Uncertain Data Management

CHANG, Lijun

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Systems Engineering and Engineering Management

The Chinese University of Hong Kong
August 2011

Thesis Committee

Professor Hong Cheng (Chair)

Professor Jeffrey Xu Yu (Thesis Supervisor)

Professor Anthony Man-Cho So (Committee Member)

Professor Xuemin Lin (External Examiner)

To my parents, Qiyuan Chang and Yumei Huang, who brought me to the world.

ABSTRACT

Uncertain data management has received a lot of attentions recently due to the fact that data obtained can be incomplete or uncertain in many real applications. Ranking of uncertain data becomes an important research issue, the possible worlds semantics-based ranking makes it different from the ranking of deterministic data. In the traditional deterministic data, we can compute a score for each object, and then the objects are ranked based on the computed scores. However, in the scenario of uncertain data, each object has a probability to be the true answer (or the existence probability), besides the computed score. A probabilistic top-k ranking query ranks objects by the interplay of score and probability based on the possible worlds semantics. Many definitions have been proposed in the literature based on the possible worlds semantics.

In this thesis, we explore the issues of uncertain data management in several different aspects. First, we propose a novel linear time algorithm to compute the positional probability, the computation of which is a primitive operator for most of the ranking definitions. Our algorithm is based on the conditional probability formulation of positional probability and the system of linear equations. Based on the formulation of conditional probability, we also prove a tight upper bound of the top-k probability of tuples, which is then used to stop the top-k computation earlier. Second, we study top-k probabilistic ranking queries with joins when scores and probabilities are stored in different relations. We focus on reducing the join cost in probabilistic top-k ranking. We investigate two probabilistic score functions, namely, expected rank value and probability of highest ranking. We give upper/lower bounds of such probabilistic score functions in random access and sequential access, and propose new I/O efficient algorithms to find top-k objects. Third, we extend the possible worlds semantics to probabilistic XML ranking query, which is to rank top-k probabilities of the answers of a twig query in probabilistic XML data. The new challenge is how to compute top-k probabilities of answers of a twig query in probabilistic XML in the presence of containment (ancestor/descendant) relationships. We focus on node queries first, and propose a new dynamic programming algorithm which can compute top-k probabilities for the answers of node queries based on the previously computed results in probabilistic XML data. We further propose optimization techniques to share the computational cost. We also show techniques to support path queries and tree queries. Fourth, we study how to rank documents using a set of keywords, given a context that is associated with the documents. We model the problem using a graph with two different kinds of nodes (document nodes and multi-attribute nodes), where the edges between document nodes and multi-attribute nodes exist with some probability. We discuss its score function, cost function, and ranking with uncertainty. We also propose new algorithms to rank documents that are most related to the user-given keywords by integrating the context information.

摘要

在许多实际的应用中，所采集到的数据都是不确定性的，例如噪音或者误差所致。 随着处理技术的进步，现在越来越多的人直接对不确定性数据进行处理。 在所有对不确定性数据进行管理的方方面面中，对数据进行排序显得至关重要。 不管是在数据处理的中间过程中，还是最后选取特定元组返回给用户，都需要对数据进行排序。 由于不确定性数据所特有的概率特性，使得现有的确定性数据排序算法不能用于不确定性数据。 在传统的确定性数据排序中，我们只需要先对每个元组算出它的分数，然后按照分数排序即可。 然而在不确定性数据中，每个元组不但有一个分数，还有一个不确定性，而这个不确定性是由概率来表示的。 不确定性数据的top-k查询是基于可能世界模型的。 在这个模型中，每一个元组的概率是用来确定可能世界的概率， 而元组的分数是用来确定一个可能世界中不同元组之间的相对排列。在已有的研究中，基于可能世界模型， 人们提出了不同的排序定义。

在这篇论文中，我们从不同方面对不确定性数据的管理进行了研究。首先，我们提出了一个线性时间复杂度的算法来计算位置概率。 简单来说，位置概率就是一个元组在所有的可能世界中排在某一个特定位置的概率。值得注意的是， 现有的不确定性数据排序都需要先算出位置概率。所以我们的快速位置概率算法可以加快不确定性数据的排序。 我们算法的主要思想是条件概率和解线性方程组。基于条件概率，我们进一步证明了一个top-k概率的上界，而且这个上界是可达的。 基于这个上界，我们可以提前结束排序算法而不影响结果的正确性。 第二，我们考虑到基于某种原因，元组的分数和概率可能存在于两个数据库表格中。 我们研究不确定性数据的top-k查询，同时也考虑到数据库表格的联接。我们的研究方向主要是要避免对两个表格中的所有数据

