\}$ except a and b. We set $c_a = \frac{1}{Pr(t_a)}$, $c_b = -\frac{1}{Pr(t_b)}$, $c_d = 0$. We have

$$\frac{Pr(S_{a0})}{Pr(t_a)} + \frac{Pr(S_{a1})}{Pr(t_a)} + \frac{Pr(S_{a2})}{Pr(t_a)} = 1 = \frac{Pr(S_{b0})}{Pr(t_b)} + \frac{Pr(S_{b1})}{Pr(t_b)} + \frac{Pr(S_{b2})}{Pr(t_b)}$$

If $\frac{Pr(S_{a1})}{Pr(t_a)} \geq \frac{Pr(S_{b1})}{Pr(t_b)}$, then

$$\begin{aligned}
&\tfrac{1}{Pr(t_a)} \times \Upsilon(t_a) - \tfrac{1}{Pr(t_b)} \times \Upsilon(t_b) \\
&= \tfrac{Pr(S_{b0})}{Pr(t_b)}(U_{a0} - U_{b0}) + (\tfrac{Pr(S_{a0})}{Pr(t_a)} - \tfrac{Pr(S_{b0})}{Pr(t_b)})U_{a0} + \tfrac{Pr(S_{b1})}{Pr(t_b)}(U_{a1} - U_{b1}) \\
&+(\tfrac{Pr(S_{a1})}{Pr(t_a)} - \tfrac{Pr(S_{b1})}{Pr(t_b)})U_{a1} + \tfrac{Pr(S_{a2})}{Pr(t_a)}(U_{a2} - U_{b2}) - (\tfrac{Pr(S_{b2})}{Pr(t_b)} - \tfrac{Pr(S_{a2})}{Pr(t_a)})U_{b2}
\end{aligned}$$

We then get

$$(\frac{Pr(S_{a0})}{Pr(t_a)} - \frac{Pr(S_{b0})}{Pr(t_b)}) + (\frac{Pr(S_{a1})}{Pr(t_a)} - \frac{Pr(S_{b1})}{Pr(t_b)}) = \frac{Pr(S_{b2})}{Pr(t_b)} - \frac{Pr(S_{a2})}{Pr(t_a)}$$

It follows that

$$\begin{aligned}
&\tfrac{1}{Pr(t_a)} \times \Upsilon(t_a) - \tfrac{1}{Pr(t_b)} \times \Upsilon(t_b) \\
&= \tfrac{Pr(S_{b0})}{Pr(t_b)}(U_{a0} - U_{b0}) + \tfrac{Pr(S_{b1})}{Pr(t_b)}(U_{a1} - U_{b1}) + \tfrac{Pr(S_{a2})}{Pr(t_a)}(U_{a2} - U_{b2}) \\
&+(\tfrac{Pr(S_{a0})}{Pr(t_a)} - \tfrac{Pr(S_{b0})}{Pr(t_b)})(U_{a0} - U_{b2}) + (\tfrac{Pr(S_{a1})}{Pr(t_a)} - \tfrac{Pr(S_{b1})}{Pr(t_b)})(U_{a1} - U_{b2}) \\
&\geq 0
\end{aligned}$$

If $\frac{Pr(S_{a1})}{Pr(t_a)} < \frac{Pr(S_{b1})}{Pr(t_b)}$, we can carry out a similar transformation as above.

$$\begin{aligned}
&\tfrac{1}{Pr(t_a)} \times \Upsilon(t_a) - \tfrac{1}{Pr(t_b)} \times \Upsilon(t_b) \\
&= \tfrac{Pr(S_{b0})}{Pr(t_b)}(U_{a0} - U_{b0}) + \tfrac{Pr(S_{a1})}{Pr(t_a)}(U_{a1} - U_{b1}) + \tfrac{Pr(S_{a2})}{Pr(t_a)}(U_{a2} - U_{b2}) \\
&+(\tfrac{Pr(S_{b1})}{Pr(t_b)} - \tfrac{Pr(S_{a1})}{Pr(t_a)})(U_{a0} - U_{b1}) + (\tfrac{Pr(S_{b2})}{Pr(t_b)} - \tfrac{Pr(S_{a2})}{Pr(t_a)})(U_{a0} - U_{b2}) \\
&\geq 0
\end{aligned}$$

So we always have $\frac{1}{Pr(t_a)} \times \Upsilon(t_a) - \frac{1}{Pr(t_b)} \times \Upsilon(t_b) \geq 0$. That is $\frac{1}{Pr(t_a)} \times \Upsilon(t_a) \geq \frac{1}{Pr(t_b)} \times \Upsilon(t_b)$. □

The cost of computing the upper bound above is $O(\tau)$.

Example 4.1.7 *Consider Example 3.4.5, after t_3 is introduced, we have*

$$\frac{Pr(S_{10})}{Pr(t_6)} = 0.12 \geq \frac{Pr(S_{30})}{Pr(t_4)} = 0$$
$$\frac{Pr(S_{12})}{Pr(t_6)} = 0.42 \leq \frac{Pr(S_{32})}{Pr(t_4)} = 0.5$$

It follows

$$\frac{\Upsilon(t_6)}{Pr(t_6)} = 1.7 \geq \frac{\Upsilon(t_4)}{Pr(t_4)} = 1.5$$

4.2 Algorithm and Complexity

Earlier in Section 3.1, we sketched a combined algorithm for the computation of top-k tuples for PRF^ω with pruning. We now fill more details on this combined algorithm.

After the first k tuples, we retrieve a new tuple t_{new}. To compute the upper bound of t_{new}, we maintain a tuple called t_{lowest}, which is among the retrieved tuples whose PRF^ω values have been computed. The tuple t_{lowest} must be the one that has the lowest ratio between its PRF^ω value and its membership probability among all retrieved tuples, i.e., $\frac{\Upsilon(t_{lowest})}{Pr(t_{lowest})} \leq \frac{\Upsilon(t')}{Pr(t')}$, for any retrieved tuple t' such that $\Upsilon(t')$ is computed. If t_{new} is involved in the same generation rule with t_{lowest}, we get the upper bound of t_{new} as $\frac{Pr(t_{new})}{Pr(t_{lowest})}\Upsilon(t_{lowest})$ from Theorem 4.1.1.

If t_{new} and t_{lowest} are from different generation rules, we check whether the condition in Theorem 4.1.2 is satisfied. If it is, we get the upper bound of t_{new} as $\frac{Pr(t_{new})}{Pr(t_{lowest})}\Upsilon(t_{lowest})$. Otherwise Theorem 4.1.4 provides an upper bound of t_{new}. As discussed in Section 3.4, when using two tuple relation, $\frac{Pr(t_{new})}{Pr(t_{lowest})}\Upsilon(t_{lowest})$ is the lowest upper bound of t_{new} we can get from the retrieved tuples whose PRF^ω values have been computed.

Example 4.2.1 *To show the procedure of the combined algorithm with pruning, we give an example. We use the uncertain table in Table 2.3. We assume the weight function is $\omega(i) = 5 - i$. We want to find top-1 tuple. We assume all the tuples are sorted in a descending order of their scores and then are retrieved one by one. Then all the tuples are retrieved in the following order: $t_2, t_1, t_5, t_6, t_3, t_4, t_7$.*

For the first retrieved tuple t_2, we compute its PRF^ω value and it is 2.8. For the second retrieved tuple t_1, t_{lowest} is t_2. t_1 and t_2 are from different generation rules. We have

$$\Upsilon(t_2) = Pr(S_{10})U_{10} + Pr(S_{11})U_{11} = 0.7U_{10}$$
$$\Upsilon(t_1) = Pr(S_{20})U_{20} + Pr(S_{21})U_{21} = 0.3U_{20} + 0.7U_{21}$$

Because $\frac{Pr(S_{10})}{Pr(t_2)} = 1 > \frac{Pr(S_{20})}{Pr(t_1)} = 0.3$*, the condition in Theorem 4.1.2 is satisfied. We get the upper bound of* t_1 *as* $\frac{Pr(t_1)}{Pr(t_2)}\Upsilon(t_2) = 4$*. Now* t_2 *is the tuple with largest* PRF^ω *value among all the retrieved tuples (not including* t_1*). We compare the upper bound of* t_1 *with* $\Upsilon(t_2)$ *and find it is greater than* $\Upsilon(t_2)$*. So* t_1 *can not be pruned. We need to compute its* PRF^ω *value and it is 3.3. Because the ratio between* t_1*'s* PRF^ω *value and its membership probability is lower than the ratio of* t_2*, we update* t_{lowese} *to* t_1*. And* t_1 *is the tuple with the largest* PRF^ω *value among all the retrieved tuples now.*

For the third retrieved tuple t_5*, we compare it with* t_{lowest} *(now is* t_1*).* t_5 *and* t_1 *are from different generation rules. We have*

$$\Upsilon(t_1) = Pr(S_{10})U_{10} + Pr(S_{11})U_{11} = U_{10}$$
$$\Upsilon(t_5) = Pr(S_{20})U_{20} + Pr(S_{21})U_{21} = 0.6U_{21}$$

Because $\frac{Pr(S_{10})}{Pr(t_1)} = 1 > \frac{Pr(S_{20})}{Pr(t_5)} = 0$*, the condition in Theorem 4.1.2 is satisfied. We get the upper bound of* t_5 *as* $\frac{Pr(t_5)}{Pr(t_1)}\Upsilon(t_1) = 1.98$*. It is smaller than* $\Upsilon(t_1)$*, so* t_5 *can be pruned.* t_{lowest} *does not need to be updated because the ratio between* t_1*'s* PRF^ω *value and its membership probability is lower than the ratio of* t_5*. And* t_1 *is still the tuple with the largest* PRF^ω *value among all the retrieved tuples.*

Similarly, we can process t_6, t_3, t_4, t_7 *one by one. All these tuples can be pruned. So the top-1 tuple is* t_1*.*

Recall from Section 4.1.2 that the time cost to compute the upper bound of a tuple is $O(\tau)$, where τ is the maximum number of real tuples in a generation rule.

Assume there are n tuples in T, and t_i is the i-th tuple, ordered by tuples' scores. In [29], the time cost of computing the PRF^ω value of t_i is $O(i\ log^2(i))$. The total time cost to find top-k tuples is $O(n^2 log^2 n)$, where n is the number of tuples in T.

We can see that the cost of pruning for a tuple is much smaller than that of computing its PRF^ω value. When we retrieve the tuples whose positions are not high, these tuples are very likely to be pruned. This is also verified in our experiments.

In the worst case in which no tuple is pruned, the time complexity of our (combined) algorithm is $O(n^2 log^2 n + n\tau)$. As the maximum value of τ is n, the time complexity of the combined algorithm remains the same as the one without pruning, namely $O(n^2 log^2 n)$. So pruning does not reduce or increase the complexity of the given algorithm.

4.3 Theoretical Comparison

In [28], the concept of *probability threshold top-k query* (PT-k query) is introduced. It defines the top-k probability of a tuple t to be the sum of the probabilities of all the possible worlds in which t is one of the top-k tuples. Given a probability threshold p, the answer set of a PT-k query is the set of all tuples whose top-k probability values are at least p. Here let us consider the answer set of a PT-k query to be the set of k tuples with highest top-k probabilities.

In [28], some pruning techniques are proposed. PT-k query answer can be thought of as a special case of PRF^{ω} by writing the weight function of the PT-k query answer as follows:

$$\omega(i) = \begin{cases} 1 & \text{if } i \leq k \\ 0 & \text{if } i > k \end{cases}$$

Therefore, our methods are applicable. The pruning theorems in [28] are given as follows, where p is the threshold.

Theorem 4.3.1 *$Pr^k(t) \leq Pr(t)$. Moreover, for an independent tuple t, if $Pr^k(t) < p$, then (1) for any independent tuple t' such that $score(t) \geq score(t')$ and $Pr(t') \leq Pr(t)$. $Pr^k(t') < p$; and (2) for any multi-tuple generation rule r such that t is ranked higher than all the tuples in r and $Pr(r) \leq Pr(t)$, $Pr^k(t'') < p$ for any $t'' \in r$.*

Theorem 4.3.2 *For tuples t and t' in the same multi-tuple rule r, if $score(t) \geq score(t')$, $Pr(t) \geq Pr(t')$, and $Pr^k(t) < p$, then $Pr^k(t') < p$.*

Here $Pr^k(t)$ is the top-k probability of a tuple t. One can think of $Pr^k(t)$ as $\Upsilon(t)$ of this paper by changing the threshold p to the k-th highest top-k probability found so far in applications. These pruning theorems can be derived from our pruning theorems but not conversely. For conclusion (1) of Theorem 4.3.1, we can derive it from our Theorem 4.1.2. We can identify t and t' to be t_1 and t_2 in Theorem 4.1.2. From the theorem, we can get $Pr^k(t') \leq \frac{Pr(t')}{Pr(t)} Pr^k(t)$. Because $Pr(t') \leq Pr(t)$, we can get $Pr^k(t') \leq Pr^k(t) < p$. For conclusion (2) of Theorem 4.3.1, we can identify t and t'' to be t_1 and t_2 in Theorem 4.1.2. From the theorem, we can get $Pr^k(t'') \leq \frac{Pr(t'')}{Pr(t)} Pr^k(t)$. Because $Pr(t'') < Pr(r) \leq Pr(t)$, $Pr^k(t'') < Pr^k(t) < p$. We can get Theorem 4.3.2 from Theorem 4.1.1. With t and t' identified to be t_1 and t_2 in Theorem 4.1.1, we can get $Pr^k(t') \leq \frac{Pr(t')}{Pr(t)} Pr^k(t)$. Because $Pr(t) \geq Pr(t')$, we have $Pr^k(t') \leq Pr^k(t) < p$.

Actually, our upper bound theorems are much stronger than those in [28]. For the condition of conclusion (1) in Theorem 4.3.1 (the same condition appears in Theorem 4.3.2), our

theorems weaken the condition from $Pr^k(t) < p$ to $Pr^k(t) < \frac{Pr(t)}{Pr(t')} \times p$. Since $\frac{Pr(t)}{Pr(t')} > 1$, our condition covers more cases. For conclusion (2) in Theorem 4.3.1, our theorems allow a tuple in a multi-tuple generation rule to compare directly with an independent tuple. This means that we can weaken the condition from $Pr(R) \leq Pr(t)$ to $Pr(t'') < Pr(t)$, for any $t'' \in R$. We can also weaken the condition from $Pr^k(t) < p$ to $Pr^k(t) < \frac{Pr(t)}{Pr(t'')} \times p$. The new condition covers more cases.

Generally speaking, the theorems in [28] must satisfy two conditions for any two given tuples t and t': (1) $score(t) > score(t')$, and (2) $Pr(t) \geq Pr(t')$. But our theorems only require the first condition. As a result, our theorems cover more cases. Even when both conditions are satisfied, as we have already shown, our theorems can prune better than the theorems in [28].

In [28] a theorem about *global constraint* is given:

Theorem 4.3.3 *Let A be a set of tuples whose top-k probability values have been computed. If* $\sum_{t \in A} Pr^k(t) > k - p$*, then for every tuple* $t' \notin A, Pr^k(t') < p$.

Based on this global constraint theorem, if the top-k probabilities of some tuples have been computed and the sum of the top-k probabilities of these tuples is greater than $k - p$, then the top-k probability of any tuple which has not been computed is smaller than p. Similarly as before, we can change the threshold p to the k-th highest top-k probability found so far in applications.

But the global constraint cannot be used for PRF^ω or PRF^e, due to the difference between the definition of PRF^ω and the *PT-k query answer*.

4.4 Discussion

A question arises. If a tuple t_1 has higher score and membership probability than another tuple t_2, must t_1 have higher PRF^ω value than t_2? The answer is no. Here we give a counter example. We consider an uncertain table which has 4 tuples: t_1, t_2, t_3, t_4. The scores of these tuples satisfy the following conditions: $score(t_4) > score(t_1) > score(t_2) > score(t_3)$. The member probabilities of t_1, t_2, t_3, t_4 are $0.3, 0.25, 0.7, 0.75$ respectively. Tuples t_1, t_3 are involved in one generation rule and tuples t_2,t_4 are involved in another generation rule. The weight function $\omega(i) = 2 - i$. We can see that t_1 has higher score and membership probability than t_2. But the PRF^ω value of t_1 is 0.375 and the PRF^ω value of t_2 is 0.425. So t_1 has lower PRF^ω value than t_2.

There are three assumptions in our work for pruning. The first is that the weight function is independent of tuples. Technically, this assumption can be removed if $\omega(t, i)$ is restricted to $\omega(score(t), i)$ (i.e., anything that depends on t depends on $score(t)$), and $\omega(score(t), i)$ is non-negative and monotonically non-decreasing w.r.t. $score(t)$ (i.e., higher the score, higher the weight). Another assumption in our work is that $\omega(t, i)$ is monotonically non-increasing w.r.t. i. This is a reasonable assumption, as in most real applications a smaller i (hence a higher position) is clearly more important than larger i's. The third assumption is that the weight function is non-negative. We will look into whether this can be removed in the future.

Chapter 5

Pruning for PRF^e

In this chapter we turn our attention to the problem of computing top-k tuples for PRF^e. This is an important topic as the overall complexity of computing top-k tuples for PRF^e is linear,[1] so it is potentially useful in dealing with extreme large databases. Its importance is also reflected in the fact that PRF^e can be used as an approximation of PRF^ω [29]. In this chapter, we propose an entirely new method for pruning for this context. The key result is an early termination condition, followed by a description of an algorithm on how this condition is actually used.

5.1 Early Termination

Here, we show a special property of PRF^e for pruning, which can terminate the top-k computation earlier.

It is known that the PRF^e value of a tuple can be determined in constant time [29]. Thus, the method of pruning given in previous sections may not be worthwhile for PRF^e. Here we study a special property of PRF^e, as presented in the lemma below.

Lemma 5.1.1 *Let T be an uncertain table and $Q = \{t_1, ..., t_q\}$ a set of tuples from T. Let the weight function be PRF^e and $\omega(i) = \alpha^i$. We assume α is a real number and $0 < \alpha < 1$. For any S_{ij} and $S_{i(j+1)}$ $(0 \leq j \leq l-2, l \geq 2)$ which are non-empty, we have $U_{i(j+1)} = \alpha \times U_{ij}$.*

Proof. Suppose the tuples in Q are involved in l $(l \leq q)$ generation rules in R. Assume $score(t_i) \geq score(t_{i+1})$ $(1 \leq i \leq q-1)$.

Following the proof for part (i) of Theorem 3.3.3, let $\eta_1 \in S_{ij}$ and $\eta_2 \in S_{i(j+1)}$. We assume $W_1, W_2, ..., W_e$ are all the possible worlds containing all the tuples in η_1 and

[1] If sorting tuples is considered part of the process, then the complexity becomes $O(n\ log\ n)$.

$W'_1, W'_2, ..., W'_e$ are all the possible worlds containing all the tuples in η_2. Assume W_v and W'_v $(1 \leq v \leq e)$ are symmetric w.r.t. η_1 and η_2, and let $W_v = A \cup \eta_1$ and $W'_v = A \cup \eta_2$. Because $\eta_1 \in S_{ij}$, there are j tuples in η_1 having a higher score than t_i. Since $\eta_2 \in S_{i(j+1)}$, there are $j+1$ tuples in η_2 having a higher score than t_i. So we know that W'_v contains one more tuple than W_v with a higher score than t_i. We thus have

$$\begin{array}{l} \beta_{W_v}(t_i) = \beta_{W'_v}(t_i) - 1 \\ \omega(\beta_{W_v}(t_i)) = \frac{1}{\alpha} \times \omega(\beta_{W'_v}(t_i)) \end{array}$$

Because

$$\begin{array}{l} \Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) Pr(W_v) \\ \Upsilon_{\eta_2}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W'_v}(t_i)) Pr(W'_v) \\ \frac{Pr(W_v)}{Pr(W'_v)} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \end{array}$$

we have

$$\begin{array}{l} \Upsilon_{\eta_1}(t_i) \\ = \sum_{v=1}^{e} \frac{1}{\alpha} \times \omega(\beta_{W'_v}(t_i)) \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times Pr(W'_v) \\ = \frac{1}{\alpha} \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \Upsilon_{\eta_2}(t_i) \end{array}$$

It follows that

$$\frac{\Upsilon_{\eta_1}(t_i)}{\prod_{t_u \in \eta_1} Pr(t_u)} = \frac{1}{\alpha} \times \frac{\Upsilon_{\eta_2}(t_i)}{\prod_{t_u \in \eta_2} Pr(t_u)}$$

Since

$$\begin{array}{l} Pr(PW(\eta_1)) = \prod_{t_u \in \eta_1} Pr(t_u) \\ Pr(PW(\eta_2)) = \prod_{t_u \in \eta_2} Pr(t_u) \end{array}$$

we get

$$\frac{\Upsilon_{\eta_1}(t_i)}{Pr(PW(\eta_1))} = \frac{1}{\alpha} \times \frac{\Upsilon_{\eta_2}(t_i)}{Pr(PW(\eta_2))}$$

This is

$$U_{i(j+1)} = \alpha \times U_{ij}$$

□

Recall that in the definition of U_{ij} (see equation (3.7)), we assume that the corresponding S_{ij} are non-empty. Now, for technical convenience we would like to deal with empty S_{ij} as well. We did this for PRF^{ω} in Subsection 3.3.2. Due to the special property of PRF^{e} (as stated in Lemma 5.1.1), we need to deal with empty S_{ij} differently from PRF^{ω}.