进行联接操作。 为了这个目标，我们证明了两种排序定义中最后用于排序的不确定性分数的上下界。同时对于一个top-k查询，我们提出了外存算法。 基于这些上下界，我们的外存算法不需要读取两个表格中的所有数据。 第三，我们把这些基于可能世界模型的排序定义扩展到概率XML数据。 由于XML数据是半结构化数据，它是用一个树的模型表示的。两个元数据之间可能有祖先/子孙的关系，对排序算法提出了新的挑战。我们首先研究一个点的查询，并提出了新的动态规划算法来计算XML节点的top-k概率。基于XML的树型结构，我们进一步对动态规划算法进行了优化。同时我们也研究了路径查询和一般树查询。 最后，我们研究了上下文敏感的基于文档的关键字查询。在这种查询中，除了文档之外，我们还有上下文。而这种上下文是用图来表示的，每一个图节点上都附有文本属性。 我们建立一个图来表示所有的相关数据，即把每一个文档看成一个节点，它的属性就是文档的内容。然后把这些文档节点连接到上下文的图的节点，而这种连接是不确定性的。 在这种模型下，我们提出了打分函数，然后讨论了不确定性排序，并提出了新的算法。

Acknowledgements

It is almost impossible to express my gratitude to my adviser, Jeffrey Xu Yu, in a couple of words. First of all, he guided me to the world of academic research. Through his brilliant insights, and gentle guidance, he has helped me formulate the problems of this thesis and solve them. He has always been encouraging me to do the best I could. I have learnt many things about research and life from him. I have learnt that, it will be pleasant to do things if you really like it. His attitude towards work and life will influence me in my future.

I am very grateful to all my outstanding committee members, Hong Cheng, Anthony Man-Cho So, and Xuemin Lin. I would like to thank them for their help and comments on my research and thesis.

I would also like to thank my collaborator, Lu Qin, who shared many great ideas with me, and helped me a lot when I was a beginner in doing research. He has fought together with me through many days and nights to finalize conference paper submissions. Without his support, none of my achievements would be possible.

During my stay at CUHK, I have made some great friends, and life would not have been the same without them. I have had the pleasure to know Bo Chen and Di Wu. We have spent so much time together at the university gym. To Bo Chen, it was always wonderful to have your support in the hiking activities. To Di Wu, it was a pleasure to run with you. I would also like to thank all the people who shared my life in CUHK: Yang Zhou, Wenting Hou, Binyang Li, Zheng Liu, Jiefeng Cheng, Miao Qiao, Yuanyuan Zhu, Weiyang Liu, Lidong Bing, Zhiwei Zhang, Xin Huang, Ronghua Li, Xiaofeng Yu, Zhenglu Yang, Bo Hu, Qi Pan, Bo Jiang, Lanjun Zhou, and Ke Zhou. I remember our happy hours in exploring the beautiful places in Hong Kong country parks, to get to the top of innumerable hills, and to see the beautiful sandy beaches, reservoirs, and white-flower Derris. I also remember our happy times together every day, in the canteen, in the swimming pool, in the fitness room, and in the play ground. Finally, my special gratitude to the department of Systems Engineering and Engineering Management at CUHK, I feel extremely fortunate to have studied at such a wonderful place.

Most of all, I am grateful to my parents, Qiyuan Chang and Yumei Huang. It is your love and support that give me strength and bravery to overcome every difficulty I met in my PhD life.

CONTENTS

Abstract i

Abstract in Chinese iii

Acknowledgements v

Contents vii

List of Tables x

List of Figures xi

1. Introduction 1

1.1. x-Relation Model and Possible Worlds Semantics 4

1.2. Contributions 6

2. Background and Related Work 9

2.1. Uncertain Top-k Semantics 9

2.2. Uncertain Database Models 12

2.3. Join Query in Uncertain Database 13

2.4. Probabilistic XML 14

2.5. Top-k Queries in Deterministic Data 15

3. Fast Computing Positional Probability and Top-k Probability 17

3.1. Introduction 17

3.2. Existing Solution 18
3.3. A New Novel Algorithm 21
3.4. Top-k Computation 26
3.5. Performance Studies 29

4. Probabilistic Ranking over Relations **31**
4.1. Introduction 31
4.2. Problem Statement 33
4.3. Bounding Ranking Functions 36
4.3.1. Ranking Function R_E 37
4.3.2. Ranking Function P_{HR} 43
4.3.3. Ranking Function with Exclusive Relationship 46
R_E Function 47
P_{HR} Function 50
4.3.4. Discussions 51
4.4. I/O Efficiency 52
4.4.1. Random Access on Relation P 53
4.4.2. Sequential Access on Relation P 55
4.4.3. Sequential and Random Access 58
4.5. Performance Studies 59

5. Query Ranking in Probabilistic XML Data **66**
5.1. Introduction 66
5.2. PXML and PXML-RANK 68
5.3. Node Query 73
5.3.1. New Containment Issues 74
5.3.2. An Overview of Our Approach 76
5.3.3. An Example 79
5.3.4. Algorithms 84
5.4. Path Query and Tree Query 91