Recall that for a fixed i, non-empty S_{ij} appears consecutively. For a tuple $t_i \in Q$, we assume S_{ij_1} is the first non-empty S_{ij} for t_i and S_{ij_2} is the last non-empty S_{ij} for t_i. Now, for empty S_{ij} we define the corresponding U_{ij} as follows.

$$U_{ij} = \begin{cases} U_{ij_1} \times \alpha^{j-j_1} & \text{if } j < j_1 \\ U_{ij_2} \times \alpha^{j-j_2} & \text{if } j > j_2 \end{cases}$$

After the setup of U_{ij} for empty S_{ij}, it is easy to check that for any S_{ij} and $S_{i(j+1)}$ $(0 \leq j \leq l-2, l \geq 2)$, we have $U_{i(j+1)} = \alpha \times U_{ij}$. So we can relax the assumption that given S_{ij} be non-empty in Lemma 5.1.1. We state this formally below.

Lemma 5.1.2 *Let T be an uncertain table and $Q = \{t_1, ..., t_q\}$ a set of tuples from T. Suppose the ranking function is PRF^e and the weight function is $\omega(i) = \alpha^i$. Assume α is a real number and $0 < \alpha < 1$. For any S_{ij} and $S_{i(j+1)}$ $(0 \leq j \leq l-2, l \geq 2)$, we have $U_{i(j+1)} = \alpha \times U_{ij}$.*

From Theorem 3.3.3, we know that for any $S_{i_1 j}$ and $S_{i_2 j}$ $(1 \leq i_1, i_2 \leq q, 0 \leq j \leq l-1)$ which are non-empty, if $score(t_{i_1}) \geq score(t_{i_2})$, then $U_{i_1 j} \geq U_{i_2 j}$. It is easy to see that after the setup of U_{ij} for empty S_{ij} as above, for any $S_{i_1 j}$ and $S_{i_2 j}$ $(1 \leq i_1, i_2 \leq q, 0 \leq j \leq l-1)$, if $score(t_{i_1}) \geq score(t_{i_2})$, then $U_{i_1 j} \geq U_{i_2 j}$. So after the setup of U_{ij} for empty S_{ij} for PRF^e, Theorem 3.3.5 still holds.

Theorem 5.1.3 *Let T be an uncertain table and $t_1 \in T$. Suppose the ranking function is PRF^e and $\omega(i) = \alpha^i$, where α is a real number and $0 < \alpha < 1$. Then for any tuple $t_2 \in T$ such that $score(t_2) \leq score(t_1)$, $\Upsilon(t_2) \leq \frac{1}{\alpha} \times \frac{1}{Pr(t_1)} \Upsilon(t_1)$.*

Proof. There are two cases: the tuples t_1 and t_2 are are involved two different generation rules or both are in the same generation rule.

First, let us assume t_1 and t_2 are involved in two different generation rules. By the new representation of Υ in (3.8), we have

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11}$$
$$\Upsilon(t_2) = Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21}$$

By Lemma 5.1.2, we have $U_{11} = \alpha \times U_{10}$ and $U_{21} = \alpha \times U_{20}$. From equation (3.9), we have

$$Pr(t_1) = Pr(S_{10}) + Pr(S_{11}) \tag{5.1}$$

$$Pr(t_2) = Pr(S_{20}) + Pr(S_{21}) \tag{5.2}$$

It follows that

$$\frac{\Upsilon(t_1)}{Pr(t_1)} = \frac{Pr(S_{10})}{Pr(t_1)} \times U_{10} + (1 - \frac{Pr(S_{10})}{Pr(t_1)}) \times U_{10} \times \alpha$$
$$\frac{\Upsilon(t_2)}{Pr(t_2)} = \frac{Pr(S_{20})}{Pr(t_2)} \times U_{20} + (1 - \frac{Pr(S_{20})}{Pr(t_2)}) \times U_{20} \times \alpha$$

Now, let us divide the two equations above, i.e.,

$$\frac{\frac{\Upsilon(t_1)}{Pr(t_1)}}{\frac{\Upsilon(t_2)}{Pr(t_2)}} = \frac{U_{10}((1-\alpha)\frac{Pr(S_{10})}{Pr(t_1)} + \alpha)}{U_{20}((1-\alpha)\frac{Pr(S_{20})}{Pr(t_2)} + \alpha)}$$

From equation (5.1), it is clear that we have $0 \leq \frac{Pr(S_{10})}{Pr(t_1)} \leq 1$. Similarly, from (5.2) we get $0 \leq \frac{Pr(S_{20})}{Pr(t_2)} \leq 1$. Thus we can get

$$\alpha \leq (1-\alpha)\frac{Pr(S_{10})}{Pr(t_1)} + \alpha \leq 1$$

and

$$\alpha \leq (1-\alpha)\frac{Pr(S_{20})}{Pr(t_2)} + \alpha \leq 1$$

So we get

$$\frac{(1-\alpha)\frac{Pr(S_{10})}{Pr(t_1)} + \alpha}{(1-\alpha)\frac{Pr(S_{20})}{Pr(t_2)} + \alpha} \geq \alpha$$

Because $U_{10} \geq U_{20}$, we get

$$\frac{\frac{\Upsilon(t_1)}{Pr(t_1)}}{\frac{\Upsilon(t_2)}{Pr(t_2)}} \geq \alpha$$

It follows that

$$\Upsilon(t_2) \leq \frac{1}{\alpha} \times \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$$

Because $0 \leq Pr(t_2) \leq 1$, we have

$$\Upsilon(t_2) \leq \frac{1}{\alpha} \times \frac{1}{Pr(t_1)}\Upsilon(t_1)$$

which is the conclusion in the theorem.

Now we consider the second case, namely t_1 and t_2 are involved in the same generation rule. In this case, by the new representation of $\Upsilon(t)$, we have

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10}$$
$$\Upsilon(t_2) = Pr(S_{20}) \times U_{20}$$

We know that $Pr(t_1) = Pr(S_{10})$ and $Pr(t_2) = Pr(S_{20})$. So we have

$$\frac{\Upsilon(t_1)}{Pr(t_1)} = U_{10}$$
$$\frac{\Upsilon(t_2)}{Pr(t_2)} = U_{20}$$

Because $U_{10} \geq U_{20}$, we divide the two equations above to get

$$\frac{\frac{\Upsilon(t_1)}{Pr(t_1)}}{\frac{\Upsilon(t_2)}{Pr(t_2)}} = \frac{U_{10}}{U_{20}} \geq 1 \geq \alpha$$

So we have

$$\Upsilon(t_2) \leq \frac{1}{\alpha} \times \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$$

Because $0 \leq Pr(t_2) \leq 1$, we have

$$\Upsilon(t_2) \leq \frac{1}{\alpha} \times \frac{1}{Pr(t_1)}\Upsilon(t_1)$$

From the derivation above, we know that when t_1 and t_2 involves in one or two generation rules, we always have

$$\Upsilon(t_2) \leq \frac{1}{\alpha} \times \frac{1}{Pr(t_1)} \Upsilon(t_1)$$

This completes the proof. □

Theorem 5.1.3 tells us that, if we know $PRF^e(t)$, then $\frac{1}{\alpha} \times \frac{1}{Pr(t)} \Upsilon(t)$ is an upper bound of all tuples whose scores are smaller than $score(t)$. Following the top-k algorithm given in [29], where tuples in an uncertain table are sorted according to their scores, when this value is lower than the kth largest PRF^e value found so far the computation can safely terminate.

5.2 Algorithm and Complexity for PRF^e

Theorem 5.1.3 says that if we know the PRF^e value of a tuple t, then we know the upper bounds of all the tuples whose scores are smaller than that of t. This can be used as an early termination condition.

For PRF^e, similar to the method for pruning for PRF^ω, we also sort the tuples in a descending order of their scores and retrieve them one by one. We compute the PRF^e values for the first k tuples. We also maintain L_k and t_{lowest}. From the $(k+1)$-th tuple, before we retrieve the tuple t, we test whether $\frac{1}{\alpha} \times \frac{1}{Pr(t_{lowest})} \Upsilon(t_{lowest})$ is smaller than the k-th largest PRF^e value found so far. We call this test *an early termination condition test*. If the condition is satisfied, it means any of the tuples which have not been retrieved is impossible to be a top-k tuple. So we stop the computation and return L_k as the top-k tuples. Otherwise we retrieve the next tuple t and compute PRF^e value of t. Then we update L_k and t_{lowest}, and move to the next tuple.

Suppose there are n tuples in T. In [29], assuming all the tuples are sorted according to their scores, the time complexity to find the top-k tuples is $O(n)$.

Our algorithm does not improve the complexity, but since tuples are ordered in descending order, the termination condition will be satisfied after some tuples are retrieved, so the computation is likely to terminate early.

Chapter 6

Experiments

We have done extensive experiments to evaluate the effectiveness of our pruning techniques introduced in Chapter 3 and 4. We combine our methods with top-k algorithms for PRF^ω and PRF^e, respectively. The procedure described in Section 3.1 and Section 4.2 uses two tuples for pruning, which was employed in our experiments. We also extended the procedure for pruning with three tuples and will comment on our experience towards the end.

All the experiments were run on a quad-core 2.3GHZ PC with 16GB RAM, running Linux operating system. The algorithms are implemented in Microsoft Visual C++ V6.0.

Below, we use "computed tuples" to mean the number of tuples whose PRF^ω or PRF^e values are actually computed in running top-k algorithms in [29] combined with pruning.

In our experiments, we implemented the PRF^ω algorithm given in [29] whose complexity is $O(n^3)$. In [29], another algorithm with complexity $O(n^2 log^2 n)$ is also provided. If we use the latter, the "computed tuples" remains the same but the gaps in running times will be smaller. For PRF^e, we implemented the algorithm given in [29], combined with the early termination condition given in Chapter 5.

6.1 Data Sets and Weight Functions

Normal data sets:

Synthetic data sets are generated for our experiments. Such a data set contains some tuples and some multi-tuple generation rules. The number of tuples involved in each multi-tuple generation rule follows the normal distribution, so does the probabilities of independent tuples and multi-tuple generation rules. To generate different data sets, we vary the mean of the membership probabilities of the independent tuples and the size of a multi-tuple generation rule.

Special data sets:

Intuitively, pruning may not be very effective on data sets like the following: the scores of the tuples are in an descending order and their membership probabilities are in an ascending order. Although these data sets do not seem to be typical in the real world, it can be used to illustrate an interesting phenomenon of performance changes from variants of these data sets, which are obtained as follows: we update such a data set by swapping the membership probabilities of different tuples such that the membership probabilities of the resulting tuples are not strictly in an ascending order. To get different data sets, we vary the ratio between the number of swapping tuples and the number of all tuples.

Real data set:

A real data set is generated from International Ice Patrol (IIP) Iceberg Sighting Databases (*http://nsidc.org/data/g00807.html*). Each tuple contains the number of days drifted (score) and a confidence value (membership probability). The generated data set consists of 4232 tuples and 826 multi-tuple generation rules. This data set is also used in [28]. We get the data set from *http://www.cs.sfu.ca/∼jpei/Software/PTKLib.rar* (the authors of [28] provided this). In [28], the method to generate this data set is described. Here we briefly introduce this method. In the iceberg sighting database, each sighting record contains the sighting date, sighting location (latitude and longitude), number of days drifted, etc. Each tuple is associated with a confidence level according to the source of sighting. These sources include: R/V (radar and visual), VIS (visual only), RAD (radar only), SAT-L(low earth orbit satellite), SAT-M(medium earth orbit satellite) and SAT-H (high earth orbit satellite). To quantify the confidence, the authors of [28] assign confidence values 0.8, 0.7, 0.6, 0.5, 0.4 and 0.3 to the above six confidence levels respectively. For the sightings with the same time stamp, if the sighting locations are very close (the differences in latitude and longitude are both smaller than 0.01), they are considered referring to the same iceberg and only one of them is correct. All tuples involved in such a sighting form a multi-tuple rule. We write a multi-tuple generation rule r as $t_1 \oplus ... \oplus t_m$. The probability of r $Pr(r)$ is set to the maximum confidence among the membership probabilities of tuples in the rule. The membership probability of a tuple in the generation rule r is adjusted to

$$Pr(t_i) = \frac{conf(t_i)}{\sum_{1 \leq j \leq m} conf(t_j)} Pr(r) \quad (1 \leq i \leq m)$$

Here $conf(t_i)$ is the the confidence value of t_i.

Weight function:

For the experiments for PRF^ω, we tried different weight functions $\omega(i)$: randomly generated weight functions (RGWFs), the weight function $\omega(i) = n - i$, where n is the number of tuples in an uncertain database and i is the position of a tuple in a possible world, and the weight function from the *PT-k query answer* [28]. All these functions are related only to i and monotonically non-increasing.

RGWFs are generated as follows: we generate n random numbers and sort them into a descending order; the number in the position i of this order is the weight value of $\omega(i)$.

In our experiments, we found the performance of our pruning methods for RGWFs similar to that using the weight function $\omega(i) = n - i$. So we only give the results for RGWFs and the one based on *PT-k query answer*.

6.2 Results

All the figures are put at the end of this chapter. Each graph (except Figure 6.7 and 6.8, which are oriented for some special purposes) in this chapter shows computed tuples or running times (on y-axis) over one of the three parameters (on x-axis): the expected membership probabilities of independent tuples, the expected number of tuples in a generation rule, and different values of k. Note that the first two parameters result in different data sets. In all our experiments, the top-k tuples found by the combined algorithm with pruning is the same as the top-k tuples found by the algorithm without pruning.

Figures 6.1, 6.2 and 6.3 show the computed tuples for normal data sets with pruning, for a RGWF $random1$ and *PT-k query answer*. The size of the database is 100,000 tuples. We set $k = 50$ in both Figure 6.1 and 6.2. In Figure 6.1, we see that the computed tuples is between 50 and 400. Compared with the size of the dataset, with pruning we only need to compute the PRF^ω values for a very small portion of the database. This is an impressive improvement. With the increase of the expected membership probability of the independent tuples in a data set, less tuples need be computed. In Figure 6.2, we see that the computed tuples is between 100 and 400. In Figure 6.3, we vary the value of k from 50 to 250. We see that only a small number of tuples are computed for their PRF^ω values. Most tuples are pruned.

Figures 6.4, 6.5 and 6.6 show the running times for normal data sets. Here we also test for 2 weight functions: a RGWF $random2$ and PT-k query answer. The size of the data set is 2000 tuples. We set $k = 50$ for $random2$ and $k = 150$ for PT-k query answer in both

Figure 6.4 and 6.5. The improvement is orders of magnitude. The improvement is similar for larger data sets (we scaled the size up to 10,000 tuples and tested some selectively).

Figures 6.7 and 6.8 uses the special data sets with different ratios between the number of swapping tuples and the number of all tuples, and different weight functions. The size of the data sets is 2000. We set $k = 50$ for $random2$ and $k = 150$ for the PT-k query answer. In the two figures (Figure 6.7 is about computed tuples and Figure 6.8 is about running times), we see that when the swapping ratio is low, pruning produces little performance gains. With the increase of the swapping ratio, more tuples are pruned and the running times reduced substantially.

Figures 6.9 and 6.10 show the computed tuples and running times for a RGWF random3 and PT-k query answer for the real data set. We vary k from 150 to 350. The improvement is similar to the ones with the synthetic normal data sets.

Comparison with pruning of [28]:

It can be shown that the main pruning theorems of [28] are special cases of our pruning theorems (see Section 4.3). We compare with the simple pruning technique used in [28]. The test data sets are normal data sets, with size 2000. We used two weight functions: a RGWF $random2$ and the PT-k query answer. Figure 6.11, 6.12 and 6.13 compare the computed tuples, while Figure 6.14, 6.15 and 6.16 are on the running times. It can be noticed that substantial improvement is generated.

Early termination for PRF^e:

Figures 6.17-6.22 show the experimental results for the early termination condition for PRF^e. We use the early termination condition given in Chapter 5 to terminate a computation when the condition is satisfied. The test data sets are normal data sets, with the size being 1,000,000 tuples. We set $\alpha = 0.95$. When k=50000 or higher, the computation terminates right after the first k tuples are retrieved. Running times are shortened to one fifth to one twentieth of the ones without pruning. With smaller k's, the number of retrieved tuples could be higher than k, but not substantially. This shows that Theorem 5.1.3 yields a highly effective pruning method.

Experiments with 3-tuples:

Theorem 3.4.4 states that when the optimal upper bound is not obtained using two tuples, we may get it by introducing a third tuple. Suppose this is the case for two tuples t_{lowest} and t_{new}, and we introduce a third tuple. It can be shown that this new tuple must be involved in a generation rule which contains some tuples whose score is between

$score(t_{lowest})$ and $score(t_{new})$ (which therefore must already been retrieved). Otherwise, it will not help improve the already computed upper bound. Thus, in our experiments a third tuple is randomly chosen satisfying the above condition. However, our experiments with the four weight functions for the normal and special synthetic data sets didn't show performance gains, as the number of computed tuples and running times are very close to those by the pruning method with two tuples. One observation is that, in most cases, the optimal upper bound has already been reached using two tuples.

Real value vs. upper bound:

We also conducted experiments to show the closeness of the PRF^{ω} values of tuples and their computed upper bounds using the method of this thesis, for the real data set. We set $k = 50$ and use the weight function $\omega(i) = n - i$. To plot the graph (Figure 6.23), we pick one tuple from every 50 tuples (i.e., $51st$, $101st$,..., and so on; the upper bounds of the first k tuples need not be computed).

From Figure 6.23, we can see that for early retrieved tuples, the upper bounds are very close to their PRF^{ω} values. This shows that our pruning method finds very good upper bounds at the beginning. For later retrieved tuples, the distance between the upper bounds and the PRF^{ω} values becomes larger. Question arises as why the distance is larger while most later tuples are still pruned?

A short answer is that later tuples are easier to be pruned. In more details, there are three factors to be considered. First, when more tuples are retrieved, the lowest PRF^{ω} value in the current top-k tuples becomes larger. If an upper bound of a tuple is lower than this value, the tuple is pruned. Second, Figure 6.23 shows the trend of the PRF^{ω} values generally decreasing with the decrease of scores. So the later tuples in this case generally have lower PRF^{ω} values. This is why the distance becomes larger. Third, for later tuples, because most of the tuples retrieved before have been pruned and their PRF^{ω} values are not computed, the maintained t_{lowest}'s ratio between its PRF^{ω} value and membership probability is distanced from the possible lowest ratio among all retrieved tuples. So the computed upper bounds are not as tight. Observe that this is to say that when pruning is effective, the distance of the real values and computed upper bounds tends to become larger, and when pruning is ineffective, more PRF^{ω} values are computed and better t_{lowest}'s ratios are generated so to make later pruning more likely. It is an interesting self-adapting process.