5.4.1. Discussions on Tree Query . . . 92

5.5. Performance Studies . . . 95

5.5.1. Test-DBLP . . . 97

5.5.2. Test-Mondial . . . 99

5.5.3. Test-XMark . . . 100

6. Context-Sensitive Document Ranking **102**

6.1. Introduction . . . 102

6.2. Problem Statement . . . 105

6.3. Related work . . . 107

6.4. The Score Function . . . 108

6.5. Ranking with Uncertainty . . . 112

6.6. Query Processing . . . 117

6.6.1. Local Context Generation . . . 118

6.7. Performance Studies . . . 120

6.7.1. Case Study for Query q_5 . . . 123

6.7.2. Case Study for Query q_6 . . . 125

6.7.3. Multi-attribute Graph . . . 127

7. Thesis Conclusion **128**

7.1. Future Directions . . . 129

Bibliography **130**

LIST OF TABLES

1.1. Temperature Readings from Sensor Network 2
1.2. Speed Monitoring in Traffic Network 2

3.1. Multi-alternative x-Relation 19
3.2. Parameters and Default Values 29

4.1. An Example Relation (SP) 33
4.2. Two Rankings 34
4.3. Two Relations (S and P) 35
4.4. $E[|I|] - R_E$ Bounds for Tuple o_i (independent) 40
4.5. Bounds of $R_E(o_i)$ in Ran/Seq Access 41
4.6. P_{HR} Bounds for Tuple o_i (independent) 44
4.7. Bounds of $P_{HR}(o_i)$ in Ran/Seq Access 45
4.8. $E[|I|] - R_E$ Bounds for Tuple o_i (independent/exclusive) 49
4.9. Upper/Lower Bounds in 3rd and 4th Iteration 57
4.10. Parameters for All Testings 60

5.1. Consecutive Computing $r^h_{v,j}$ for $h = \omega(e_4), \omega(e_5)$ 89
5.2. Queries Used for All Datasets 96
5.3. Parameters Used for Testing 97

6.1. Keyword Queries 122

LIST OF FIGURES

1.1. x-Relation Data 5

3.1. Computational Cost 25

3.2. Computing $p_{i,j}$ 30

4.1. R_E Bound Changes for o_3 and o_5 42

4.2. Real Data 61

4.3. Uniform Distribution 61

4.4. Normal Distribution 62

4.5. Positive Correlated 63

4.6. Negative Correlated 64

4.7. P_{HR} Testing 65

5.1. XML, PXML, and Twig Query 69

5.2. PXML Subtrees 70

5.3. List v.s. XML Tree 75

5.4. An Overview of Our Approach 76

5.5. Computing $p_{i,j}$ 78

5.6. An Example 80

5.7. Cut the Tree into Several Parts 82

5.8. Computing $r^h_{v,j}$ 87

5.9. Computing H-PROB (T_P, φ_{i+1}, k) 88

5.10. Convert a Node with l Children to a Complete Binary Tree 90

5.11. Handling Multiple Path Results in a Single Node 91
5.12. A PXML Tree 92
5.13. Compute top-k probabilities for a Subtree Answer 94
5.14. Testing DBLP Dataset 98
5.15. Testing Mondial Dataset 99
5.16. Testing XMark Datasets 101

6.1. Movie Reviews & Movie Records 103
6.2. Example 106
6.3. x-Relation Model 113
6.4. The System Architecture 117
6.5. Execution Time 123
6.6. Least Rank Position to Include the Top-k (1 to 10) Documents 124
6.7. (Discounted) Cumulative Gain for q_5 124
6.8. (Discounted) Cumulative Gain for q_6 125
6.9. Top-1 Document with Context 126
6.10. (Discounted) Cumulative Gain for q_5 and q_6 127

CHAPTER 1

INTRODUCTION

Uncertain data management is an import issue in sensor network, information extraction [68], data cleaning, data integration [13], and market decision making. Due to the nature of measuring and storage, most information obtained is either incomplete or uncertain. Directly modeling, storing, and analyzing uncertain data has regain the public interests from researchers during the last ten years. Among the different aspects of uncertain data management, ranking is of particular importance due to the nature of humans to make things in order. In this thesis, we focus our attentions to the ranking of uncertain data. First of all, we illustrate two motivating examples.

Example 1: In a sensor network deployed in a habitat, each sensor records and reports temperature of the surrounding environment. There are totally five sensors, $s_1, s_2, \cdots, s_5$, deployed in three locations, A, B, and C. Each sensor reading comes with a confidence value Conf., which is the confidence (or interpreted as probability) that the reading is valid. Tab. 1.1 shows the temperature sensor readings at a given sampling time. For example, for the first reading, sensor s_1 reports, at 02:12am, Feb. 4th 2009, that the temperature at location A is 24 degree Celsius. A question posted on the readings can be, "what are the two warmest locations?". □

Example 2: In a traffic-monitoring system, radars detect cars' speeds automatically. Several radars are deployed in different locations along several highways. The car identification (e.g., by plate number) is performed by a human operator or OCR of