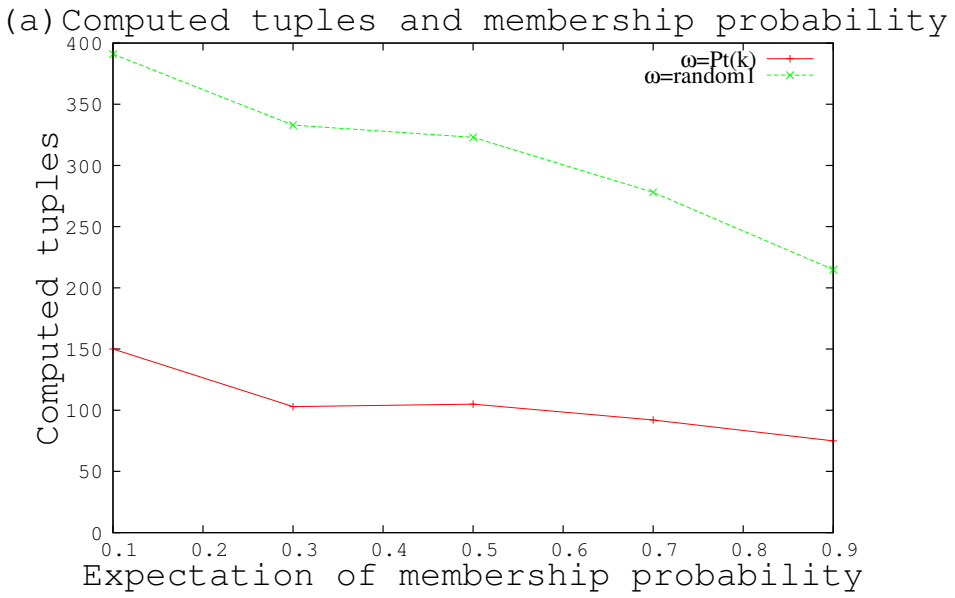

Figure 6.1: Computed tuples for PRF^{ω} on normal data sets: membership probabilities of independent tuples

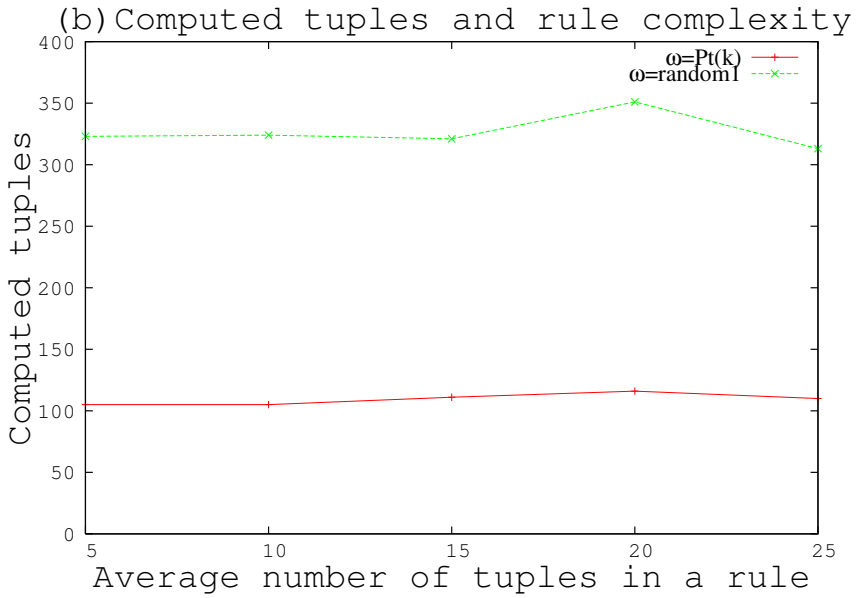

Figure 6.2: Computed tuples for PRF^{ω} on normal data sets: sizes of multi-tuple generation rules

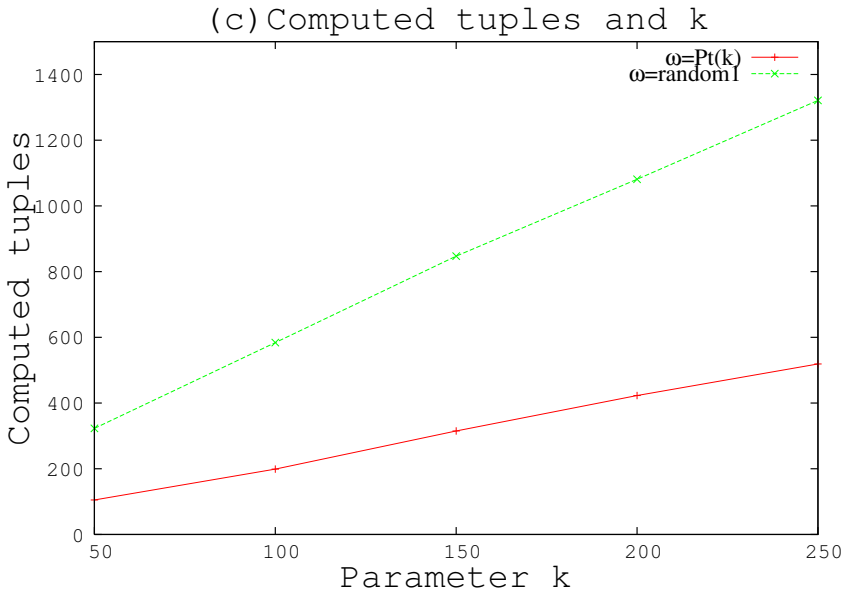

Figure 6.3: Computed tuples for PRF^{ω} on normal data sets: parameter k

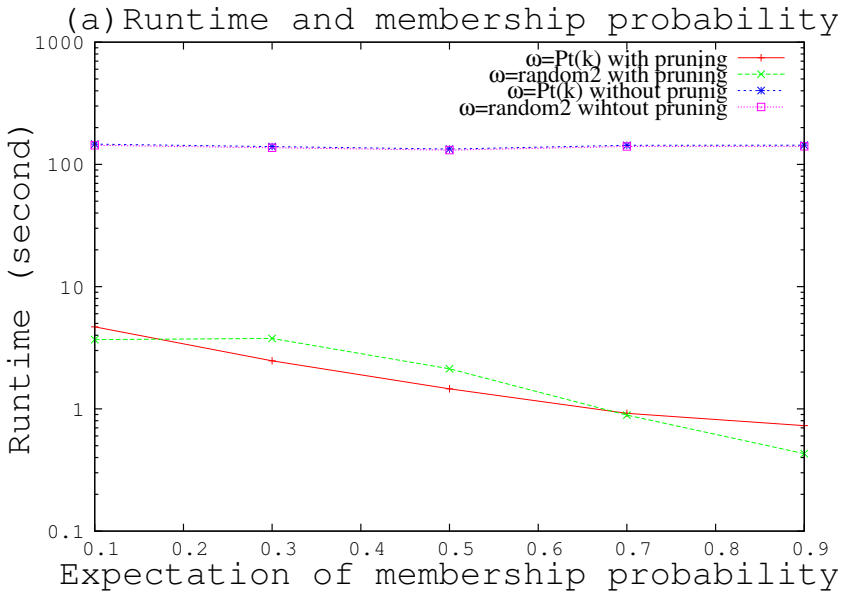

Figure 6.4: Running times for PRF^{ω} on normal data sets: membership probabilities of independent tuples

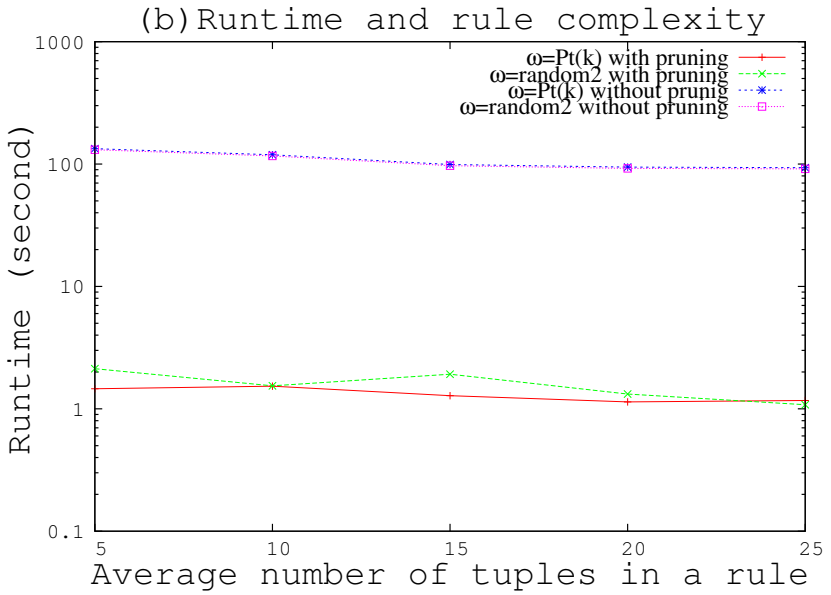

Figure 6.5: Running times for PRF^{ω} on normal data sets: sizes of multi-tuple generations

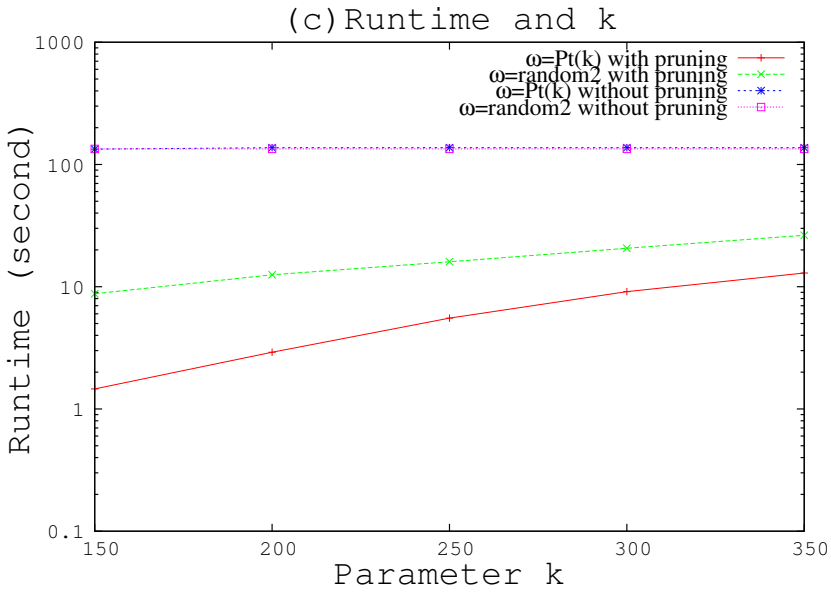

Figure 6.6: Running times for PRF^{ω} on normal data sets: parameter k

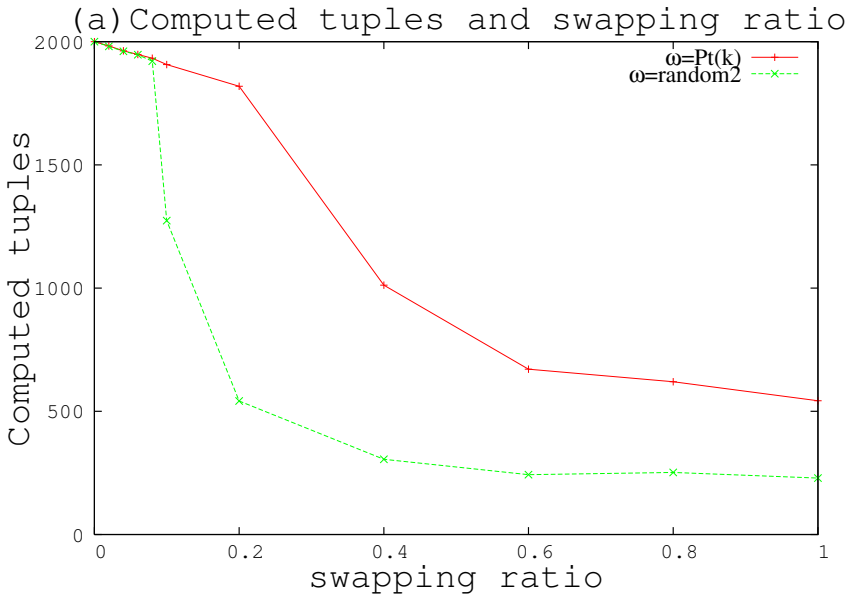

Figure 6.7: Comparison on special data sets: computed tuples

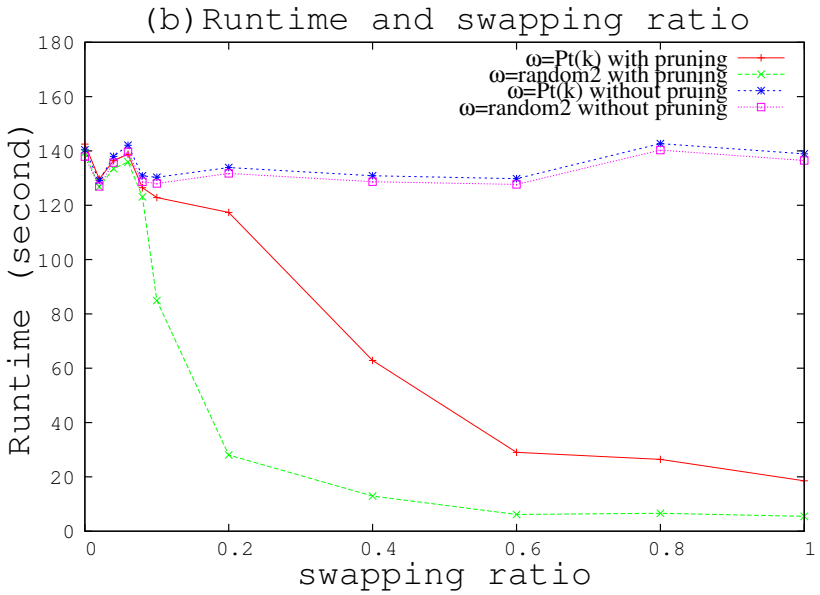

Figure 6.8: Comparison on special data sets: running times

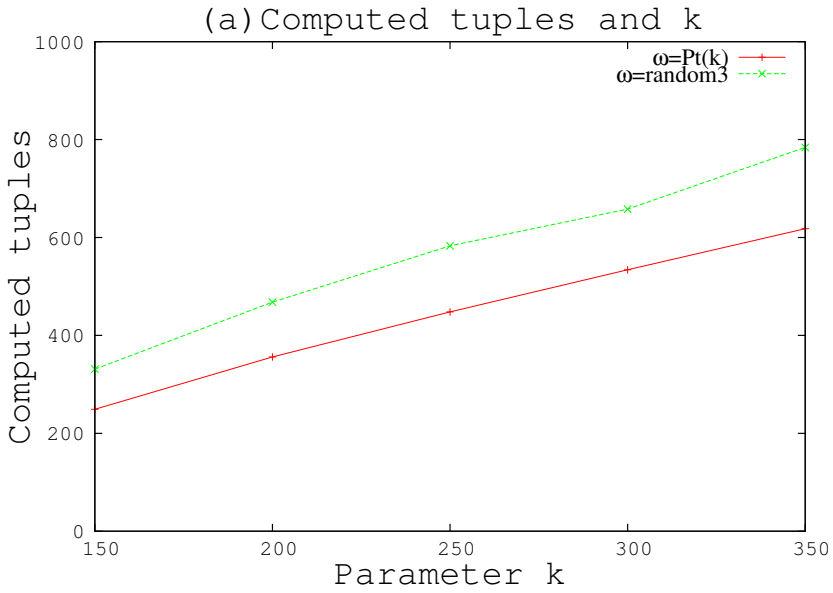

Figure 6.9: Computed tuples for PRF^{ω} on real data set

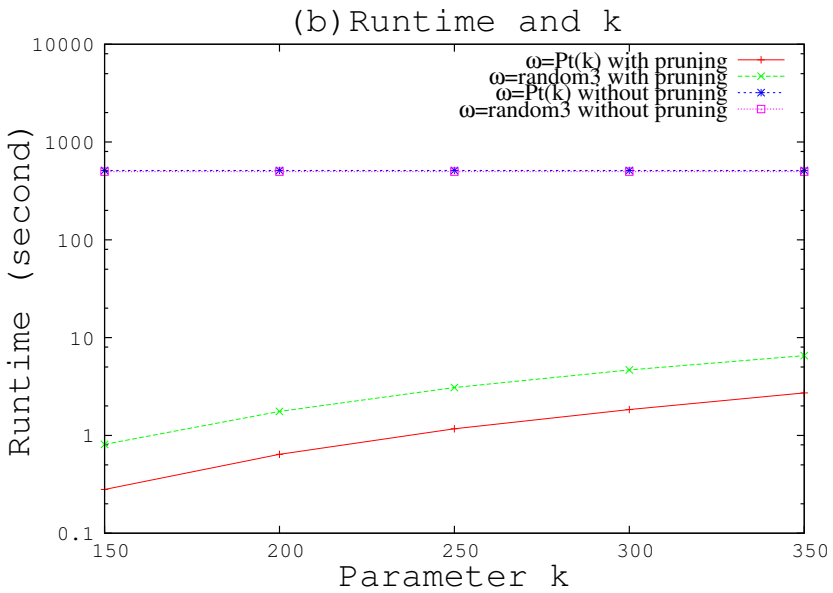

Figure 6.10: Running times for PRF^{ω} on real data set

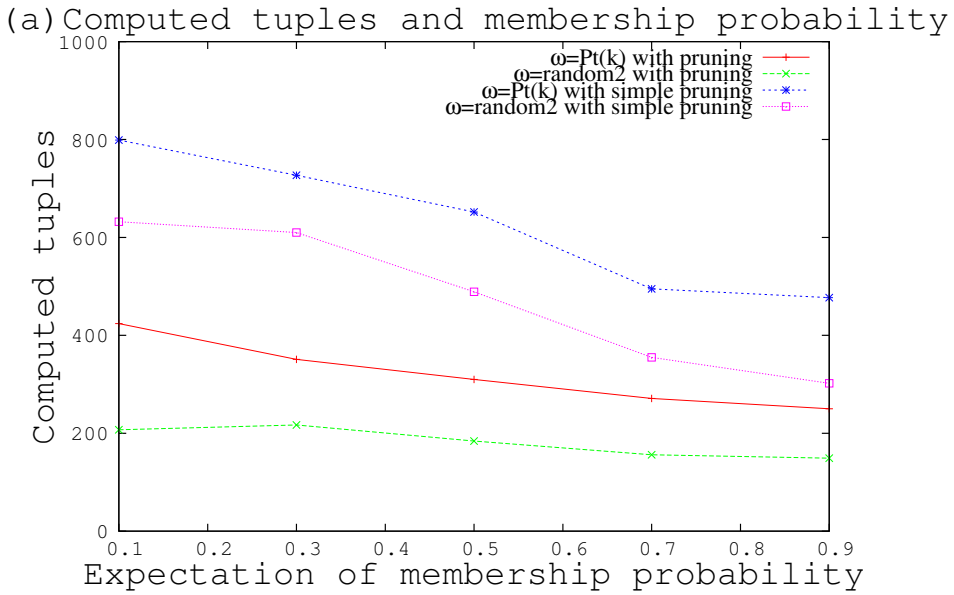

Figure 6.11: Comparison with previous pruning method (computed tuples): membership probabilities of independent tuples

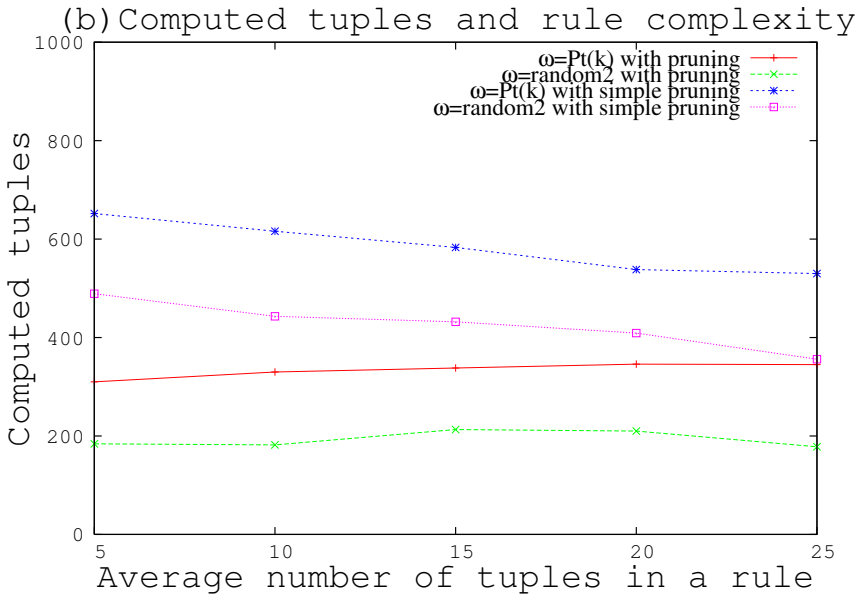

Figure 6.12: Comparison with previous pruning method (computed tuples):sizes of multi-tuple generation rules

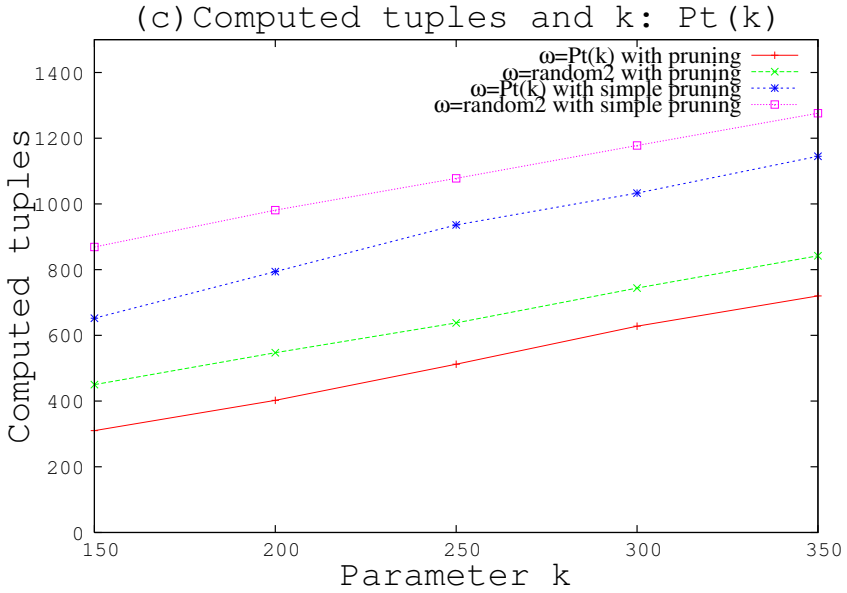

Figure 6.13: Comparison with previous pruning method (computed tuples): parameter k

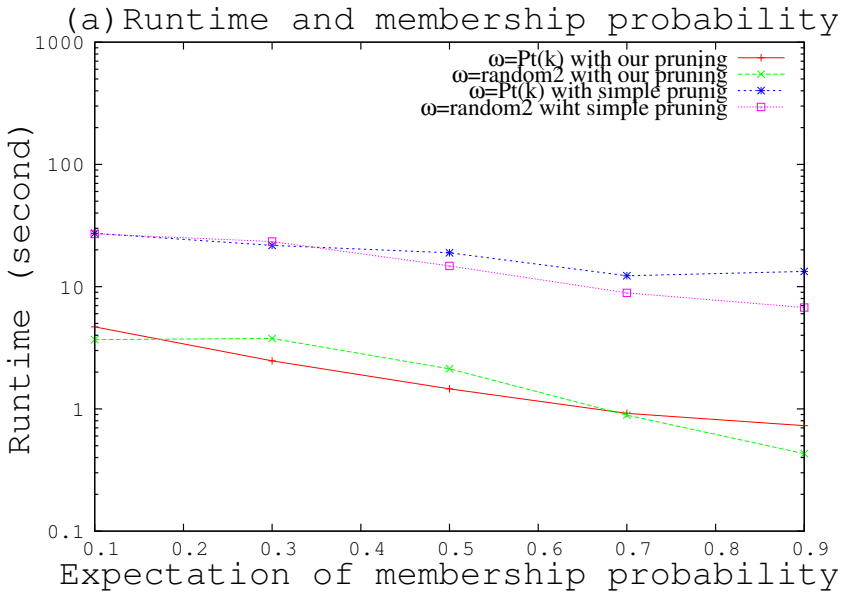

Figure 6.14: Comparison with previous pruning method (running times): membership probabilities of independent tuples