RID	Loc.	Time	Sensor	Temp.	Conf.
R_1	A	04/02/09 02:12	s_1	24	1.0
R_2	A	06/02/09 02:12	s_1	30	1.0
R_3	B	05/02/09 04:13	s_2	29	0.6
R_4	B	05/02/09 04:15	s_3	26	0.4
R_5	C	06/02/09 03:16	s_4	28	0.5
R_6	C	06/02/09 03:20	s_5	27	0.5

Table 1.1: Temperature Readings from Sensor Network

plate number images. Tab. 1.2 shows a snapshot of the speed readings. The special attribute "Conf." in each tuple indicates the probability that the whole tuple gives correct information. For example, at 08:12, a radar records that the speed of the car with plate number X-321 is 120 km/h at location L_1. Based on the speed readings, we can rank the cars by their speeds. We can also pose a constraint to rank only the cars that have traveled between "08:00" and "09:59". □

TID	Loc.	Time	PlateNo	Speed	Conf.
T_1	L_1	08:12	X-321	120	0.7
T_2	L_1	08:32	Y-256	100	0.6
T_3	L_2	08:13	Y-256	70	0.4
T_4	L_3	09:05	X-224	80	0.4
T_5	L_4	08:56	X-224	90	0.5
T_6	L_4	10:20	W-341	102	1.0

Table 1.2: Speed Monitoring in Traffic Network

In the above two examples, although the score of a tuple (temperature in Example 1, and speed in Example 2) is deterministic, the tuples are probabilistic. There are multiple sources that contribute to data (or tuple) uncertainty. First of all, the instrument (sensor or radar) used to measure data may failure duo to technique or energy issues. Second, uncertainty may be involved during postprocessing of the raw data. For example, in the traffic monitoring system, the plate numbers are extracted from images by human operator or OCR, where the probability of getting the right plate number depends on the quality of the images.

In real applications, the probabilities of objects (or tuples) are computed based on particular applications. For example, in [68], Michelakis et al. provide a probabilistic framework for handling the uncertainty in rule-based information extraction. Specifically, for each extraction task, they build a parametric exponential model of uncertainty that captures the interaction between the different rules, where the exponential form follows from maximum-entropy considerations. Beskales et al. [13] model possible repairs in a duplicate detection problem as possible worlds of repair uncertain database. Each possible repair has a probability, which is determined by the parameter of a clustering algorithms, and the result of a query over the uncertain data is combined from query results over each possible repairs. We consider two real datasets in our performance studies, where the details of obtaining probability of tuples are discussed in Sec. 4.5 and Sec. 6.7.

Unlike the traditional deterministic data where ranking is based solely on the score, in uncertain data, each tuple has both a score and a probability. In order to rank uncertain data, both tuples' probabilities and scores need to be factored in the interpretation of ranking. This effectively introduces two interacting ranking dimensions, the interplay of which decides meaningful ranking semantics. For example, it is not meaningful to rank a top-scored tuple with insignificant probability as top-1 result. Also, it is not meaningful to rank a tuple with highest probability and low score as top result. Moreover, combining scores and probabilities into one measure will lose valuable information that can be used to get more meaningful answers conforming with the probabilistic query models as illustrated below.

1.1. x-Relation Model and Possible Worlds Semantics

In the literature of uncertain data modeling, a lot of uncertain database models have been proposed [3, 29, 7]. Among them, the x-Relation model is most popularly used in answering rank queries [3, 93]. In the x-Relation model, an x-Relation contains a set of independent x-tuples (called generation rules in [86, 45]). An x-tuple consists of a set of mutually exclusive tuples (or called alternatives) to represent a discrete probability distribution of the possible tuples the x-tuple may take in a randomly instantiated instance. In an x-tuple, each alternative t has a score $score(t)$, and a probability $p(t)$ that represents its existence probability over possible instances. In the x-Relation model, the alternatives of x-tuples are assumed to be disjoint. In the following, we denote an x-Relation as $\mathcal{X}$, an x-tuple as τ, and call an alternative a tuple, denoted as t.

Example 3: Fig. 1.1(a) shows an x-Relation which consists of three x-tuples, $\tau_1 = \{t_1, t_3\}$, $\tau_2 = \{t_2\}$, and $\tau_3 = \{t_4\}$. The x-tuple τ_1 indicates a probability distribution over t_1 and t_3, with probability $p(t_1) = 0.3$ for its true content to be t_1, with probability $p(t_3) = 0.5$ for its true content to be t_3, and with probability $1 - p(t_1) - p(t_3) = 0.2$ for none of t_1 and t_3 to be the true content. $\square$

The relations in Example 1 and Example 2 conform to the x-Relation model. Consider Tab. 1.1, tuples R_3 and R_4 form an x-tuple because they report different temperatures for location B at almost the same time. Therefore, the x-Relation of Tab. 1.1 consists of four x-tuples, $\{\tau'_1, \tau'_2, \tau'_3, \tau'_4\}$, with $\tau'_1 = \{R_1\}$, $\tau'_2 = \{R_2\}$, $\tau'_3 = \{R_3, R_4\}$, and $\tau'_4 = \{R_5, R_6\}$. For the speed readings in Tab. 1.2, the constraint is that the same car can not appear at two different locations within 20 minutes, then the x-Relation consists of four x-tuples, with $\tau''_1 = \{T_1\}$, $\tau''_2 = \{T_2, T_3\}$, $\tau''_3 = \{T_4, T_5\}$, and $\tau''_4 = \{T_6\}$.