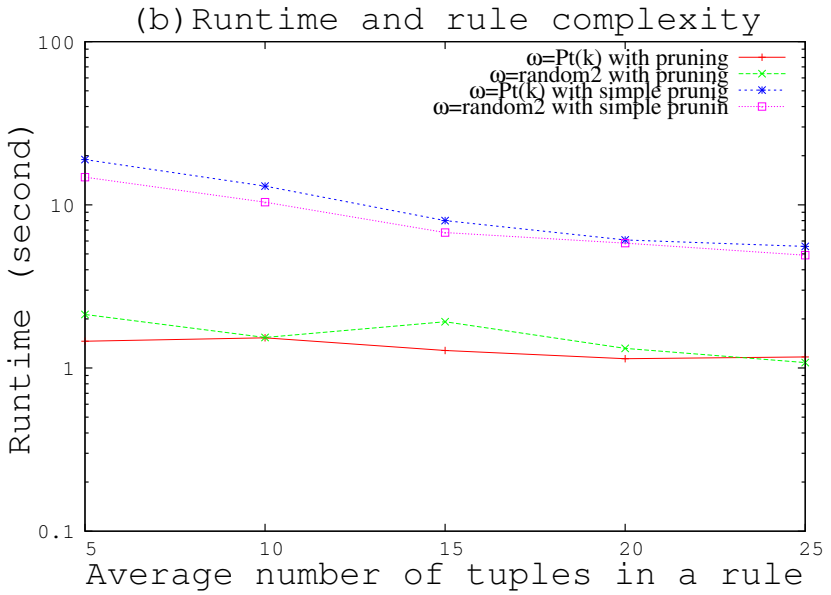

Figure 6.15: Comparison with previous pruning method (running times):sizes of multi-tuple generation rules

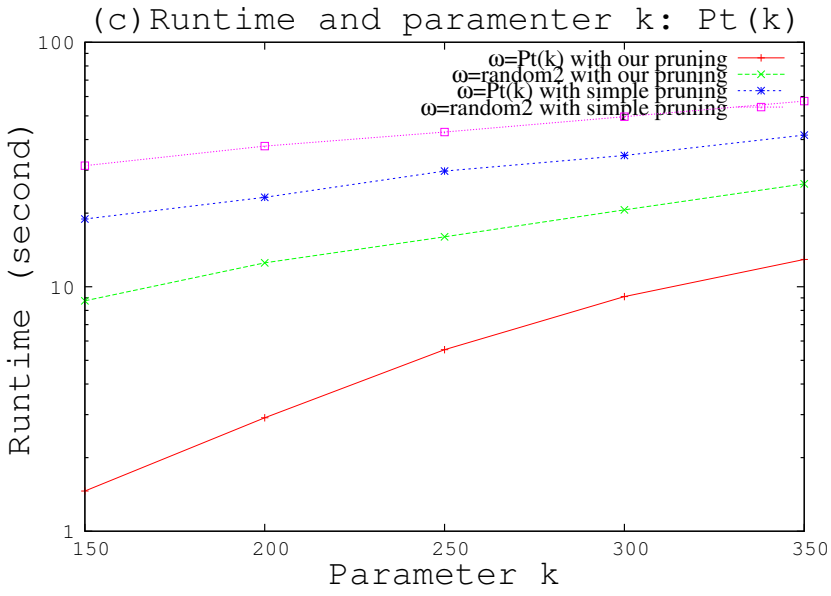

Figure 6.16: Comparison with previous pruning method (running times): parameter k

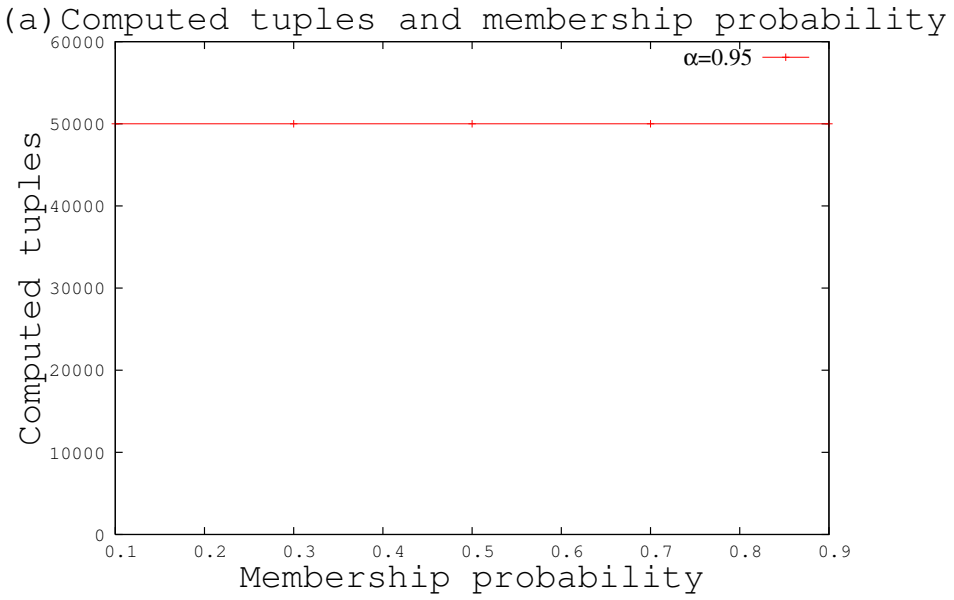

Figure 6.17: Computed tuples for early termination: membership probabilities of independent tuples

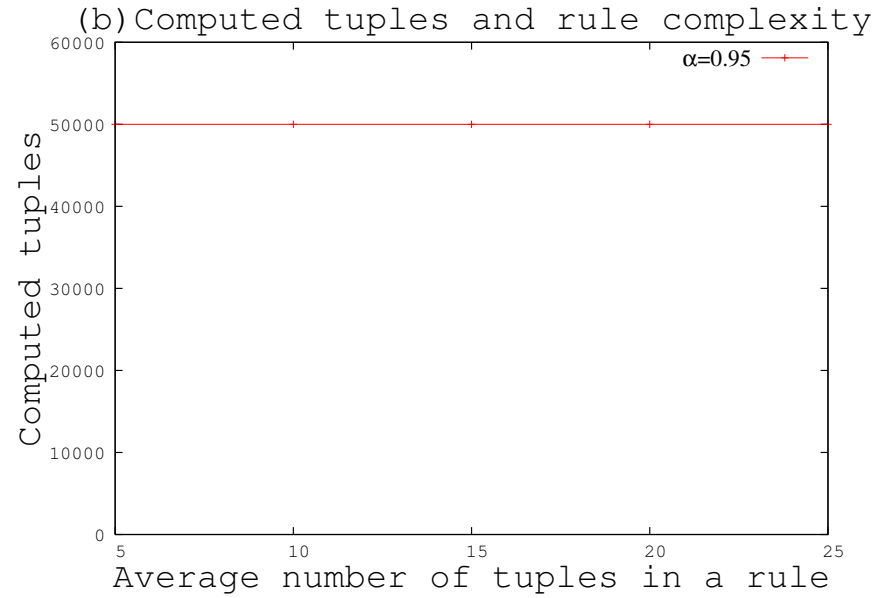

Figure 6.18: Computed tuples for early termination: sizes of multi-tuple generation rules

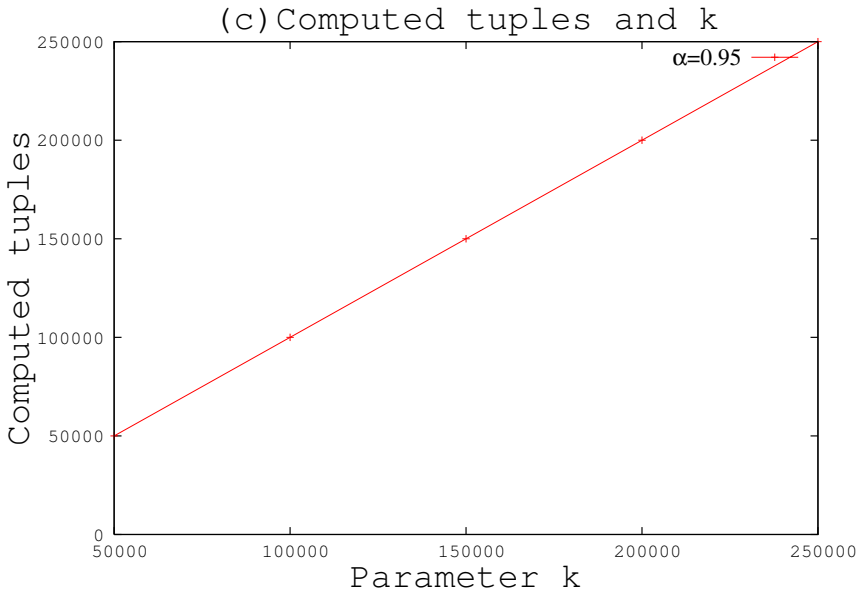

Figure 6.19: Computed tuples for early termination: parameter k

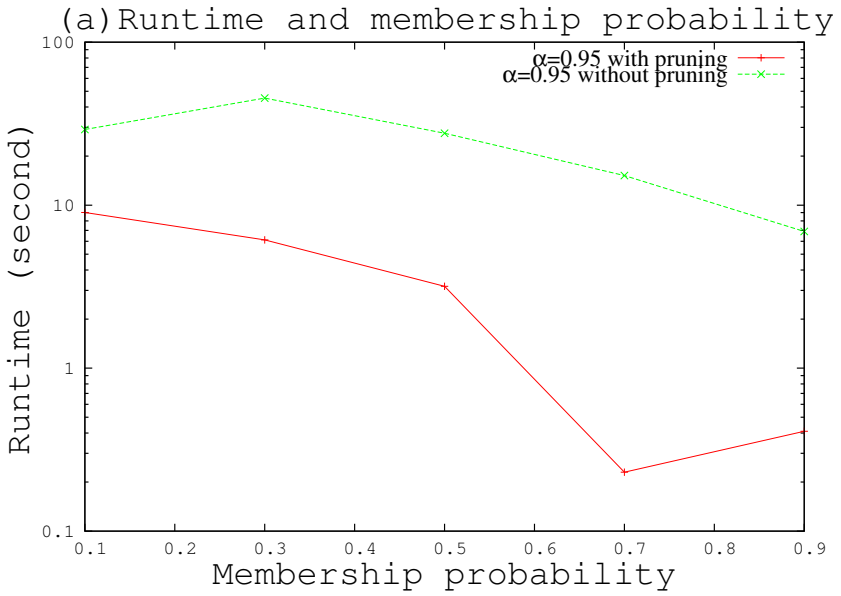

Figure 6.20: Running times for early termination: membership probabilities of independent tuples

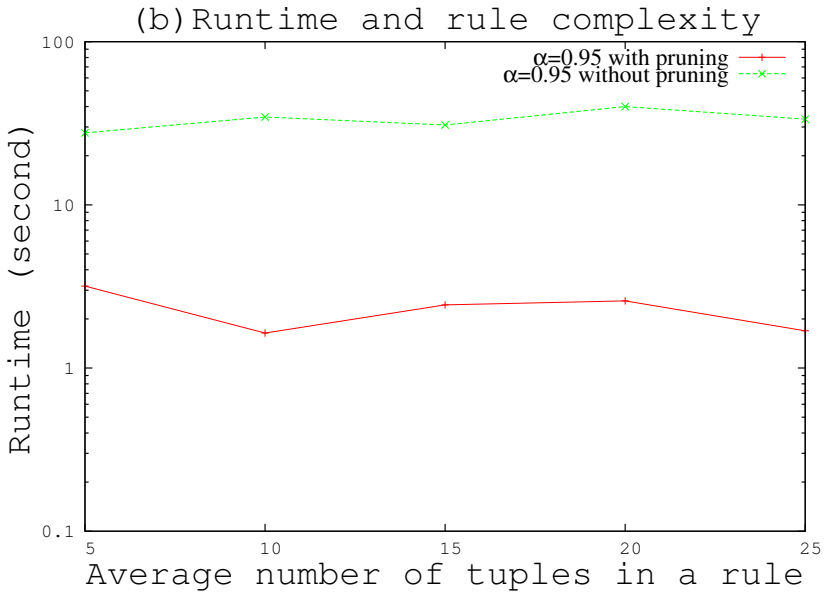

Figure 6.21: Running times for early termination: sizes of multi-tuple generation rules

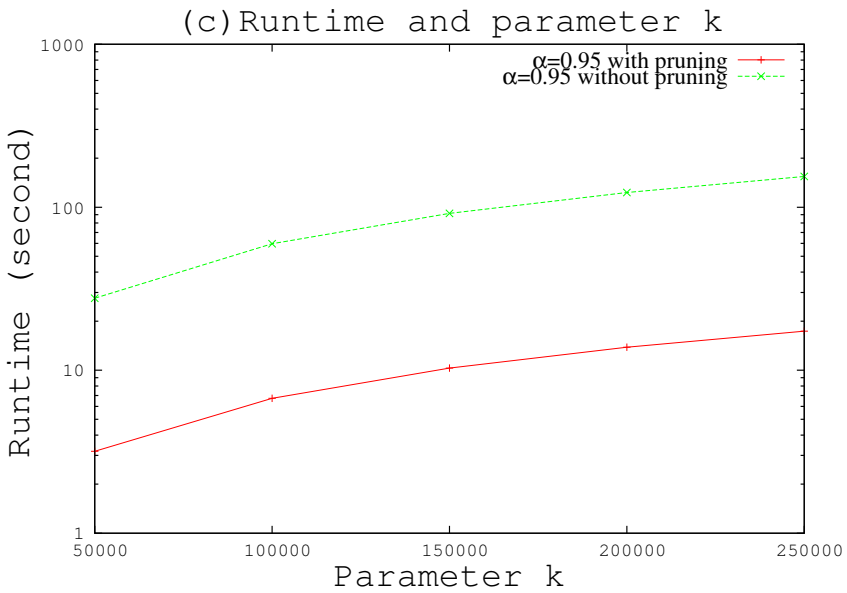

Figure 6.22: Running times for early termination: parameter k

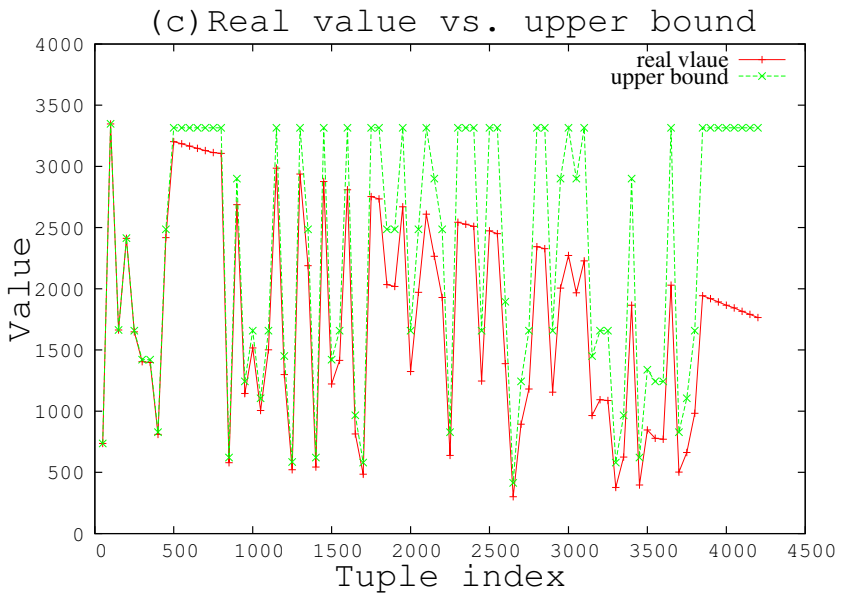

Figure 6.23: Real value vs. upper bound

Chapter 7

A Theory of Top-k Ranking for Objects with Uncertain Data

In this chapter, we present a new ranking theory for objects with uncertain data, where two contributors to uncertainty of data are considered. The first is that the values of an attribute are given in terms of a probability distribution, and the second is that the values of attributes of objects satisfy some stated constraints.

We present this ranking theory in three stages. The first assumes discrete domains. In this case, it is convenient and conceptually intuitive to define top-k objects using the notion of possible worlds. This material is given in Section 7.1. This formulation is extended to include continuous domains. We show that top-k ranking for objects in this context is closely related to the mathematical problems in high-dimensional spaces, in particular, the problem of computing volumes of a high-dimensional polyhedron represented by a system of inequations can be viewed as a subproblem of top-k object ranking of our theory. This material is presented in Section 7.2. Due to this relationship, we can apply the algorithms studied in mathematics for the former to top-k object ranking where the constraints and aggregation function are linear expressions and the probability distributions are continuous uniform. Further in Section 7.3, we consider different weights to different positions of objects and add the aggregation values of objects to top-k ranking such that the ranking result is more reasonable. In Section 7.4, we compare our ranking theory with related work in the literature. We show that a number of definitions of top-k objects in the literature are just special cases of our ranking theory. In addition, our ranking theory can improve the quality of ranking result or extend the application area of the ranking problems in the existing approaches. Section 7.5 gives a limited study on the computation of top-k objects under our definition. Section 7.6 provides a summary.

7.1 Top-k Ranking for Discrete Domains

In this section, we present a theory of top-k ranking for objects whose data values are from discrete domains. The theory is formulated using the possible world semantics.

Definition 7.1.1 *An* uncertain database *(or just a* database*) is a 5-tuple* $D = \langle O, A, X, P, F \rangle$*, where* $O = \{o_1, ..., o_n\}$ *is a set of objects;* $A = \{a_1, ..., a_m\}$ *a set of attributes;* $X = \{x_{ij} \mid 1 \leq i \leq n,\ 1 \leq j \leq m\}$ *where* x_{ij} *is a variable representing the value of the object* o_i *under* a_j*;* $P = \{p_{ij} \mid 1 \leq i \leq n,\ 1 \leq j \leq m\}$ *where* p_{ij} *is the probability distribution of variable* x_{ij}*, and* $F = \{f_1, f_2, ..., f_l\}$ *where each* f_i *is an equation or inequation on* X*.*

In this section, we assume that each variable $x_{ij} \in X$ has a finite discrete domain, and therefore the probability distribution of a variable is also discrete.

Without confusion, given a database D, we will use o_i for objects, a_j for attributes, where $1 \leq i \leq n$ and $1 \leq j \leq m$.

For a database $D = \langle O, A, X, P, F \rangle$, each f_i in F can be written as

$$g(x_{11}, ..., x_{1m}, x_{21}, ..., x_{nm})\ \mathcal{R}\ 0$$

where $\mathcal{R} \in \{\leq, \geq, <, >, =\}$. We assume that g is a *continuous* function.

Definition 7.1.2 *Let* $D = \langle O, A, X, P, F \rangle$ *be a database. An aggregation function for* D *is a mapping* $t : \Re^m \rightarrow \Re$*, where* $\Re$ *is the set of real numbers.*

In our formulation, an application of an aggregation function, written $t(x_{i1}, ..., x_{im})$ (sometimes also written $t(o_i)$, for convenience) is to compute the collective value of object o_i across all attributes. We call such a value an *aggregation value* of object o_i.

For example, suppose in the given database there are two objects, o_1, o_2, and three attributes, a_1, a_2, a_3. Suppose an aggregation function is defined as: $t(x, y, z) = 2x + 3y + z$. Then, the aggregation value of object o_1 is $2x_{11} + 3x_{12} + x_{13}$ and that of o_2 is $2x_{21} + 3x_{22} + x_{23}$.

Given n objects and m attributes, we are interested in tuples of the form

$$\eta = (c_{11}, ..., c_{1m}, c_{21}, ..., c_{nm}) \tag{7.1}$$

where c_{ij} is a value of x_{ij}, i.e., a value of object o_i under attribute a_j. The probability of this tuple, denoted by $Pr(\eta)$, is defined by

$$Pr(\eta) = p_{11}(c_{11}) ... \times p_{21}(c_{21}) \times ... \times p_{nm}(c_{nm}) \tag{7.2}$$

A tuple of (7.1) represents one possible set of values for the underlying variables. Thus a tuple represents a scenario of all objects having their concrete attribute values. Although we know there is one *actual world* (the set of *actual* values for the variables), we do not know which one it is and thus every such set serves as a "possible world".

If $Pr(\eta) > 0$, the tuple η is nontrivial. Following the general idea of the possible world semantics, we define a *possible world* in this context to be a set of the values in η associated with their variables. Given a tuple η in the form (7.1), this can be conveniently denoted by

$$\eta' = \{[x_{11}, c_{11}], ..., [x_{1m}, c_{1m}], [x_{21}, c_{21}], ..., [x_{nm}, c_{nm}])$$

That is, a possible world consists of $n \times m$ elements, each of which is a variable taking a value from its domain. In other words, η is an *assignment* of values to variables for all objects. For notational convenience, we will continue to use the notation of tuple in the form (7.1) to denote a possible world. Thus, the probability of the possible world η', denoted $Pr(\eta')$, is defined to be that of the corresponding tuple η, i.e., $Pr(\eta') = Pr(\eta)$.

For notational convenience, in the sequel, given an aggregation function t, an object o_i $(1 \leq i \leq n)$, and a tuple η of the form (7.1), the *aggregation value* of o_i w.r.t. η, denoted $t_{o_i}(\eta)$, is the aggregation value of o_i computed by t when variable x_{ij} take values c_{ij} $(1 \leq j \leq m)$.

Then, whether an object o_i is a top-k object is determined by how many possible worlds that "support" o_i. Formally, let $D = \langle O, A, X, P, F \rangle$ be a database, $\eta = (c_{11}, ..., c_{1m}, c_{21}, ..., c_{nm})$ a possible world, and t an aggregation function. Given an object o_i, if there are at least $n - k$ other objects $o_{i'}$ such that $t_{o_i}(\eta) \geq t_{o_{i'}}(\eta)$, then we say that the possible world η *supports* object o_i (or, η is *a support* to o_i).

In other words, η supports object o_i whenever η places o_i ahead of at least $n - k$ other objects, under the aggregation function t. This is like *casting a vote*. η supports o_i when it casts its vote to o_i as a top-k object.

We now bring the constraints into the formulation.

Definition 7.1.3 *Let $D = \langle O, A, X, P, F \rangle$ be a database. A possible world $\eta = (c_{11}, ..., c_{1m}, c_{21}, ..., c_{nm})$ is said to be* effective *if the values in this possible world satisfy all the inequations and equations in F.*

If a support to an object is effective, it will be called an *effective support*.

For each object o, we define the *support set of* o, denoted by S_o, to be the set of all the possible worlds that are effective supports to o.

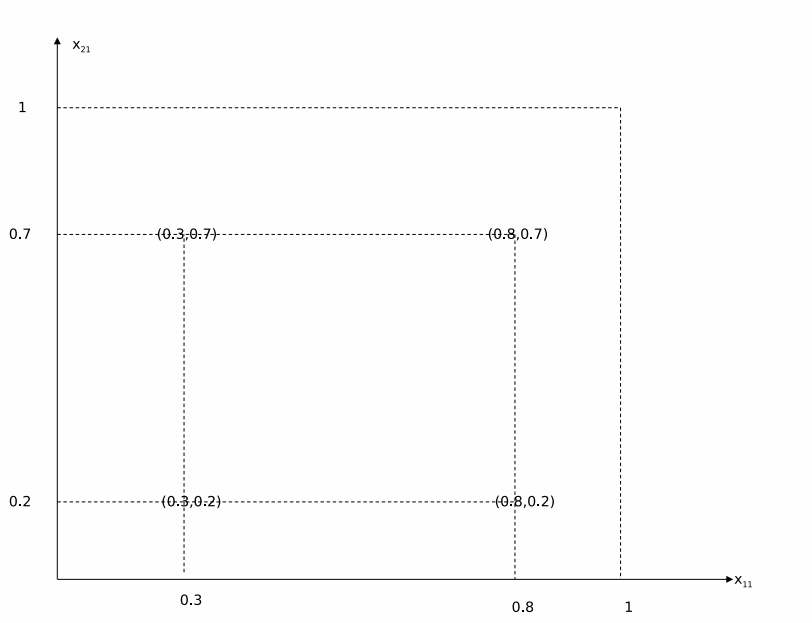

Figure 7.1: Example 7.1.6

Definition 7.1.4 *Let $D = \langle O, A, X, P, F \rangle$ be a database, t an aggregation function. The* support strength *of an object o is defined as $\sum_{\eta \in S_o} Pr(\eta)$.*

Definition 7.1.5 *Let $D = \langle O, A, X, P, F \rangle$ be a database, t be an aggregation function. The top-k objects in D are the k objects with highest support strengthes.*

Here we give some examples of ranking problems covered by this formulation of top-k ranking.

Example 7.1.6 *Suppose there are two objects $O = \{o_1, o_2\}$ and one attribute $A = \{a_1\}$. We thus have two variables $X = \{x_{11}, x_{21}\}$. Assume both domains are $[0, 1]$ and the probability distribution of x_{11} is $p_{11}(x_{11} = 0.3) = 0.7$ (meaning that the probability of the value of x_{11} being 0.3 is 0.7, similarly below) and $p_{11}(x_{11} = 0.8) = 0.3$, and that of x_{21} is $p_{11}(x_{21} = 0.2) = 0.4$ and $p_{21}(x_{21} = 0.7) = 0.6$. Assume we want to find top-1 object.*

The two variables x_{11}, x_{21} in this example can be viewed intuitively as a 2-dimensional space. A possible world can then be viewed as a point in this space, and the variable-value pairs in a possible world as coordinate values. There are 4 possible worlds in this example, which are shown in Figure 7.1. The probability of the possible world $(0.3, 0.2)$ is 0.28. It supports o_1. The probability of the possible world $(0.3, 0.7)$ is 0.42. It supports o_2. The probability of the possible world $(0.8, 0.2)$ is 0.12. It supports o_1. The probability of the

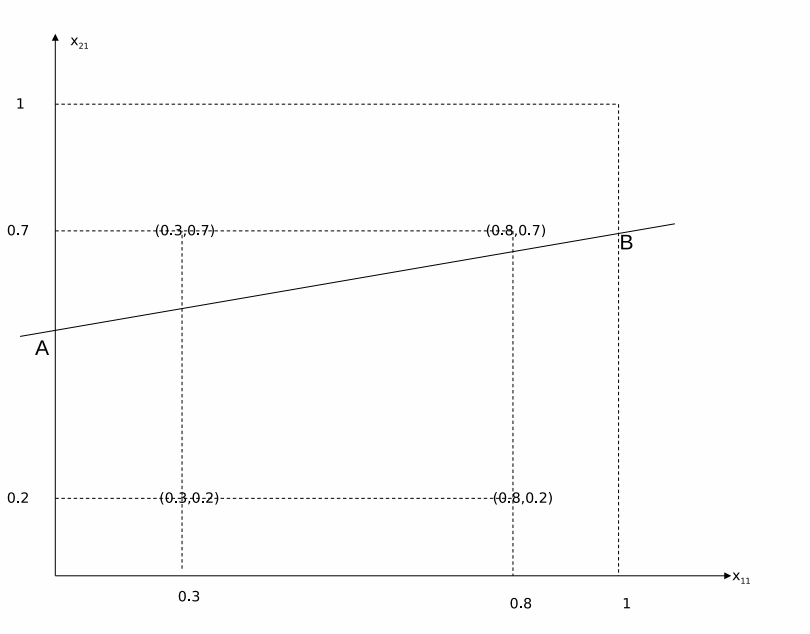

Figure 7.2: Example 7.1.7

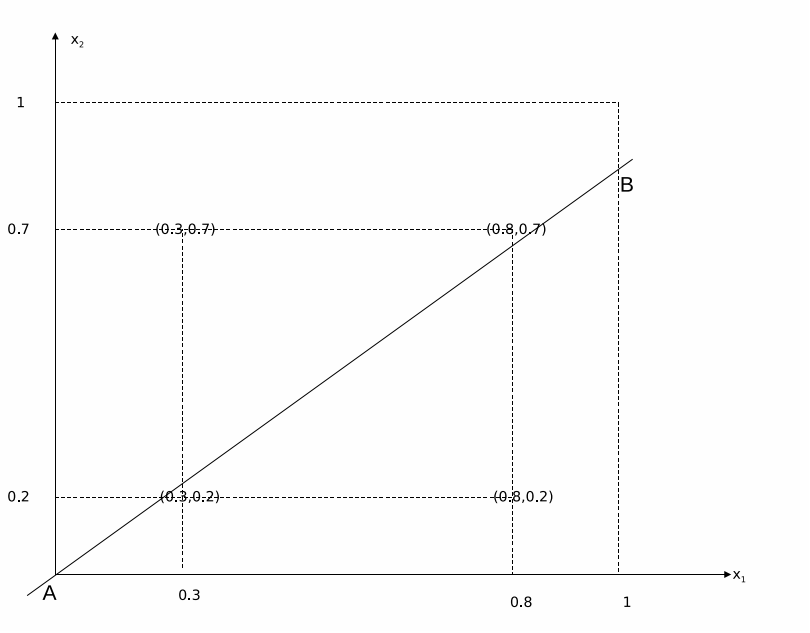

Figure 7.3: Example 7.1.8

possible world $(0.8, 0.7)$ *is 0.18. It supports* o_1*. It can be easily seen that the support strength of* o_1 *is 0.58 and the support strength of* o_2 *is 0.42. Thus,* o_1 *is the top-1 object.*

Example 7.1.7 *The conditions are the same as in Example 7.1.6, but we have a constraint,* $x_{21} > 0.2x_{11} + 0.5$*. This is shown in Figure 7.2. The constraint is captured by the line* AB *in the sense the possible worlds strictly above it (note* $>$ *in the constraint) satisfy the constraint. Apparently, there are only two effective possible worlds. The possible world* $(0.3, 0.7)$ *supports* o_2*, and the possible world* $(0.8, 0.7)$ *supports* o_1*. The support strength of* o_1 *is 0.18 and the support strength of* o_2 *is 0.42. So* o_2 *is the top-1 object.*

Example 7.1.8 *Let us consider a real example. There are two paintings* o_1 *and* o_2*. We are interested in knowing which painting is more expensive. But we do not have exact information about their prices. We have some uncertain information about the prices. We use the variable* x_1 *to represent the price of* o_1 *and the variable* x_2 *to represent the price of* o_2*. We know that the price of* o_1 *satisfies the following probability distribution:* $Pr(x_1 = 0.3) = 0.7$ *(meaning that the probability of the price of* o_1 *being 0.3 million dollars is 0.7, similarly below) and* $Pr(x_1 = 0.8) = 0.3$*. We also know that the price of* o_2 *satisfies the following probability distribution:*$Pr(x_2 = 0.2) = 0.4$ *and* $Pr(x_2 = 0.7) = 0.6$*. And we know that the price of* o_2 *is higher than the price of* o_1 *with 20 percent discount. This can be represented by an inequation:* $x_2 > 0.8x_1$*.*

We can treat two paintings as two objects in our theory. There is only one attribute: price. We have known the probability distribution of the price of each object. We also know a constraint between the prices of these two objects: $x_2 > 0.8x_1$*. We want to find top-1 object. There are 4 possible worlds in this example. The probability of the possible world* $(0.3, 0.2)$ *is 0.28. It supports* o_1*. The probability of the possible world* $(0.3, 0.7)$ *is 0.42. It supports* o_2*. The probability of the possible world* $(0.8, 0.2)$ *is 0.12. It supports* o_1*. The probability of the possible world* $(0.8, 0.7)$ *is 0.18. It supports* o_1*. All the possible worlds are shown in Figure 7.3. The possible worlds above the line* AB *satisfy the constraint* $x_2 > 0.8x_1$*. After considering the constraint, there are only two effective possible worlds. The effective possible world* $(0.3, 0.7)$ *supports* o_2*, and the effective possible world* $(0.8, 0.7)$ *supports* o_1*. The support strength of* o_1 *is 0.18 and the support strength of* o_2 *is 0.42. So* o_2 *is the top-1 object. Based on our theory, from the uncertain information we have, we conclude that the painting* o_2 *has higher probability to be more expensive than the painting* o_1*.*

7.2 Extension to Continuous Domains in High-Dimensional Space

In the definition above, we used possible worlds to define top-k ranking, where probability distributions are assumed to be discrete. When a probability distribution is continuous, we have continuous domains for variables. If we continue to use a possible world to represent a point in a high-dimensional space, then we are going to have infinitely many possible worlds (which is fine). In this case, it is equivalent, yet more convenient, to represent a point by its coordinate values. This does not change the nature of the semantics even for discrete domains. However, some technical details need to be handled for continuous domains.

The definitions of database $D = \langle O, A, X, P, F\rangle$ and aggregation function are the same as before. Each variable $x_{ij} \in X$ can be viewed a dimension in an $n \times m$ dimensional space, and a tuple $\eta = (c_{11}, ..., c_{1m}, c_{21}, ..., c_{nm})$ represents a point by its coordinate values. We then can represent the support set in Section 7.1 by a system of equations and inequations, which over n variables defines a q-dimensional space, where $q \leq n$. This space contains all the points whose coordinate values satisfy all the equations and inequations and does not contain any points whose coordinates conflict with any equation or inequation.

We assume that the domain of each variable in X is bounded finitely, i.e.,

$$l_{ij} \leq x_{ij} \leq u_{ij} \ (1 \leq i \leq n, 1 \leq j \leq m) \tag{7.3}$$

where l_{ij} and u_{ij} are real numbers. To get the top-k objects, we need to define some spaces.

The first space, denoted by Γ, is defined by all the inequations and equations in F and the domain of each variable in X.

The second space, denoted V_i w.r.t o_i, is defined by the domain of each variable in X and the constraints for the notion of support - *η supports an object o iff there are at least $n - k$ other objects o' such that $t_o(\eta) \geq t_{o'}(\eta)$*. This space can be represented by systems of equations and inequations and we will describe it later.

Let $D = \langle O, A, X, P, F\rangle$ be a database and t be an aggregation function. For an object o_i, we define the *support space to o_i* to be the space $\Upsilon_i = V_i \cap \Gamma$. We will say that *all the points in Υ_i support o_i*.

For $D = \langle O, A, X, P, F\rangle$, we assume that all the probability distributions in P are independent. If all the probability distribution in P are discrete, we get the joint probability mass function of X:

$$f(x_{11}, x_{12}, ..., x_{1m}, x_{21}, ..., x_{nm}) = \prod_{i=1}^{n}\prod_{j=1}^{m} p_{ij}.$$

We define the *support strength of* o_i as (assuming $\eta = (c_{11}, ..., c_{1m}, c_{21}, ..., c_{nm})$):

$$\Lambda(o_i) = \sum_{\eta \in \Upsilon_i} f(x_{11} = c_{11}, ..., x_{21} = c_{21}, ..., x_{nm} = c_{nm}).$$

If some of the probability distributions in P are continuous, we get the joint probability density function of X:

$$f(x_{11}, x_{12}, ..., x_{1m}, x_{21}, ..., x_{nm}) = \prod_{i=1}^{n} \prod_{j=1}^{m} p_{ij}.$$

Let Θ be the set of points which contain all the points with joint probability density function value greater than 0 and does not contain any point with joint probability density function value equal to 0.

Consider all the spaces below

$$\Upsilon_i \cap \Theta \ (1 \leq i \leq n) \tag{7.4}$$

We discuss these spaces in two cases.

First, if all the spaces in expression (7.4) are empty, we define *support strength* of any object $o_i \in O$ to be 0.

Second, if not all the spaces in (7.4) are empty, let s $(0 \leq s \leq n \times m)$ be the maximal dimension among all spaces in (7.4). Given an i $(1 \leq i \leq n)$, let $\chi_i = \{\eta_1, ..., \eta_q\}$, where q is a positive integer, be the set of s-dimensional spaces in $\Upsilon_i \cap \Theta$, and

$$x_{u_{e1}v_{e1}}, ..., x_{u_{ed}v_{ed}}$$

where $1 \leq e \leq q, 1 \leq d \leq n \times m, 1 \leq u_{ew} \leq n, 1 \leq v_{ew} \leq m, 1 \leq w \leq d$, be the variables which take different values in η_e. Then we define the *support strength* of o_i, $\Lambda(o_i)$, as follows

- If $\chi_i = \emptyset$ then $\Lambda(o_i) = 0$;
- If χ_i contains q points, then
  $\Lambda(o_i) = \sum_{(c_{11}, ..., c_{nm}) \in \chi_i} f(x_{11} = c_{11}, ..., x_{nm} = c_{nm})$
- If χ_i contains q s-dimensional spaces $(1 \leq s \leq n \times m)$ then
  $\Lambda(o_i) = \sum_{\eta_e \in \chi_i} \idotsint_{\eta_e} f(x_{11}, ..., x_{nm}) dx_{u_{e1}v_{e1}} ... dx_{u_{ed}v_{ed}}$
  (if variable x_{ij} can only take a fixed value c_{ij} in η_e, replace x_{ij} with c_{ij} in $\idotsint_{\eta_e} f(x_{11}, ..., x_{nm})$).

Then, the top-k objects in D are the k objects with the highest support strengthes.

Recall that we have defined V_i w.r.t o_i. Now let us see how to formally express V_i in some systems of equations and inequations. For an object $o_i \in O$, let W_{ij} be a set of $n-k$ objects in $O - \{o_i\}$, i.e., $W_{ij} = \{o_{a_1}, o_{a_2}, ..., o_{a_{n-k}}\}$ where $o_{a_i} \neq o_i$. Because we can choose any $n-k$ objects from $O - \{o_i\}$, we know there are C_{n-1}^{n-k} different W_{ij}. So $1 \leq j \leq C_{n-1}^{n-k}$.

Let $W_{ij} = \{o_{a_1}, o_{a_2}, ..., o_{a_{n-k}}\}$. Let U_{ij} denote the following set of inequations

$$t(x_{i1}, ..., x_{im}) \geq t(x_{a_h 1}, ..., x_{a_h m}) \;\; 1 \leq h \leq n-k$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \; (1 \leq i \leq n, 1 \leq j \leq m)$$

and V_{ij} denote the space defined by U_{ij}. Define

$$V_i = \cup_{1 \leq j \leq C_{n-1}^{n-k}} V_{ij}$$

For a database $D = \langle O, A, X, P, F \rangle$, where $O = \{o_1, ..., o_n\}$ and $A = \{a_1, ..., a_m\}$, we assume the maximal dimension of all the support spaces of objects is s. When the probability distributions in P are all continuous uniform distributions over entire domains, we just need to calculate the volume of the s dimension support space to each object o_i to get top-k objects. Because each point has the same probability density function value and the support strength of an object is the product of the the volume of the s dimension support space to this object and the probability density function value, we can use the volume of s dimension support space to an object to measure the support strength of an object. So in this situation, the top-k objects in D is the k objects with largest volumes of s dimension support spaces. So we have the following theorem.

Theorem 7.2.1 *Let $D = \langle O, A, X, P, F \rangle$ be a database, where $O = \{o_1, ..., o_n\}$ is a set of objects; $A = \{a_1, ..., a_m\}$ a set of attributes; $X = \{x_{ij} \mid 1 \leq i \leq n, \; 1 \leq j \leq m\}$ where x_{ij} is a variable representing the value of the object o_i under a_j; $P = \{p_{ij} \mid 1 \leq i \leq n, \; 1 \leq j \leq m\}$ where p_{ij} is continuous uniform probability distribution of variable x_{ij} over the entire domain, and $F = \{f_1, f_2, ..., f_l\}$ where each f_i is an equation or inequation on X. Assume the maximal dimension of all the support spaces of objects is s. The top-k objects in D are the k objects with largest k volumes of s dimension support spaces.*

Example 7.2.2 *We keep all the conditions as in Example 7.1.7, except the probability distribution. We change the probability distribution of x_{11} and x_{21} to be continuous uniform distribution. The example is illustrated in Figure 7.4. Since the probability distribution of*

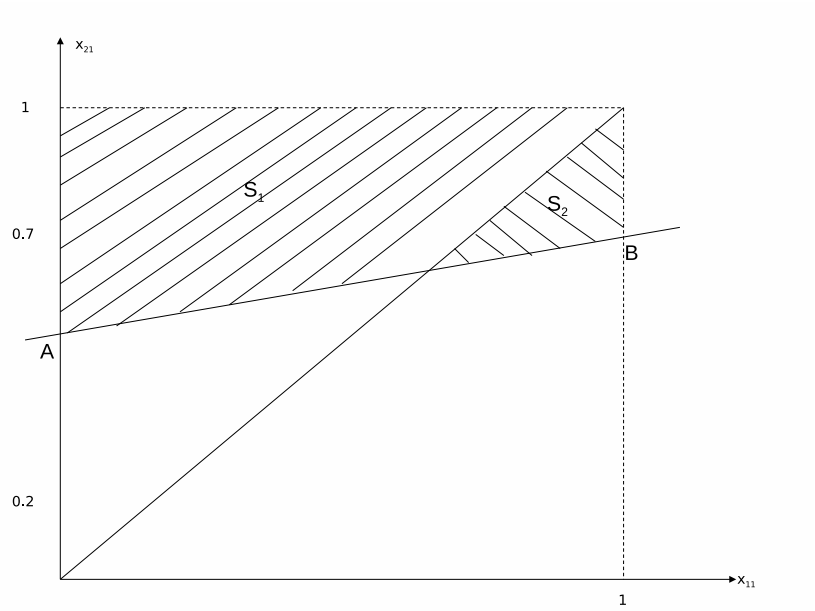

Figure 7.4: Example 7.2.2

each variable is continuous uniform distribution, we use volume to measure the support strength to an object. For this example, as the problem is in 2-dimensional space, we can use area to measure the support strength to an object. All the points inside S_1 support o_1 and all the points inside S_2 support o_2. S_1 is the support space to o_1 and S_2 is the support space to o_2. The maximal dimension of all the support spaces to objects is 2. S_1 can be described by the following inequations:

$$x_{11} \geq x_{21}$$
$$x_{21} > 0.2x_{11} + 0.5$$
$$0 \leq x_{11} \leq 1$$
$$0 \leq x_{21} \leq 1$$

S_2 can be described by the following inequations:

$$x_{21} \geq x_{11}$$
$$x_{21} > 0.2x_{11} + 0.5$$
$$0 \leq x_{11} \leq 1$$
$$0 \leq x_{21} \leq 1$$

As the area of S_2 is larger than the area of S_1, o_2 is the top-1 object.

Example 7.2.3 *We keep all the conditions as in Example 7.2.2, except the constraint. We change the constraint to $x_{21} = 0.2x_{11} + 0.5$. The example is shown in Figure 7.5. We can use the length of a line to measure the support strength of an object. All the points in line*

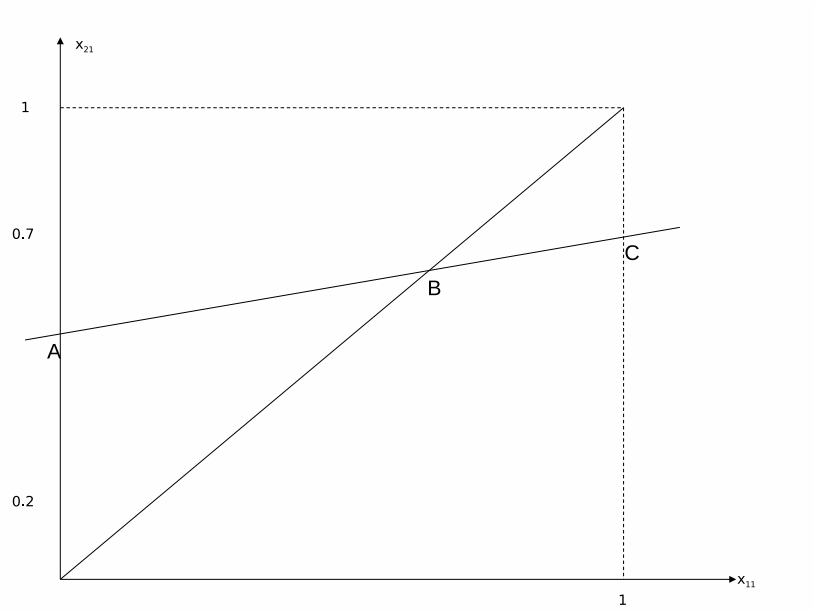

Figure 7.5: Example 7.2.3

BC supports o_1 and all the points in line AB supports o_2. Line BC is the support space to o_1 and line AB is the support space to o_2. The maximal dimension of all the support spaces to objects is 1. Because the length of AB is longer than the length of BC, o_2 is the top-1 object. This example shows that constraint can be equations.