Possible Worlds: In general, an x-Relation, $\mathcal{X}$, is a probability distribution over a set of possible instances $\{I_1, I_2, \cdots\}$. A possible instance, I_j, maintains zero or one alternative for every x-tuple $\tau \in \mathcal{X}$. The probability of an instance I_j, $\Pr(I_j)$, is

x-*tuple*	*tuple*	*score*	*prob*
τ_1	t_1	100	0.3
	t_3	80	0.5
τ_2	t_2	90	1.0
τ_3	t_4	70	0.8

(a) x-Relation

Possible world (I)	$\Pr(I)$	top-2
$\{t_2\}$	$(1-p(t_1)-p(t_3))p(t_2)(1-p(t_4)) = 0.04$	t_2
$\{t_2, t_4\}$	$(1-p(t_1)-p(t_3))p(t_2)p(t_4) = 0.16$	t_2, t_4
$\{t_1, t_2\}$	$p(t_1)p(t_2)(1-p(t_4)) = 0.06$	t_1, t_2
$\{t_1, t_2, t_4\}$	$p(t_1)p(t_2)p(t_4) = 0.24$	t_1, t_2
$\{t_2, t_3\}$	$p(t_3)p(t_2)(1-p(t_4)) = 0.10$	t_2, t_3
$\{t_2, t_3, t_4\}$	$p(t_3)p(t_2)p(t_4) = 0.40$	t_2, t_3

(b) Possible Worlds

Figure 1.1: x-Relation Data

the probability that x-tuples take certain or none alternative in I_j, such that $\Pr(I_j) = \prod_{t \in I_j} p(t) \times \prod_{\tau \notin I_j}(1 - \Pr(\tau))$ where $\tau \notin I_j$ means x-tuple τ takes no alternative in I_j and $\Pr(\tau) = \sum_{t \in \tau} p(t)$. The entire set of possible instances of an x-Relation, $\mathcal{X}$, denoted as $pwd(\mathcal{X})$, is the set of all the subsets I_j ($\subseteq \mathcal{X}$) with probability greater than 0 ($\Pr(I_j) > 0$).

Example 4: Fig. 1.1(b) shows the total 6 possible worlds for the x-Relation in Fig. 1.1(a). The possible world $\{t_1, t_2\}$ means that, τ_1 takes the alternative t_1, τ_2 takes the alternative t_2, and τ_3 takes none. The probability of this possible world becomes $p(t_1)p(t_2)(1-p(t_4)) = 0.06$. Note that the sum of the probabilities of all the possible worlds is equal to 1. □

Positional Probability: Several probabilistic top-k semantics have been proposed recently under the x-Relational model [86, 94, 45, 53]. One fundamental concept underlying all these ranking semantics is the positional probability, $p_{i,j}$, which is the probability of a tuple, t_i, to be ranked at the j-th position across the entire set of possible worlds [86, 93], i.e.,

$$p_{i,j} = \sum_{I \in pwd(\mathcal{X}), t_i = \Psi_j(I)} \Pr(I) \tag{1.1}$$

where $\Psi_j(I)$ denote the tuple with the j-th largest score in an instance I of the possible worlds. We will show how the different ranking semantics are connected to the positional probability in Chapter. 2. It is important to note that all these probabilistic ranking semantics need to compute the $p_{i,j}$ values for all $t_i \in \mathcal{X}$ and $j = 1, \cdots, k$, and computing $p_{i,j}$ is the dominant cost in such probabilistic ranking queries.

Example 5: Consider the x-Relation shown in Fig. 1.1(a) and the possible worlds shown in Fig. 1.1(b), the probability for t_2 to be ranked at the first place is $p_{2,1} = 0.04 + 0.16 + 0.1 + 0.4 = 0.7$, and the probability for t_3 to be ranked 2nd is $p_{3,2} = 0.1 + 0.4 = 0.5$. The tuple t_1 has the highest score 100 but with a low probability 0.3, which is less than $p_{2,1} = 0.7$, the probability of t_2 to be ranked top across possible worlds. □

1.2. Contributions

Due to the inherent uncertainty in real world data, and that ranking is a fundamental process in analyzing data, we study several aspects of uncertain data ranking. In a nutshell,

1. We present a linear time algorithm to compute the positional probability, and propose a tight upper bound for top-k probability.

2. We study top-k probabilistic ranking queries with joins when scores and probabilities are stored in different relations.

3. We extend the possible worlds semantics to ranking twig query results of a probabilistic XML.

4. We study context sensitive document ranking, where the connection between documents and the underlying multi-attribute graph is uncertain.

We will now briefly describe these points.

In Chapter. 3, we focus on the fast computation of positional probability, $p_{i,j}$, which is defined as the probability of a tuple t_i to be ranked at the j-th position across possible worlds. Among the different ranking semantics studied in the literature, most of them need to compute the positional probability. The cost of computing positional probabilities is the dominant cost. We propose a new novel algorithm that computes such probability efficiently based on conditional probability and the system of linear equations. We prove the correctness of our approach, and show that the time complexity is linear. Based on the formulation of conditional probability, we prove a tight upper bound of the top-k probability of tuples, which is then used to stop the top-k computation earlier. The content of this chapter can also be found in [19].

In Chapter. 4, we study top-k probabilistic ranking queries with joins when s-cores and probabilities are stored in different relations. The existing probabilistic ranking queries have an implicit assumption that both scores based on which object-s are ranked and probabilities of the existence of the objects are stored in the same relation. However, we observe that, in general, scores and probabilities are highly possible to be stored in different relations, e.g. in column-store database, data integration, and data warehouse. We focus on reducing the join cost in probabilistic top-k ranking. We investigate two probabilistic score functions, namely, expected rank value and probability of highest ranking. We give upper/lower bounds of such probabilistic score functions in random access and sequential access, and discuss the advantages/ disadvantages of random and sequential accesses. We propose new I/O efficient algorithms to find top-k objects with probabilistic ranking functions, using random access, sequential access, and the combination of random and sequential access by taking the advantages from both random/sequential access. We have published these results in [22].

In Chapter. 5, we study probabilistic XML rank query, which is to rank top-k probabilities of the answers of a twig query in probabilistic XML data. The new challenge is how to compute top-k probabilities of answers of a twig query in probabilistic XML in the presence of containment (ancestor/descendant) relationships. The existing dynamic programming approaches to compute top-k probabilities over a set of tuples cannot be directly applied, because in the context of probabilistic XML any node/edge may possibly have impacts on the top-k probabilities of answers. In our study, we consider all the three issues, namely, ranking, probability, and structures. We focus on node queries first, and propose a new dynamic programming algorithm which can compute top-k probabilities for the answers of node queries based on the previously computed results in probabilistic XML data. We further propose optimization techniques to share the computational cost. We also show techniques to support path queries and tree queries. The contents of this chapter has also been published in [20].

In Chapter. 6, we study how to rank documents using a set of keywords, given a context that is associated with the documents. The uniqueness of the problem is that the documents to be ranked are associated with sets of interrelated multi-attribute tuples called context, which contains additional information that assists users to rank the relevant documents with some uncertainty. We model the problem using a graph with two different kinds of nodes (document nodes and multi-attribute nodes), where the edges between document nodes and multi-attribute nodes exist with some probability. We discuss its score function, cost function, and ranking with uncertainty. We also propose new algorithms to rank documents that are most related to the user-given keywords by integrating the context information. We have published these results in [18, 21].

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1. Uncertain Top-k Semantics

Due to the two interacting ranking dimensions of uncertain data, i.e., score and probability, several ranking semantics have been proposed in the literature to compute the top-k answers by the interplay of score and probability, based on the possible worlds semantics.

U-Topk and U-kRanks Query: Soliman et al. are the first to study the ranking issues in probabilistic data under the possible world semantics [86], and propose two probabilistic ranking queries: Uncertain Top-k query (U-Topk query) and Uncertain k Ranks Query (U-kRanks query). A U-Topk query returns the k tuples that are most likely to be top-k tuples in possible worlds. A U-kRanks query returns, for each rank position j ranged from 1 to k, the tuple t_i that most likely to be ranked at jth position across possible worlds, i.e., the answer to a U-kRanks query on an x-Relation $\mathcal{X}$ is a vector $(t_1^*, \cdots, t_k^*)$, where $t_j^* = \arg\max_{t_i} p_{i,j}$ for $j = 1, \cdots, k$, (see the definition of positional probability, $p_{i,j}$, in Chapter. 1.1). Consider the x-Relation and its possible worlds shown in Fig. 1.1, the top-2 results based on U-topk query semantics are $\{t_2, t_3\}$ which has probability 0.5 to be the top-2 answer across possible worlds, and the top-2 results based on U-kRanks query semantics are (t_2, t_3) where $p_{2,1} = 0.66$ and $p_{3,2} = 0.5$. Yi et al. [92, 93] improve the performance of the U-Topk and U-kRanks queries using a dynamic programming approach. Soliman et al. also study ranking aggregate queries in probabilistic data based on these two semantics in [87].