Example 7.2.4 *We keep all the conditions in Example 7.2.2, except the constraint. We change the constraint to $(x_{11} - 0.9)^2 + (x_{21} - 1.1)^2 = 0.04$. The example is shown in Figure 7.6. All the points in the arc BC support o_1 and all the points in the arc AB support o_2. Because the length of the arc AB is longer than the length of the arc BC, o_2 is the top-1 object. This example shows that constraint can be non-linear equations or inequations.*

Example 7.2.5 *We keep all the conditions in Example 7.2.2, except the probability distribution. We change the probability distribution of x_{11} p_{11} and the probability distribution of x_{21} p_{21} to continuous non-uniform distribution. The example is also shown in Figure 7.4. We cannot use volume to measure the support strength. We have to use integration. As all the points inside S_1 support o_1, we can use the following integration to compute the support strength of o_1: $\iint_{S_1} p_{11}p_{21}dx_{11}dx_{21}$. Similarly, we can use the following integration to compute the support strength of o_2: $\iint_{S_2} p_{11}p_{21}dx_{11}dx_{21}$.*

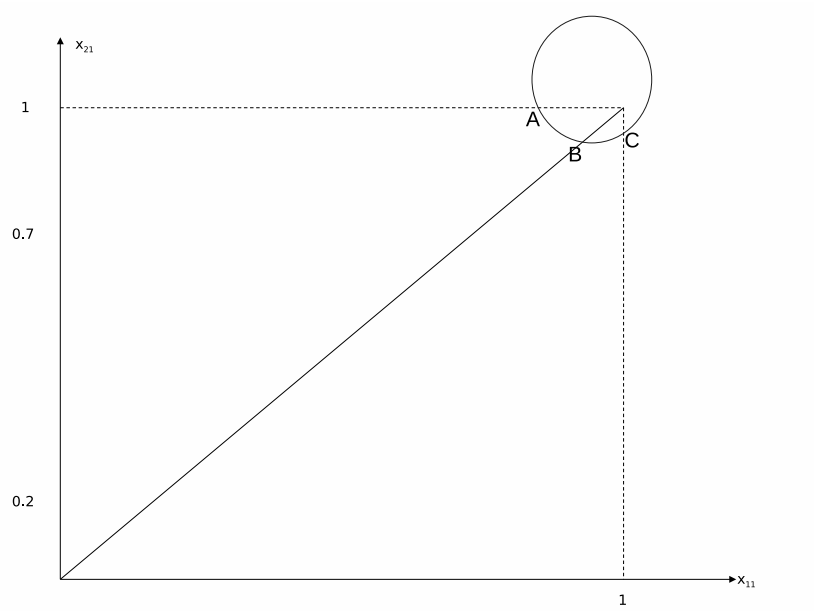

Figure 7.6: Example 7.2.4

7.3 Further Extensions

From a point in a high-dimensional space, we can rank the underlying objects according to their aggregation values. In this ranking, each object has a position. In the formulation of the previous section, there is no difference for an object to be ranked at the first position or 2nd position. Each position in the top k positions has the same influence to the final ranking result, and each position lower than k has no influence to the final ranking result. This is reasonable in some applications. But sometimes it is desirable to assign weights to different positions. For example, if an object ranks the first in a point, it should get more support than the object ranked the second from the same point. This concept is common in real life. For example, in a sport event a gold metal weighs more than a silver metal.

In this section, we define the position of an object in a point to be the number of objects with higher aggregation values.

In our original ranking theory, the aggregation values of objects are used to give an order of objects in a point. After the order is determined, the aggregation value itself will not be used in the ranking again. For example, candidate A is preferred over candidate B for trustworthy, but the extent of this preference is not considered previously. Therefore, sometimes it is desirable to include the aggregation values in the process of ranking. In this section, we extend our ranking theory in Section 7.2 so that different positions can get

different weights and aggregation values of objects themselves can be included in top-k ranking.

We note that the notion of *parameterized ranking functions* introduced in [29] embodies a similar concept.

We now give the details. Let database $D = \langle O, A, X, P, F\rangle$ and aggregation function t be the same as before.

The space V_i^b w.r.t. o_i is defined by the domain of each variable in X and the constraints for the notion of *b-support - a point η in a high-dimensional space gives a b-support to an object o iff there are exactly b other objects o' such that $t_{o'}(\eta) > t_o(\eta)$*.

As in the previous section, we let Γ be the space defined by the inequations or equations in F. We define $\Upsilon_i^b = V_i^b \cap \Gamma$ to be the *b-support space* to o_i. Note that o_i is in the b-th position in all the points of the b-support space to o_i.

Let us use $Position(o_i)$ to represent the position of an object in a point. Let $\omega : \Re \times N \rightarrow \Re$ be a weight function. The expression $\omega(t(o_i), Position(o_i))$ specifies a weight for an object in a position. This weight function includes the aggregation value of an object. $\omega(t(o_i), Position(o_i))$ can be defined in many different ways. It can be independent of $t(o_i)$ or $Position(o_i)$. If $\omega(t(o_i), Position(o_i)) = 1$ $(0 \leq Position(o_i)) \leq k-1)$ and $\omega(t(o_i), Position(o_i)) = 0$ $(k \leq Position(o_i) \leq n-1)$, then we get the same top-k ranking definition as the one in Section 7.2.

For $D = \langle O, A, X, P, F\rangle$, we still assume that all the probability distributions in P are independent. If all the probability distributions in P are discrete, we get the joint probability mass function of X:

$$f(x_{11}, x_{12}, ..., x_{1m}, x_{21}, ..., x_{nm}) = \prod_{i=1}^{n}\prod_{j=1}^{m} p_{ij}$$

We define the *support strength* of o_i as

$\Lambda(o_i) = \sum_{b=0}^{n-1} \sum_{(c_{11},...,c_{1m},c_{21},...,c_{nm})\in\Upsilon_i^b} \omega(t(o_i), b) f(x_{11} = c_{11}, ..., x_{1m} = c_{1m}, x_{21} = c_{21}, ..., x_{nm} = c_{nm})$.

If some of the probability distributions in P are continuous, we get the joint probability density function of X:

$$f(x_{11}, x_{12}, ..., x_{1m}, x_{21}, ..., x_{nm}) = \prod_{i=1}^{n}\prod_{j=1}^{m} p_{ij}$$

We define Θ to be the same as in Section 7.2.

We observe all the spaces below

$$\Theta \cap \Upsilon_i^b \ (1 \leq i \leq n, 0 \leq b \leq n-1) \tag{7.5}$$

We discuss these spaces in two cases.

First, if all the spaces in (7.5) are empty, we define *the support strength* of any object $o_i \in O$ to be 0.

Second, if not all the spaces in (7.5) are empty, let s $(0 \leq s \leq n \times m)$ be the maximal dimension of spaces in (7.5). Given an i $(1 \leq i \leq n)$, let $\chi_i^b = \{\eta_1, ..., \eta_q\}$, where q is a positive integer, be the set of s-dimensional spaces in $\Upsilon_i^b \cap \Theta$ and

$$x_{u_{e1}v_{e1}}, ..., x_{u_{ed}v_{ed}}$$

where $1 \leq e \leq q, 1 \leq d \leq n \times m, 1 \leq u_{ew} \leq n, 1 \leq v_{ew} \leq m, 1 \leq w \leq d$, be the variables which can take different values in η_e, then we define *the b-support strength of o_i*, $\Lambda^b(o_i)$, as follows

- If $\chi_i^b = \emptyset$ then $\Lambda^b(o_i) = 0$;
- If χ_i^b contains q points, then $\Lambda^b(o_i) =$
  $\sum_{(c_{11},...,c_{nm}) \in \chi_i^b} \omega(t(o_i), b) f(x_{11} = c_{11}, ..., x_{nm} = c_{nm})$
- If χ_i^b contains q s-dimension spaces$(1 \leq s \leq n \times m)$ then
  $\Lambda^b(o_i) = \sum_{\eta_e \in \chi_i^b} \iint_{\eta_e} \omega(t(o_i), b) f(x_{11}, ..., x_{nm}) dx_{u_{e1}v_{e1}} ... dx_{u_{ed}v_{ed}}$
  (if variable x_{ij} can only take a fixed value c_{ij} in η_e, replace x_{ij} with c_{ij} in
  $\iint_{\eta_e} \omega(t(o_i), b) f(x_{11}, ..., x_{nm})$)

We define *the support strength of o_i*: $\Lambda(o_i) = \sum_{b=0}^{n-1} \Lambda^b(o_i)$.

The top-k objects in D are the k objects with highest support strength (if smaller values of weights are considered more important, then the top-k objects in D are the k objects with lowest support strength).

Here we show how to express V_i^b as some systems of inequations. For an object $o_i \in O$, let $W_{ij}^b (0 \leq b \leq n-1)$ be a set of b objects in $O - \{o_i\}$. Because we can choose any b objects in $O - \{o_i\}$, there are C_{n-1}^b different W_{ij}^b for each b. So for each b, $1 \leq j \leq C_{n-1}^b$.

Let $W_{ij}^b = \{o_{a_1}, o_{a_2}, ..., o_{a_b}\}$. And let $O - W_{ij}^b - \{o_i\} = \{o_{a_{b+1}}, o_{a_{b+2}}, ..., o_{a_{n-1}}\}$. Let U_{ij}^b denote the following set of inequations:

$$\begin{array}{l} t(x_{a_h 1}, ..., x_{a_h m}) > t(x_{i1}, ..., x_{im}) \ 1 \leq h \leq b \\ t(x_{i1}, ..., x_{im}) \geq t(x_{a_h 1}, ..., x_{a_h m}) \ b+1 \leq h \leq n-1 \\ l_{ij} \leq x_{ij} \leq u_{ij} \ (1 \leq i \leq n, 1 \leq j \leq m) \end{array}$$

Let V_{ij}^b denote the space defined by U_{ij}^b. Let

$$V_i^b = \cup_{1 \leq j \leq C_{n-1}^b} V_{ij}^b$$

7.4 Comparison with Related Work

7.4.1 Application of Constraint

As introduced in Section 2.5, Agrawal et al. [36] propose a method for ranking objects when some relations between objects are specified. In [36], an example of actors is given. We know some preferences between actors and we want to find top-k popular actors. To see the difference of the method in [36] from ours, assume there is only one set of preferences which consist of a partial order. When using the method in [36], two actors can rank before or after each other if there is no preference between them. For instance, suppose there are 100 actors $A_1, ..., A_{100}$. We assume the partial order is: A_1 is more popular than $A_2, ..., A_{98}$ and A_{99} is more popular than A_{100}. Using the method in [36], we cannot tell which one between A_1 and A_{99} is more popular. But our theory says A_1 has more support strength than A_{99} . Here we assume each actor has a popularity score and this score is an uniform probability distribution in $[0, 1]$. And we assume the weight function is only related to positions of actors in possible worlds and positions in front have higher weights. The partial order is the constraints. Then we get our ranking result which ranks A_1 ahead of A_{99}. Due to the preferences, A_1 is more popular than most other actors and A_{99} is more popular than only one actor, A_1 has more chances to be ranked ahead of A_{99}. These observations lead to the conclusion that out ranking theory gives more reasonable ranking results than the one in [36].

As introduced in Section 2.6, PageRank is an algorithm for page ranking in the world wide web [37]. The relations between pages can be described by a group of linear equations:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where p_i is a page, $PR(p_i)$ is the page rank of the page p_i, $M(p_i)$ is the set of pages that link to p_i, d is the *damping factor*, $L(p_j)$ is the number of outbound links on page p_j, and N is the total number of pages.

One can find that this can be thought of as an extreme case of our ranking theory given in Section 7.3. One can think of pages as objects, and there is only one attribute *page rank*. $X = \{x_{11}, x_{21}, ..., x_{n1}\}$. The domain for each variable is $[0, 1]$. We can assume the probability distribution of each variable is continuous uniform distribution over the entire domain. For simplicity, we assume the joint probability density function is 1. The set of equations above is the constraint set F. The aggregation function $t(o_i) = x_{i1}$. The weight function is $\omega(t(o_i), Position(o_i)) = t(o_i) = x_{i1}$. And we want to find top-N objects.

There is only one solution for this system of linear equations. So the support space to each object is just this point. The support strength of each page is the page rank of each page.

If some equations are missed in the group of equations above such that the number of variables is more than the number of equations for some reasons, our ranking theory can still give the page rank for pages using the setting we just introduced because we can still compute the support strength for each object and it is the page rank of each page.

7.4.2 Comparison with top-k ranking in uncertain databases

As introduced in Subsection 2.2.3, in [34], the authors propose the definition of top-k ranking for attribute-level uncertainty model in uncertain databases. We can think of a tuple under the attribute-level uncertainty model as an object. Take a look at the definition of the database $D = \{O, A, X, P, F\}$ in Section 7.3. The objects set O is the set of tuples. There is only one attribute to be ranked. So A contains one attribute. $X = \{x_{11}, x_{21}, ..., x_{n1}\}$. Each variable has a discrete probability distribution. There are no constraints. So F is empty. The weight function $\omega(t(o_i), Position(o_i)) = Position(o_i)$. Here $Position(o_i)$ is $rank_W(t_i)$ (a possible world W can be thought of as a point). So the definition of top-k ranking under the attribute-level uncertainty model in [34] can be thought of as a special case of our extended ranking theory in Section 7.3.

For the definition of top-k ranking in [34], our definition can improve the quality of the ranking result by considering different weights for positions. The weights of positions in [34] are fixed. But in real applications, we may wish to adjust the weights of positions depending on different conditions. Sometimes we may want to include the values of objects into the process of ranking. This is allowed in our ranking theory but not in [34].

7.4.3 Comparison with top-k ranking in continuous probabilistic datasets

As introduced in Section 2.3, in [56], the authors propose *parameterized ranking function (PRF)* for top-k ranking in continuous probabilistic datasets. Here, if we treat tuples as objects and add the following three restrictions, the top-k tuples under the definition PRF is then the top-k objects defined in Section 7.3. First, we assume each tuple's existence probability is 1. That is, it is certain that all the tuples in a given dataset do exist. It follows that each possible world contains n tuples. Second, we restrict the weight function as a mapping to real numbers. Finally, we assume that the domain of each tuple's score is defined by an interval (whose bounds are real numbers). Under these restrictions, let the object set O be the set of tuples, the attribute be the only one in A, the variable set be

$X = \{x_{11}, x_{21}, ..., x_{n1}\}$, and F be empty. Then, the answer of top-k query here is just a special case of our definition of top-k objects in Section 7.3.

As introduced in Section 2.4, in [35], the authors propose to rank records with uncertain scores in databases. Actually, if we treat records as objects, the answer of k-Utop-Rank$(1, k)$ query is the top-k objects defined in Section 7.2. Take a look at the definition of the database $D = \{O, A, X, P, F\}$ in Section 7.2. Here the objects set O is the set of records. Each record is an object. Each object only has an attribute. So there is only one attribute in A. $X = \{x_{11}, x_{21}, ..., x_{n1}\}$. Each variable has a finite domain: the interval $[lo_i, up_i]$. The probability density function of each variable in X is given. There are no constraints. So F is empty. Thus the answer of k-Utop-Rank$(1, k)$ query is just a special case of our definition of top-k objects in Section 7.2. In our definition, there is no need to transfer the score intervals to a partial order.

Both definitions in [56] and [35] consider only one attribute. But our theory can handle multiple attributes with uncertain values. So our theory extends the application areas of the ranking on uncertain scores with continuous probability distributions.

7.5 Computation

Due to the high complexity in the computation of top-k objects under the general definition of our ranking theory, we present a limited study on the method of computation. In this section, we study the computational problem for two special cases:

- Computing the top-k objects whose strengths are calculated as the volumes of polyhedra expressed in linear inquations - this is an exercise to illustrate how to use an existing algorithm for the latter to compute the former.
- Computing the top-k objects for objects with two attributes with probability distributions - this extends the previous work of [56] in which a polynomial algorithm is designed to compute top-k tuples with one attribute under a probability distribution.

7.5.1 Computing top-k objects as computing volumes of high-dimensional polyhedra

Under some restrictions, the problem of computing top-k objects in our theory is equivalent to computing volumes of high-dimensional polyhedra expressed by linear inequations. In this section we show this connection.

Here are the restrictions. Let $D = \langle O, A, X, P, F \rangle$ be a database. We restrict F to linear inequations and require that the aggregation function be linear. We also restrict the probability distributions in P to continuous uniform distributions over the whole domains. Let the maximal dimension of all the support spaces to objects to be s. Based on Theorem 7.2.1, we only need to compute the volumes of s dimension support spaces to objects to get the top-k objects. Since the aggregation function is linear, we can see that U_{ij} only contains linear inequations. As F only contains linear inequations, the s dimension support space to object o_i is the union of the high-dimensional spaces each of which is a high-dimensional polyhedron represented by a system of linear inequations. Thus, under these restrictions, the computation of top-k objects can be transformed to the subproblems each of which computes the volume of a high-dimensional polyhedron represented by a system of linear inequations.

Given an algorithm for the computation of volumes of a high dimensional polyhedron represented by a system of linear inequations, we can apply it to compute top-k objects.

The computation of the volume of a high-dimensional polyhedron represented by a system of linear inequalities has been studied in [61],[1] in which the authors divide the polyhedron into simplices and adds their individual volumes. Here we present an algorithm, which is adopted from one in [61], to compute the volume of a a high dimensional polyhedron represented by a system of linear inequalities. The pseudo code for the algorithm is given in Figures 7.7 and 7.8.

Comments on Algorithm 1:

Let Q be a set of vertices with distinct subscripts. $v_i \in Q$ is said to be *minimally indexed* w.r.t. Q if for any $v_j \in Q$, $j \geq i$. Denote $\eta(Q)$ the *minimally indexed* vertex. Similarly for *maximally indexed* w.r.t. Q, and denote it by $\beta(Q)$.

Given the set of vertices of a simplex $S = \{v_{u_0}, v_{u_1}, ..., v_{u_w}\}$, the formula to compute the volume is

$$\frac{|Det(v_{u_1} - v_{u_0}, v_{u_2} - v_{u_0}, ..., v_{u_n} - v_{u_0})|}{w!}$$

where v_{ui} is the coordinate of the vertex.

Algorithm 1 is adopted from the exact algorithm given in [61] (as compared to an approximation algorithm). We made some adjustments to make it suite our problem. The

[1]This paper is one of the early ones in this area of study, and many more improved algorithms have been published since then. Our goal in this chapter is to focus on the relation between computing top-k objects and computing the volumes of the corresponding polyhedra. While we address the logical connections, we are not concerned with the efficiency issues.

Algorithm 1: Computing Volume of Polyhedron

Input: The collection of inequalities, expressed by

$$Ax \geq B$$

in which A ann B are real arrays with dimension of the matrix $A[\omega, n \times m]$ and $B[\omega]$.

Output: Volume of the polyhedron represented by $Ax \geq B$

Associate G_i with the ith inequality, initially set to $\emptyset$
H is a set of vertices, initially set to $\emptyset$
V is a real number, initially set to 0
p is an integer (intended to index vertices), initially set to 1

For each subsystem of $n \times m$ inequalities A', B' of $Ax \geq B$
  If $A'x = B'$ has a unique solution v_p satisfying $Ax \geq B$
    then
      $H = H \cup \{v_p\}$
      $p = p + 1$
      For each inequality $A_i x \geq B_i$ contained
        in $A'x \geq B'$
          $G_i = G_i \cup \{v_p\}$

For each G_i
  If $|G_i| < n \times m$ **then** remove G_i
Re-index remaining G_i's to F_1, ...,F_u

Initialize $S = \{v_1\}$
Call $Vol(n \times m - 1, H, S)$
Return V

Figure 7.7: The main algorithm

---

algorithm is divided into two stages. The first stage is to get the set of all the vertices of the polyhedron represented by the given system of linear inequations and the set of vertices of each face of the polyhedron. The second stage is to use these sets of vertices to divide the polyhedron into simplices and add all the simplices' volumes together.