Top-k Probability: Hua et al. [44, 45] study a PT-k query, which returns the set of tuples whose top-k probabilities are above a user-specified threshold. Let $tkp(t_i)$ be the top-k probability of a tuple, t_i, which is the marginal probability that t_i is ranked top-k in the possible worlds [44],

$$tkp(t_i) = \sum_{I \in pwd(\mathcal{X}), t_i \in topk(I)} \Pr(I) = \sum_{j=1}^{k} p_{i,j} \tag{2.1}$$

where $t_i \in topk(I)$ means that the tuple t_i is ranked as one of the top-k tuples in the instance I. Three approaches, namely, dynamic programming method, sampling method, and poisson approximation based method, are proposed in [45] to answer such PT-k queries. With similar idea of PT-k query, Zhang and Chomicki [94] return the k tuples with highest top-k probabilities for a top-k query, which is called the Global Top-k query. For an x-Relation $\mathcal{X}$, the answer to a Global Top-k query is a set of size k, $\{t_1^*, \cdots, t_k^*\}$, which satisfies $tkp(t_j^*) \geq tkp(t)$ for any $j = 1, \cdots, k$ and $t \notin \{t_1^*, \cdots, t_k^*\}$. Consider the x-Relation and its possible worlds shown in Fig. 1.1, The top-2 probability for the four tuples are, $tkp(t_1) = 0.3$, $tkp(t_2) = 1.0$, $tkp(t_3) = 0.5$, and $tkp(t_4) = 0.16$. Therefore, the result for a Global top-2 query is $\{t_2, t_3\}$. Jin et al. [53] study the U-Topk/U-kRanks/Global-topk queries in an uncertain stream environment under a sliding-window model, and design both space- and time-efficient synopses to continuously monitor the top-k results.

Expected Rank: Cormode et al. [28] propose the expected rank query, and rank tuples based on their expected rank values. The expected rank value of a tuple t_i, $R_E(t_i)$, is defined as follows,

$$R_E(t_i) = \sum_{I \in pwd(\mathcal{X})} \Pr(I) \cdot rank_I(t_i) \tag{2.2}$$

where $rank_I(t_i)$ is the rank of a tuple o_i in a possible instance I, i.e., the number of tuples in I whose scores are larger than t_i,

$$rank_I(t_i) = \begin{cases} |\{t_j \in I \mid score(t_j) > score(t_i)\}|, & \text{if } t_i \in I; \\ |I|, & \text{otherwise.} \end{cases}$$

Note that the top tuple has rank 0. In essence, the expected rank considers the rank of a tuple as a random variable, and ranks a tuple based on the expected value of the random variable. Consider the x-Relation and its possible worlds shown in Fig. 1.1, the rank of t_1 in the first possible world is 1, and the expected rank values of the four tuples are, $R_E(t_1) = 1.36$, $R_E(t_2) = 0.3$, $R_E(t_3) = 1.46$, and $R_E(t_4) = 1.8$. Li et al. [59] compute the top-k answers in the scenario of uncertain distributed data based on the expected rank semantics, where subsets of the tuples are distributed at different places.

Probability of Highest Rank: The probability for a tuple t_i to be ranked at the first place is defined as follows.

$$P_{HR}(t_i) = \sum_{\substack{I \in pwd(\mathcal{X}) \\ t_i \in I \\ rank_I(t_i)=0}} \Pr(I) \tag{2.3}$$

It ranks tuple, t_i, based on the summation of the probability of the possible instances where t_i appears and is ranked at the first place (rank 0). In [12], Beskales et al. compute the probability of highest rank for each object, where the rank is defined on the distance between it and the query object, and then report the k objects with highest such probabilities. Similarly, in [72, 64], the authors retrieve k objects from an uncertain spatial database, that have the highest probability to be a skyline point.

Others: Li et al. [61] propose two parameterized ranking functions to rank uncertain data. In [85, 36], the authors study ranking tuples whose scores are uncertain. In [76], Re et al. find the k most probable answers for a given general SQL query. In this scenario, each answer has a probability instead of a score, which intuitively represents the confidence of its existence, ranking is only based on probabilities.

2.2. Uncertain Database Models

In recent years, there have been a lot of probabilistic databases designed, which include, Trio system [3, 11], MystiQ system [29, 77], SPROUT system [71], MayBMS system [8, 7], ORION system [23], PrDB system [81], and MCDB system [51].