The input to the algorithm is a set of inequations, denoted by a matrix $Ax \geq B$. We assume there are ω inequations. Then we associate a vertex set G_i to each inequation in the system of linear inequalities. To compute the vertex set of the polyhedron, for each subsystem A', B' of $Ax \geq B$ having $n \times m$ rows, we attempt to generate a solution that satisfies $A'x = B'$. If there is such a solution and it satisfy $Ax \geq B$, then this solution consists of the coordinates of one vertex of the polyhedron. Each inequation in $Ax \geq B$

Procedure $Vol(d, last, S)$

$L = \emptyset$
if $d > 0$ **then**
  for each F_i **do**
    $l = F_i \cap last$
    if $l \notin L$ **then**
      $L = \{l\} \cup L$
      if $\eta(l) \notin S$ **then**
        $Vol(d-1, l, S \cup \{\eta(l)\})$
else
  if $\beta(last) \notin S$ **then**
    $S = S \cup \{\beta(last)\}$
    $V = V + |Det$(coordinates of the vertices in S)$|/(n \times m)!$

Figure 7.8: Computing volume given vertices

---

corresponds to a hyperplane in the $n \times m$-dimensional space. Some hyperplanes are the faces of the polyhedron and some are not. If a hyperplane is a face of the polyhedron, the vertex set corresponds to it contains at least $n \times m$ vertices. So we can check the number of vertices in each G_i and remove any G_i which contains less than $n \times m$ vertices. Then we reindex the remaining G_i to F_j, which is the set of vertices of a face of the polyhedron.

After getting the set of all the vertices of the polyhedron H and the set of the vertices of each face of the polyhedron F_i, we call the procedure Vol to compute the volume of the polyhedron. Given H and each F_i, the authors of [61] prove that the procedure Vol divides the polyhedron into simplices and adds the volumes of all simplices together to get the volume of the polyhedron.

The time complexity of Algorithm 1 is $O(\omega^{n\times m+1})$. We assume F contains p linear inequations. To compute the volume of support space of one object, we need to compute the volumes of C_{n-1}^{n-k} different polyhedra represented by systems of linear inequations. Then we add them together to get the volume of the support space of an object. For each system of linear inequations, it contains $n - k + p + 2n \times m$ inequations. So the time complexity to compute the volume of the support space of an object is $O(C_{n-1}^{n-k}(n - k + p + 2n \times m)^{n\times m+1})$. The total time complexity to compute top-k objects is $O(nC_{n-1}^{n-k}(n - k + p + 2n \times m)^{n\times m+1})$. C_{n-1}^{n-k} is a polynomial in $n - 1$ of degree $k - 1$. So the total time complexity is $O(n(n-1)^{k-1}(n - k + p + 2n \times m)^{n\times m+1})$.

Example 7.5.1 We show an example to illustrate how to use Algorithm 1 to compute the

volume of a polyhedron represented by a system of linear inequalities. Assume a system of linear inequalities as below:

$$0 \leq x_{ij} \leq 1 \qquad 1 \leq i \leq 2, 1 \leq j \leq 2$$
$$x_{11} \geq x_{21}$$
$$x_{11} + x_{12} \geq x_{21} + x_{22}$$

Now we use Algorithm 1 to compute the volume of the polyhedron represented by these inequations. First we generate all the vertices:

$$v_1 = (0,0,0,0) \quad v_2 = (0,0,1,0) \quad v_3 = (0,0,1,1)$$
$$v_4 = (1,0,0,0) \quad v_5 = (1,0,0,1) \quad v_6 = (1,0,1,0)$$
$$v_7 = (1,0,1,1) \quad v_8 = (1,1,0,0) \quad v_9 = (1,1,1,0)$$
$$v_{10} = (1,1,1,1)$$

Then we generate F_i as follows:

$$F_1 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$$
$$F_2 = \{v_1, v_4, v_5, v_8\}$$
$$F_3 = \{v_1, v_2, v_4, v_6, v_8, v_9\}$$
$$F_4 = \{v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$$
$$F_5 = \{v_2, v_3, v_6, v_7, v_9, v_{10}\}$$
$$F_6 = \{v_3, v_5, v_7, v_{10}\}$$
$$F_7 = \{v_1, v_2, v_3, v_8, v_9, v_{10}\}$$
$$F_8 = \{v_1, v_3, v_5, v_8, v_{10}\}$$

Then the procedure Vol computes the volume as: $9/24 = 0.375$.

Now we show an example of how to compute top-k objects.

Example 7.5.2 *Let $D = \langle O, A, X, P, F \rangle$ be a database, with $O = \{o_1, o_2, o_3\}$, $A = \{a_1, a_2\}$, $X = \{x_{ij} \mid 1 \leq i \leq 3, 1 \leq j \leq 2\}$, and $F = \{x_{11} \geq x_{21}\}$ (i.e., the value of o_1 under a_1 is equal or greater than o_2 under a_1). Assume the aggregation function is $t(x_{i1}, x_{i2}) = x_{i1} + x_{i2}$ and the domain of each variable is $[0, 1]$. We want to find top-2 objects.*

The support space of object o_1 composes of two parts, which are the high-dimensional polyhedra represented by the following two systems of linear inequalities, respectively:

$$0 \leq x_{ij} \leq 1 \qquad 1 \leq i \leq 3, 1 \leq j \leq 2$$
$$x_{11} \geq x_{21}$$
$$x_{11} + x_{12} \geq x_{21} + x_{22}$$

and

$$0 \leq x_{ij} \leq 1 \qquad 1 \leq i \leq 3, 1 \leq j \leq 2$$
$$x_{11} \geq x_{21}$$
$$x_{11} + x_{12} \geq x_{31} + x_{32}$$

We can find that the maximal dimension of all the support spaces to objects in D is 6. The support space of o_1 is 6 dimension. So we just need to compute the volume of the support space of o_1. Then we can use an algorithm, for example, the one in [61], to compute the volume of a high-dimensional polyhedron represented by a system of inequations. The volume of the support space of object o_1 is the sum of the volumes of the two polyhedra. The support space of o_2 and o_3 is also 6 dimension. We can compute the volume of the support space of objects o_2 and o_3 similarly. Then the top-2 objects are the 2 objects with largest 2 volumes of support spaces.

7.5.2 Computing top-k objects with multiple attributes

As we introduced in Subsection 7.4.3, the top-k tuples defined in [56] can be thought of as a special case of our top-k objects defined in Section 7.3. So the polynomial time algorithm to compute top-k tuples in [56] (cf. Section 2.3) can be thought of as an algorithm to compute the top-k objects in our definition under some restrictions: there is only one attribute that needs to be ranked, the probability distribution of the values under this attribute is uniform or can be described by a piecewise polynomial function, and there are no constraints. Here we show that the number of attributes can be relaxed from one to two and after some processing, we can still use the polynomial time algorithm in [56] to compute the top-k objects.

We assume the aggregation function is a linear expression with non-negative parameters:

$$t(x, y) = aX + bY (a \geq 0, b \geq 0)$$

Actually, we only need to compute the probability distribution of $t(x, y)$ from the probability distributions of X and Y, then we can use the methods in [56] to compute the top-k objects in polynomial time. We show that the computation of the probability distribution of $t(x, y)$ from the probability distributions of X and Y takes polynomial time, so that the overall complexity of computing the top-k objects here is still polynomial. From our analysis, it is not difficult to see that the method of computing the probability distribution of t for two attributes can be extended to three attributes.

Simple Case

Here we assume an object has two numerical attributes which are considered in ranking. We assume X and Y to be the random variables to represent the values of an object under these two attributes. We assume X and Y are independent.

We denote $f_X(x)$ as the probability density function of X and $f_Y(y)$ as the probability density function of Y. $f_X(x) > 0$ when $L_X \leq x \leq U_X$ and $f_X(x) = 0$ when $x < L_X$ or $x > U_X$. $f_Y(y) > 0$ when $L_Y \leq y \leq U_Y$ and $f_Y(y) = 0$ when $y < L_Y$ or $y > U_Y$. We assume $f_X(X)$ is composed of only one expression in $[L_X, U_X]$ and $f_Y(y)$ is only composed of one expression in $[L_Y, U_Y]$. We know that the joint probability density function of (X, Y), $f(x, y)$, is $f_X(x) \times f_Y(y)$. We assume the aggregation function

$$Z = aX + bY \ (a \geq 0, b \geq 0)$$

where Z is a function of X and Y.

Here we show how to compute the probability density function of Z. We compute the cumulative distribution function of Z first, then we compute the differential of the cumulative distribution function to get the probability density function. The cumulative distribution function of Z $F_Z(z)$ can be obtained as follows:

$$F_Z(z) = P(Z < z) = P(aX + bY < z) = \int\int_{aX+bY<z} f(x, y)dxdy \tag{7.6}$$

It is easy to see that $F_Z(z) = 0$ when $z < aL_X + bL_Y$ and $F_Z(z) = 1$ when $z > aU_X + bU_Y$.

In the 2-dimension space (x, y), $Z = aX + bY$ is a group of lines in which each line has a different value of Z and is parallel to other lines. We can think of a line as moving from bottom left to up right of the xy-plane with the increase of the value of Z. The line $Z = aX + bY$ passes through a point (x_i, y_i) when $z = ax_i + by_i$. For two points (x_1, y_1) and (x_2, y_2), if $ax_1 + by_1 > ax_2 + by_2$, the line $ax_1 + by_1 = aX + bY$ is in the up right of the line $ax_2 + by_2 = aX + bY$. We can also think a line $Z = aX + bY$ passes through the point (x_2, y_2) before the point (x_1, y_1) from bottom left to up right. For any point (x, y) in xy-plane, we will call $ax + by$ *the Z value* of this point.

Now we discuss how to get $F_Z(z)$ when $aL_X + bL_Y \leq z \leq aU_X + bU_Y$.

From equation (7.6), we know that we need to compute the integration of the probability density function $f(x, y)$ to get the cumulative distribution function of Z. We need to specify the integration area.

If the line $Z = aX + bY$ passes through the point (L_X, U_Y) before (or at the same time) the point (U_X, L_Y) ($aL_X + bU_Y \leq aU_X + bL_Y$), we discuss the cumulative distribution function in the following three cases.

When $aL_X + bL_Y \leq z \leq aL_X + bU_Y$, the integration area is a triangle enclosed by the lines $Z = aX + bY$, $X = L_X$ and $Y = L_Y$. So we have

$$F_Z(z) = \int_{L_X}^{\frac{z-bL_Y}{a}} dx \int_{L_Y}^{\frac{z-ax}{b}} f(x,y)dy$$

When $aL_X + bU_Y < z \leq aU_X + bL_Y$, the integration area is a trapezoid enclosed by the lines $Z = aX + bY$, $Y = U_Y$, $X = L_X$ and $Y = L_Y$. We divide this integration area into a rectangle enclosed by the lines $Y = U_Y$, $X = L_X$, $Y = L_Y$, $X = \frac{z-bU_Y}{a}$ and a triangle enclosed by the lines $X = \frac{z-bU_Y}{a}$, $Y = L_Y$, $Z = aX + bY$. So we have

$$F_Z(z) = \int_{L_X}^{\frac{z-bU_Y}{a}} dx \int_{L_Y}^{U_Y} f(x,y)dy + \int_{\frac{z-bU_Y}{a}}^{\frac{z-bL_Y}{a}} dx \int_{L_Y}^{\frac{z-ax}{b}} f(x,y)dy$$

When $aU_X+bL_Y < z \leq aU_X+bU_Y$, the integration area is a pentagon enclosed by the line $Z = aX + bY$, $Y = U_Y$, $X = L_X$, $Y = L_Y$ and $X = U_X$. We divide this integration area into a rectangle enclosed by the lines $Y = U_Y$, $X = L_X$, $Y = L_Y$, $X = \frac{z-bU_Y}{a}$ and a trapezoid enclosed by the lines $X = \frac{z-bU_Y}{a}$, $Y = L_Y$,$X = U_X$, $Z = aX+bY$. So we have

$$F_Z(z) = \int_{L_X}^{\frac{z-bU_Y}{a}} dx \int_{L_Y}^{U_Y} f(x,y)dy + \int_{\frac{z-bU_Y}{a}}^{U_X} dx \int_{L_Y}^{\frac{z-ax}{b}} f(x,y)dy$$

Similarly, if the line $Z = aX + bY$ passes through the point (U_X, L_Y) before the point (L_X, U_Y) ($aU_X + bL_Y < aL_X + bU_Y$), we can also discuss the cumulative distribution function in the following three cases.

When $aL_X + bL_Y \leq z \leq aU_X + bL_Y$, we have

$$F_Z(z) = \int_{L_X}^{\frac{z-bL_Y}{a}} dx \int_{L_Y}^{\frac{z-ax}{b}} f(x,y)dy$$

When $aU_X + bL_Y < z \leq aL_X + bU_Y$, we have

$$F_Z(z) = \int_{L_X}^{U_X} dx \int_{L_Y}^{\frac{z-ax}{b}} f(x,y)dy$$

When $aL_X + bU_Y < z \leq aU_X + bU_Y$, we have

$$F_Z(z) = \int_{L_X}^{\frac{z-bU_Y}{a}} dx \int_{L_Y}^{U_Y} f(x,y)dy + \int_{\frac{z-bU_Y}{a}}^{U_X} dx \int_{L_Y}^{\frac{z-ax}{b}} f(x,y)dy$$

So we get the cumulative distribution function of Z, namely $F_Z(z)$, as a piecewise function. After computing the differential of $F_Z(z)$, we get the probability density function of Z.

From the computational procedure above, we can see that the cost of computing the probability density function consists of some integrations and differentiations. The costs

of integrations or differentiations depend on the joint probability density function $f(x, y)$. Recall that $f(x, y) = f_X(x) \times f_Y(y)$. So the costs depend on the probability density functions $f_X(x)$ and $f_Y(y)$.

Below, let us give an example to show the procedure of computing the probability distribution.

Example 7.5.3 *We assume the probability density function of X is*

$$f_X(x) = \begin{cases} 0 & \text{if } x < 0 \\ \frac{1}{2}x & \text{if } 0 \leq x \leq 2 \\ 0 & \text{if } x > 2 \end{cases}$$

and the probability density function of Y is

$$f_Y(y) = \begin{cases} 0 & \text{if } y < 0 \\ 2y & \text{if } 0 \leq y \leq 1 \\ 0 & \text{if } y > 1 \end{cases}$$

Let the aggregation function be $Z = X + Y$. We know that $L_X = 0$, $U_X = 2$, $L_Y = 0$, and $U_Y = 1$. We also know that $Z = X + Y$ will pass (L_X, U_Y) first.

When $0 \leq Z \leq 1$, we have

$$F_Z(z) = \int_0^z dx \int_0^{z-x} xydy = \frac{1}{24}z^4$$

When $1 < z \leq 2$, we have

$$F_Z(z) = \int_0^{z-1} dx \int_0^1 xydy + \int_{z-1}^z dx \int_0^{z-x} xydy = \frac{1}{4}z^2 - \frac{1}{3}z + \frac{1}{8}$$

When $2 < z \leq 3$, we have

$$F_Z(z) = \int_0^{z-1} dx \int_0^1 xydy + \int_{z-1}^2 dx \int_0^{z-x} xydy = -\frac{1}{24}z^4 + \frac{5}{4}z^2 - 3z + \frac{17}{8}$$

And we also know that, When $z < 0$ we have

$$F_Z(z) = 0$$

When $z > 3$, we have

$$F_Z(z) = 3$$

After calculating the differentials of $F_Z(z)$, we get the probability density function as follows:

$$f_Z(z) = \begin{cases} 0 & \text{if } z < 0 \\ \frac{1}{6}z^3 & \text{if } 0 \leq z \leq 1 \\ \frac{1}{2}z - \frac{1}{3} & \text{if } 1 < z \leq 2 \\ -\frac{1}{6}z^3 + \frac{5}{2}z - 3 & \text{if } 2 < z \leq 3 \\ 0 & \text{if } z > 3 \end{cases}$$

Complex Case

Now we assume $f_X(x)$ and $f_Y(y)$ are piecewise functions. $f_X(x) > 0$ when $L_{X_0} \leq x \leq U_{X_p}(p > 0)$ and $f_X(x) = 0$ when $x < L_{X_0}$ or $x > U_{X_p}$. $f_Y(y) > 0$ when $L_{Y_0} \leq y \leq U_{Y_q}(q > 0)$, and $f_Y(y) = 0$ when $y < L_{Y_0}$ or $y > U_{Y_q}$. We assume $[L_{X_0}, L_{X_p}]$ is divided into p parts:

$$[L_{X_0}, L_{X_1}], [L_{X_1}, L_{X_2}], ..., [L_{X_{p-1}}, L_{X_p}]$$

The piecewise function of $f_X(x)$ in $[L_{X_{i-1}}, L_{X_i}]$ is $f_{X_i}(x)$. We assume $[L_{Y_0}, L_{Y_q}]$ is divided into q parts:

$$[L_{Y_0}, L_{Y_1}], [L_{Y_1}, L_{Y_2}], ..., [L_{Y_{q-1}}, L_{Y_q}]$$

The piecewise function of $f_Y(y)$ in $[L_{Y_{i-1}}, L_{Y_i}]$ is $f_{Y_i}(y)$. Then $f(x, y)$ is also piecewise function. In the domain

$$L_{X_{i-1}} \leq x \leq L_{X_i}, L_{Y_{j-1}} \leq y \leq L_{Y_j} \ (1 \leq i \leq p, 1 \leq j \leq q)$$

the piece expression of $f(x, y)$ is $f_{X_i}(x) \times f_{Y_j}(y)$.

We keep the other assumptions the same as for the simple case above. Now we show how to compute the probability density function of Z.

Again, we first compute the cumulative distribution function of Z. Then we calculate the differential of the cumulative distribution function of Z to get the probability density function of Z. The formula in (7.6) can still be used to calculate the cumulative distribution function of Z. But now we need to discuss the cumulative distribution function in more cases.

When $z < aL_{X_0} + bL_{Y_0}$, $F_Z(z) = 0$. When $z > aL_{X_p} + bL_{Y_q}$, $F_Z(z) = 1$. Now we discuss how to get $F_Z(z)$ when $aL_{X_0} + bL_{Y_0} \leq z \leq aL_{X_p} + bL_{Y_q}$.

We notice the following points in the xy-plane: (L_{X_i}, L_{Y_j}) $(0 \leq i \leq p, 0 \leq j \leq q)$. We call these points *boundary points*. All these points form pq small rectangles. A small rectangle has four vertex:

$$(L_{X_i}, L_{Y_j}), (L_{X_{i+1}}, L_{Y_j}), (L_{X_{i+1}}, L_{Y_{j+1}}), (L_{X_i}, L_{Y_{j+1}})$$

The integration area consists of some whole small rectangles and some parts of small rectangles. In each small rectangle in the integration area, we need to compute the integration of $f(x, y)$. Then we add all these together to get the cumulative distribution function of Z. The integration area will change with the moving of the line $Z = aX + bY$. When the line

is moving between two groups of boundary points (the points in each group have the same Z values), the resulting expression of the integration in each small rectangle (full or part) in the integration area will not change so that the cumulative distribution function will remain unchanged. When the line moves over a group of boundary points with the same Z values, the small rectangles in the integration area will be different so that some of the resulting expressions in some small rectangles will change. So the cumulative distribution function needs to be updated. One boundary point can be a vertex of at most 4 small rectangles. These small rectangle can be in the up right, up left, bottom left or bottom right of this point. We call these rectangles *related rectangles* to this point. If the rectangle is in the up right of the point, we will call it the *up right related rectangle* to this point. Similarly, we have the *up left related rectangle, bottom left related rectangle, bottom right related rectangle* to the point. When the line $Z = aX + bY$ moves over this point, the integrations in related rectangles to this point need to be recomputed. Because the cumulative distribution function of Z is the sum of the resulting expression of the integration in the small rectangles in the integration area, we need to update the cumulative distribution function.

We use a two dimensional array M to store the boundary points. Assume there are r different values among all the Z values of these points (some points may have the same Z values) . Here $r \leq (p+1)(q+1)$. Let us order the boundary points according to their Z values, and let the array E store all these values in an increasing order. Each line $M[u](0 \leq u \leq r-1)$ in M stores all the points that have the Z value $E[u]$. If $u_1 < u_2$, the Z values of the points in $M[u_1]$ is smaller than the Z values of the points in $M[u_2]$. In each line $M[u]$ of M, the points are sorted according to the increasing order of the x-coordinate values of the points.

We use a two dimension array W to store the resulting expressions of the integrations in all the small rectangles. When the bottom left vertex of a small rectangle is (L_{X_i}, L_{Y_j}) $(0 \leq i \leq p-1, 0 \leq j \leq q-1)$, we use $W[i][j]$ to store the resulting expression of the integration in this small rectangle.

We use an array S to store the piece expressions of the cumulative distribution function. There are $r-2$ elements in S. The expression $S[i]$ $(0 \leq i \leq r-2)$ is the piece expression of the cumulative distribution function in the domain $E[i] \leq z < E[i+1]$

We will visit the points in M one by one, and visit $M[u]$ from smaller u to lager u. In each line $M[u]$, we will visit $M[u][v]$ from smaller v to larger v. Before visiting any point in M, we set each element in W and S empty.

When a point A is visited, we assume it is stored in $M[u][v]$ and its coordinate is

(L_{X_i}, L_{Y_j}). For this point, we check all its related rectangles and do some operations.

For the up right related rectangle to A (if it exists), we calculate the following integration:

$$F_Z(z) = \int_{L_{X_i}}^{\frac{z-bL_{Y_j}}{a}} dx \int_{L_{Y_j}}^{\frac{z-ax}{b}} f(x,y)dy$$

Here $f(x,y) = f_{X_{i+1}}(x) \times f_{Y_{j+1}}(y)$. Then we store the resulting expression of this integration in $W[i][j]$. And we add this expression into $S[u]$.

For the bottom left related rectangle to A (if it exists), we calculate the following integration:

$$F_Z(z) = \int_{L_{X_{i-1}}}^{L_{X_i}} dx \int_{L_{Y_{j-1}}}^{Y_j} f(x,y)dy$$

Here $f(x,y) = f_{X_i}(x) \times f_{Y_j}(y)$. We cut the expression in $M[i-1][j-1]$ from $S[u]$. Then we store the new expression in $M[i-1][j-1]$ and add it into $S[u]$.

For the bottom right related rectangle to A (if it exists), we need to handle it in two different situations.

If $aL_{X_{i+1}} + bL_{Y_{j-1}} \geq aL_{X_i} + bL_{Y_j}$, we calculate the following integration:

$$F_Z(z) = \int_{L_{X_i}}^{\frac{z-bL_{Y_j}}{a}} dx \int_{L_{Y_{j-1}}}^{L_{Y_j}} f(x,y)dy + \int_{\frac{z-bL_{Y_j}}{a}}^{\frac{z-bL_{Y_{j-1}}}{a}} dx \int_{L_{Y_{j-1}}}^{\frac{z-ax}{b}} f(x,y)dy$$

If $aL_{X_{i+1}} + bL_{Y_{j-1}} < aL_{X_i} + bL_{Y_j}$, we calculate the following integration:

$$F_Z(z) = \int_{L_{X_i}}^{\frac{z-bL_{Y_j}}{a}} dx \int_{L_{Y_{j-1}}}^{L_{Y_j}} f(x,y)dy + \int_{\frac{z-bL_{Y_j}}{a}}^{L_{X_{j+1}}} dx \int_{L_{Y_{j-1}}}^{\frac{z-ax}{b}} f(x,y)dy$$

In both of the two situations above, $f(x,y) = f_{X_{i+1}}(x) \times f_{Y_j}(y)$. We cut the expression in $M[i][j-1]$ from $S[u]$. Then we store the new expression in $M[i][j-1]$ and add it into $S[u]$.

For the up left related rectangle to A (if it exists), we need to handle it in two different situations.

If $aL_{X_{i-1}} + bL_{Y_{j+1}} > aL_{X_i} + bL_{Y_j}$, we calculate the following integration:

$$F_Z(z) = \int_{L_{X_{i-1}}}^{L_{X_i}} dx \int_{L_{Y_j}}^{\frac{z-ax}{b}} f(x,y)dy$$

If $aL_{X_{i-1}} + bL_{Y_{j+1}} < aL_{X_i} + bL_{Y_j}$, we calculate the following integration:

$$F_Z(z) = \int_{L_{X_{i-1}}}^{\frac{z-bL_{Y_{j+1}}}{a}} dx \int_{L_{Y_j}}^{L_{Y_{j+1}}} f(x,y)dy + \int_{\frac{z-bL_{Y_{j+1}}}{a}}^{L_{X_j}} dx \int_{L_{Y_j}}^{\frac{z-ax}{b}} f(x,y)dy$$

In both of the two situations above, $f(x,y) = f_{X_i}(x) \times f_{Y_{j+1}}(y)$. We cut the expression in $M[i-1][j]$ from $S[u]$. Then we store the new expression in $M[i-1][j]$ and add it into $S[u]$.

After all the points in M have been visited and operated, S and E store $r-1$ pieces of expressions of the cumulative distribution function of Z and the domains. The other 2 pieces of expressions of the the cumulative distribution function of Z and the domains are: $F_Z(z) = 0$ when $z < aL_{X_0} + bL_{Y_0}$ and $F_Z(z) = 1$ when $z > aL_{X_p} + bL_{Y_q}$. So we have the cumulative distribution function of Z. Then we calculate the differential of the cumulative distribution function of Z to get the probability density function of Z.

In our method to compute probability density function of Z, the probability density functions of X and Y could be different functions. Here we restrict the probability density functions of X and Y to be piecewise polynomials and give the complexity analysis. We assume there are n objects. And we assume γ is the maximum degree among all the piecewise polynomials of the probability density functions of values of all the objects under any attribute. It costs $O(\gamma^3)$ to compute the probability density function of the aggregation value of an object. For the simple case, the time complexity to compute the probability density functions of the aggregation values of all the objects is $O(n\gamma^3)$. For the complex case, we assume p is the maximum pieces of all the piecewise polynomials of the probability density functions of values of all the objects under any attribute. Then the time complexity to compute the probability density functions of the aggregation values of all the objects is $O(np^2\gamma^3)$. As we introduced in Subsection 7.4.3, the algorithm in [56] can compute the top-k objects with one attribute when we know the probability density functions of values of all the objects under this attribute. When the probability density functions are piecewise polynomials, the algorithm can find top-k objects in polynomial time. The time complexity is introduced in Subsection 7.4.3. After we get the probability density functions of the aggregation values of all the objects, we can use the algorithm in [56] to compute top-k objects. So the total time complexity to compute top-k objects with two attributes is still polynomial.

Example 7.5.4 *We assume the probability density function of X is*

$$f_X(x) = \begin{cases} 0 & \text{if } x < 0 \\ \frac{1}{4}x & \text{if } 0 \le x \le 2 \\ 1 - \frac{1}{4}x & \text{if } 2 < x \le 4 \\ 0 & \text{if } x > 4 \end{cases}$$

and the probability density function of Y is

$$f_Y(y) = \begin{cases} 0 & \text{if } y < 0 \\ y & \text{if } 0 \le y \le 1 \\ 2 - y & \text{if } 1 < y \le 2 \\ 0 & \text{if } y > 2 \end{cases}$$

Let the aggregation function be $Z = X + Y$. We know that $L_{X_0} = 0$, $L_{X_1} = 2$,$L_{X_2} = 4$, and $L_{Y_0} = 0$, $L_{Y_1} = 1$, $L_{Y_2} = 2$. There are 9 boundary points :

$$(0,0), (0,1), (0,2), (2,0), (2,1), (2,2), (4,0), (4,1), (4,2)$$

There are 7 different Z values for these points. M is as follows:

$$\begin{array}{ll} M[0][0] = (0,0) & \\ M[1][0] = (0,1) & \\ M[2][0] = (0,2) & M[2][1] = (2,0) \\ M[3][0] = (2,1) & \\ M[4][0] = (2,2) & M[4][1] = (4,0) \\ M[5][0] = (4,1) & \\ M[6][0] = (4,2) & \end{array}$$

E is as follows:

$$\begin{array}{l} E[0] = 0 \\ E[1] = 1 \\ E[2] = 2 \\ E[3] = 3 \\ E[4] = 4 \\ E[5] = 5 \\ E[6] = 6 \end{array}$$

We set every element in W and S empty, and visit the points in M one by one. We start the visiting from $M[0][0] = (0,0)$. There is only up right related rectangle to $(0,0)$. We have

$$F_Z(z) = \int_0^z dx \int_0^{z-x} \frac{1}{4} xy dy = \frac{1}{96} z^4$$

We store $F_Z(z)$ in $W[0][0]$ and add it into $S[0]$. So

$$W[0][0] = \frac{1}{96} z^4$$

and

$$S[0] = \frac{1}{96} z^4$$

.

Then we visit $M[1][0] = (0, 1)$. *For the up right related rectangle to* $(0, 1)$, *we have*

$$F_Z(z) = \int_0^{z-1} dx \int_1^{z-x} \frac{1}{4}x(2-y)dy = -\frac{1}{96}z^4 + \frac{1}{12}z^3 - \frac{3}{16}z^2 + \frac{1}{24}z + \frac{1}{96}$$

We store $F_Z(z)$ *in* $W[0][1]$ *and add it into* $S[1]$. *So*

$$W[0][1] = -\frac{1}{96}z^4 + \frac{1}{12}z^3 - \frac{3}{16}z^2 + \frac{1}{24}z + \frac{1}{96}$$

and

$$S[1] = -\frac{1}{96}z^4 + \frac{1}{12}z^3 - \frac{3}{16}z^2 + \frac{1}{24}z + \frac{1}{96}$$

For the bottom right related rectangle to $(0, 1)$, *we have*

$$F_Z(z) = \int_0^{z-1} dx \int_0^1 \frac{1}{4}xydy + \int_{z-1}^{z} dx \int_0^{z-1} \frac{1}{4}xydy = \frac{1}{8}z^3 - \frac{1}{4}z^2 + \frac{1}{8}z$$

We cut $W[0][0]$ *from* $S[1]$. *So we have*

$$S[1] = S[1] - W[0][0] = -\frac{1}{48}z^4 + \frac{1}{12}z^3 - \frac{3}{16}z^2 + \frac{1}{24}z + \frac{1}{96}$$

Then we store $F_Z(z)$ *into* $W[0][0]$ *and add it into* $S[1]$. *So we have*

$$W[0][0] = \frac{1}{8}z^3 - \frac{1}{4}z^2 + \frac{1}{8}z$$

and

$$S[1] = S[1] + W[0][0] = -\frac{1}{48}z^4 + \frac{5}{24}z^3 - \frac{7}{16}z^2 + \frac{1}{6}z + \frac{1}{96}$$

Similar to the procedure above, we will visit each point in M *and compute* $F_Z(z)$ *for each related rectangle. And we update* W *and* S. *After visiting all the points, we have 6 pieces of expressions of the cumulative distribution function of* Z *in* S. *We also know* $F_Z(z) = 0$ *when* $z < 0$, *and* $F_Z(z) = 1$ *when* $z > 6$. *Then we calculate the differential of* $F_Z(z)$ *to get the probability density function of* Z.

7.6 Summary

In this chapter, we propose a ranking theory for objects with uncertain data. We present this ranking theory in three stages. In the first stage, we consider discrete domains. In the second, we consider continuous domains, and in the third we consider different weights of positions and the aggregation values of objects into the result of final ranking. Our ranking theory is built on the basis of the concept of high-dimensional space in mathematics. In

comparison with the related work, we show that some top-k ranking semantics in the literature are just special cases of our ranking theory. Due to the high complexity of computation of top-k objects in this context, we conduct research on computation only for some special cases.

Chapter 8

Conclusion

This thesis addresses two general issues of extensive recent interests in top-k ranking with uncertain data:

- Given a ranking function, how to efficiently compute the top-k tuples, and
- What is top-k ranking for objects with multiple attributes.

To address the first question, we focus on pruning for recently proposed unifying framework of top-k ranking based on what are called parameterized ranking function (PRF), for uncertain databases under the x-tuple model. Our theoretical work reveals the insights in some fundamental properties of how tuples in an uncertain database are intimately related, in the context of possible worlds. A new representation of PRF value of a tuple is formulated, based on which we develop a generic upper bound method. Then, we show how practical pruning methods can be derived based on this generic method. Given a tuple set Q, our upper bound method is independent of the size of Q as long as there are at least two. This makes it possible to apply the method to pruning using more than two tuples. Our experiments show that these pruning methods are powerful, which can generate substantial performance gains in computing top-k tuples.

The second issue addressed in this thesis is a general notion of object ranking with uncertain data, where objects are modeled by probability distributions of their values in attributes and by constraints. In this thesis, we propose a ranking theory for object ranking in this context. A key concept of the strength of an object is its "supports" to be in top-k. More supports means more strength. We show that our theory can be viewed as a generalization of a number of ranking definitions in the literature. We also relate this theory of object ranking with the problem of computing volumes of high-dimensional polyhedra in mathematics

Here is a list of future work:

- The pruning methods developed in this thesis is under the x-tuple model for uncertain databases. But there exist more general data models for uncertain databases such as probabilistic and/xor tree model. We know that x-tuple model is a special case of probabilistic and/xor tree model. Actually, the parameterized ranking function is already defined for the probabilistic and/xor tree model. It would be interesting to see whether our methods can be extended to general data models for uncertain databases such as the probabilistic and/xor tree model. Such an extension is non-trivial.

- Parameterized Ranking Fucntions (PRFs) are defined under tuple-level uncertain model. Our pruning methods are developed under tuple-level uncertain model. The definition of PRF can be easily extended to attribute-level uncertain model. Our pruning methods should be able to be updated to improve the efficiency of computation of top-k tuples for PRF defintion under attribute-level uncertain model.

- In [56], the authors proposed parameterized ranking functions (PRFs) to rank tuples with uncertain score described by a continuous probability distribution. Actually, this PRF definition is extended from the PRF definition of the tuples with discrete values in [29]. The computation of top-k tuples for continuous PRF definition share some common characteristics with the computation of top-k tuples under the discrete PRF definition. In Chapter 3, we proposed pruning methods for the discrete PRF definition. One should be able to extend the pruning methods to improve the performance of the computation of top-k tuples under the continuous PRF definition.

- As introduced in Section 7.5.2, we show a polynomial algorithm to compute top-k objects when objects have two numerical attributes under consideration for ranking. We just need to compute the probability density function of the aggregation value of an object $Z = aX + bY$ from the probability distribution of X and Y. The method here can be extended to three attributes easily. When the number of attributes is more than three, the problem is one in high-dimensional space. Our method of computing the probability density function of the aggregation value of an object $Z = aX + bY$ for two attributes should be able to be extended to n attributes. But what is unclear is the mathematical means of modeling the process in a high-dimensional space. We believe it can be done.

Bibliography

[1] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 614–656, 2003.

[2] K. Chakrabarti, V. Ganti, J. Han, and D. Xin, "Ranking objects by exploiting relationships:computing top-k over aggregation," in *Proceedings of SIGMOD*, pp. 371–382, 2006.

[3] S. Chaudhuri, L. Gravano, and A. Marian, "Optimizing top-k selection queries over multimedia repositories," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 8, pp. 992–1009, 2004.

[4] P. Andritsos, A. Fuxman, and R.J.Miller, "Clean answers over dirty databases," in *Proceedings of ICDE*, p. 30, 2006.

[5] R. Cheng, D.V.Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1112–1127, 2004.

[6] J. Considine, F. Li, G.Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *Proceedings of ICDE*, pp. 449–460, 2004.

[7] V. de Almeida and R. Hartmut, "Supporting uncertainty in moving objects in network databases," in *ACM International Workshops on Geographic Information Systems*, pp. 31–40, 2005.

[8] A. Fuxman, E. Fazli, and R. Miller, "Conquer: Efficient management of inconsistent databases," in *Proceedings of SIGMOD*, pp. 155–166, 2005.

[9] A. Silberstein, R. Braynard, C. Ellis, and K. Munagala, "A sampling-based approach to optimizing top-k queries in sensor networks," in *Proceedings of ICDE*, p. 68, 2006.

[10] N. Fuhr and T. Rolleke, "A probabilistic relational algebra for the integration of information retrieval and database systems," *ACM Transactions on Information System*, vol. 15, no. 1, pp. 32–66, 1997.

[11] X. L. Dong, A. Y. Halevy, and C. Yu, "Data integration with uncertainty," in *Proceedings of VLDB*, pp. 687–698, 2007.

[12] T.S.Jayram, R.Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu, "Avatar information extraction system," *IEEE Data Engineering Bulletin*, vol. 29, no. 1, pp. 40–48, 2006.

[13] R. Gupta and S. Sarawagi, "Creating probabilistic databases from information extraction models," in *Proceedings of VLDB*, pp. 965–976, 2006.

[14] E. Adar and C. Re, "Managing uncertainty in social networks," *IEEE Data Engineering Bulletin*, vol. 30, no. 2, pp. 15–22, 2007.

[15] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," in *Proceedings of VLDB*, pp. 864–875, 2004.

[16] J. Widom, "Trio: A system for integrated management of data,accuracy, and lineage," in *Proceedings of CIDR*, pp. 262–276, 2005.

[17] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *Proceedings of SIGMOD*, pp. 551–562, 2003.

[18] P. Sen, A. Deshpande, and L. Getoor, "Prdb: Managing and exploiting rich correlations in probabilistic databases," *VLDB Jounal*, vol. 18, no. 5, pp. 1065–1090, 2009.

[19] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," in *Proceedings of SIGMOD*, pp. 34–48, 1987.

[20] D. Barbara, H. Garcia-Molina, and D. Porter, "The management of probabilistic data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 5, pp. 487–502, 1992.

[21] N. Dalvi and D. Suciu, "Management of probabilistic data: foundations and challenges," in *Proceedings of PODS*, pp. 1–12, 2007.

[22] N. Friedman, L. Getoor, D. Koller, and A. Preffer, "Learning probabilistic relational models," in *Proceedings of IJCAI*, pp. 1300–1309, 1999.

[23] N. Fuhr, "A probabilistic framework for vague queries and imprecise information in databases," in *Proceedings of VLDB*, pp. 696–707, 1990.

[24] T. Imielinski and J. W. Lipski, "Incomplete information in relational databases," *JACM*, vol. 31, no. 4, pp. 761–791, 1984.

[25] 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, pp. 419–469, 1997.

[26] A. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working models for uncertain data," in *Proceedings of ICDE*, p. 7, 2006.

[27] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Proceedings of VLDB*, pp. 15–26, 2007.

[28] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *Proceedings of SIGMOD*, pp. 1357–1364, 2008.

[29] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probablistic databases," in *Proceedings of VLDB*, pp. 502–513, 2009.

[30] M. Soliman, I. Ilyas, and K. Chang, "Top-k query processing in uncertain databases," in *Proceedings of ICDE*, pp. 896–905, 2007.

[31] 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*, vol. 20, no. 12, pp. 1699–1711, 2008.

[32] I. Ilyas, G. Beskales, and M. Soliman, "A survey of top-k query processing techniques in relational databases systems," *ACM Computing Surveys*, vol. 40, no. 4, 2008.

[33] J. Li and A. Deshpande, "Consensus answers for queries over probabilistic databases," in *Proceedings of PODS*, pp. 259–268, 2009.

[34] G. Cormode, F. Li, and K. Yi, "Semantics of ranking queries for probabilistic data and expected ranks," in *Proceedings of ICDE*, pp. 305–316, 2009.

[35] M. A. Soliman and I. F.Ilyas, "Ranking with uncertain scores," in *Proceedings of ICDE*, pp. 317–328, 2009.

[36] R. Agrawal, R. Rantzau, and E. Terzi, "Context-sensitive ranking," in *Proceedings of SIGMOD*, pp. 383–394, 2006.

[37] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks*, vol. 30, no. 1-7, pp. 107–117, 1998.

[38] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis, "Automated ranking of database query results," in *Proceedings of CIDR*, 2003.

[39] G. Amato, F. Rabitti, P. Savino, and P. Zezula, "Region proximity in metric spaces and its use for approximate similarity search," *ACM Transactions on Information Systems*, vol. 21, no. 2, pp. 192–227, 2003.

[40] H. Bast, D. Majumdar, R. Scenkel, M. Theobald, and G. Weikum, "PIO-Top-K: Index-access optimized top-k query processing," in *Proceedings of VLDB*, pp. 475–486, 2006.

[41] C. Bohm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, 2001.

[42] K. C.-C. Chang and S. won Hwang, "Minimal probing: Supporting expensive predicates for top-k queries," in *Proceedings of SIGMOD*, pp. 346–357, 2002.

[43] P. Ciaccia and M. Patella, "Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces," in *Proceedings of ICDE*, pp. 244–255, 2000.

[44] G. Das and D. Gunopulos, "Answering top-k queries using views," in *Proceedings of VLDB*, pp. 451–462, 2006.

[45] A. P. de Vries, N. Mamoulis, N. Nes, and M. L. Kersten, "Efficient k-nn search on vertically decomposed data," in *Proceedings of SIGMOD*, pp. 322–333, 2002.

[46] M. Soliman, I. Ilyas, and K.-C. Chang, "Probabilistic top-k and ranking-aggregate queries," *ACM Transactions on Database Systems*, vol. 33, no. 3, pp. 1–54, 2008.

[47] B. Kanagal and A. Deshpande, "Online filtering, smoothing and probabilistic modeling of streaming data," in *Proceedings of ICDE*, pp. 1160–1169, 2008.

[48] Y. Tao, R. Cheng, X. Xiao, W. Ngai, B. Kao, and S.Prabhakar, "Indexing multi-dimensional uncertain data with arbitrary probability density functions," in *Proceedings of VLDB*, pp. 922–933, 2005.

[49] G. Beskales, M. Soliman, and I.F.Ilyas, "Efficient search for the top-k probable nearest neighbors in uncertain databases," in *Proceedings of VLDB*, pp. 326–339, 2008.

[50] V. Ljosa and A. Singh, "Apla: Indexing arbitrary probability distributions," in *Proceedings of ICDE*, pp. 946–955, 2007.

[51] V. Ljosa and A. Singh, "Top-k spatial joins of probabilistic objects," in *Proceedings of ICDE*, pp. 566–575, 2008.

[52] A. Deshpande, C. Guestrin, S. Madden, and J. Hellerstein, "Model-driven data acquisition in sensor networks," in *Proceedings of VLDB*, pp. 588–599, 2004.

[53] C. Bohm, A. Pryakhin, and M. Schubert, "The gauss-tree: Efficient object identification in databases of probabilistic feature vectors," in *Proceedings of ICDE*, p. 9, 2006.

[54] R. Cheng, J. Chen, and M. Mokbel, "Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data," in *Proceedings of ICDE*, pp. 973–982, 2008.

[55] T.Tran, L. Peng, B. Li, and Y. Diao, "Pods: a new model and processing algorithms for uncertain data streams," in *Proceedings of SIGMOD*, pp. 159–170, 2010.

[56] J. Li and A. Deshpande, "Ranking continous probabilistic datasets," *Proceedings of VLDB*, vol. 3, no. 1, pp. 638–649, 2010.

[57] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum, "Probabilistic ranking of database query results," in *Proceedings of VLDB*, pp. 888–899, 2004.

[58] K. Crammer and Y. Singer, "Pranking with ranking," pp. 641–647, 2001.

[59] K. Crammer and Y. Singer, "A family of additive online algorithms for category ranking," *Journal of Machine Learning Research*, vol. 3, pp. 1025–1058, 2003.

[60] W. W.Cohen, R. E.Schapire, and Y. Singer, "Learning to order things," *Journal of Artificial Intelligence Research*, vol. 10, pp. 243–270, 1999.

[61] J. Cohen and T. Hickey, "Two algorithms for determining volumes of convex polyhedra," *Journal of the ACM*, vol. 26, no. 3, pp. 401–414, 1979.