The Trio system [3] designs a model for incomplete and probabilistic databases based on maybe-tuples, X-tuples, and lineage expressions, searching for a balance between expressiveness and simplicity. The x-Relation model we used in this thesis is similar to the model used in the Trio system [3]. Tuple-independent probabilistic databases are discussed by Dalvi and Suciu [29], motivated by queries with approximate predicates which introduce an independent event for every potential match. However, queries over uncertain databases are hard in general. Even on a totally independent probabilistic database, answering a general SQL query is #P-complete [29], specifically, it is #P-complete to compute the existence probability of a result. The SPROUT system [71] is also a tuple-independent probabilistic database. They consider the conjunctive queries without self-joins that are known to be tractable on any tuple-independent database, and queries that are not tractable in general but become tractable on probabilistic databases restricted by functional dependencies. They study optimizations that push or pull our operator or parts thereof past joins, and propose an efficient secondary-storage operator for exact computation of queries on tuple-independent probabilistic databases

In MayBMS [7], they represent uncertain database by U-relations, which is a succinct and purely relational representation system. U-relations support attribute-level uncertainty using vertical partitioning. The ORION system [23] designs a model for continuously moving objects, which represents the possible locations of object by probability distribution functions. In PrDB [81], they use graphical models, which is a probabilistic modeling technique developed within the statistics and machine learning community, to model uncertain data. PrDB captures more complex models (correlated tuples and attributes) and allows compact representation (shared and schema-level correlations). They show how query evaluation in PrDB translates into inference in an appropriately augmented graphical model. The MCDB system [51] manages uncertain data based on a Monte Carlo approach. It represents uncertainty via "VG functions", which are used to pseudorandomly generate realized values for uncertain attributes. By storing parameters, and not probabilities, and by estimating, rather than exactly computing, the probability distribution over possible query answers, MCDB can easily handle arbitrary joint probability distributions over discrete or continuous attributes, arbitrarily complex SQL queries, and arbitrary functionals of the query-result distribution such as means, variances, and quantiles.

2.3. Join Query in Uncertain Database

Join queries in uncertain database have been studied in [24, 84, 35, 4]. Cheng et al. [24] first study probabilistic threshold join queries. In [24], the join attributes have uncertain values, which are represented as probability distribution functions (pdfs). Two tuples are joined with a probability which is defined as the probability that the two pdfs choose the same value. They propose the concept of x-bounds, where x is a constant probability number between 0 and 1. The idea is to have a series of x-bounds for some data structure (e.g., each data page), for various x values between 0 and 1, and then tries to prune the whole data structure during query processing to save costs. Singh et al. [84] study the same probabilistic threshold join query, but on categorical data. They propose two index structures for efficiently searching uncertain categorical data, one based on the R-tree and another based on an inverted index structure. Using these structures, they find join results efficiently.

Ge [35] study the semantics of uncertain join, and formalize two kinds of join operations on uncertain data, namely v-join and d-join, which are each useful for different applications. They devise efficient query processing algorithms for the two join operations using probability theory. Specifically, for v-join operation, they use probability bounds that are based on the moments of random variables to either early accept or early reject a candidate v-join result tuple. They also devise an indexing mechanism and an algorithm called Two-End Zigzag Join to further save I/O costs. For d-join operation, they first reduce the problem to a special form of similarity join in a multidimensional space, and then design an efficient algorithm called condensed d-join and an optimal condensation scheme based on dynamic programming. Agrawal et al. [4] study the problem of finding join results in an uncertain database. They study how to get the "top-k" results by confidence, or how to obtain results sorted by confidence. They address the problem of processing all the queries when sufficient memory is not available, minimizing retrieval cost. Although they consider the join issues in a top-k query, they treat each probability attribute as an ordinary numeric attribute, and rank the answers based on the aggregated probabilities. Above all, all these join queries do not involve the possible worlds semantics and the top-k ranking semantics discussed previously.

2.4. Probabilistic XML

The topic of probabilistic XML (PXML) has been studied recently. Many models have been proposed, together with the complexity analysis of query evaluations. Nierman et al. [70] first introduce a simple probabilistic XML model, ProTDB, which is a probabilistic tree database with probabilistic types, independent and mutually-exclusive. Hung et al. [46, 47] model the probabilistic XML as directed acyclic graphs, with probabilities defined on sets of children, therefore supporting arbitrary distributions over sets of children. Keulen et al. [90] use a probabilistic tree approach for data integration, where its probability and possibility nodes are similar to independent node and mutually-exclusive node, respectively. Abiteboul et al. [2] propose a "fuzzy trees" model, where nodes are associated with conjunctions of probabilistic event variables. They also give a full complexity analysis of query and update on the "fuzzy tree" in [82].

Cohen et al. [25] incorporate a set of constraints to express more complex dependencies among the probabilistic data. They also propose efficient algorithms to solve the constraint-satisfaction, query evaluation, and sampling problem under a set of constraints. In [55], Kimelfeld et al. summarize and extend the probabilistic XML models previously proposed, the expressiveness and tractability of queries on different models are discussed. They also study the problem of evaluating twig queries over probabilistic XML that may return incomplete or partial answers with respect to a probability threshold to users in [57]. Li et al. [62] study the problem of top-k keyword search over probabilistic XML data, which is to retrieve k SLCA results with the k highest probabilities of existence, and propose two efficient algorithms. All the above work focus on the probabilistic XML models and XML queries. The ranking issues, especially possible worlds semantics based ranking, are not addressed in these works.

2.5. Top-k Queries in Deterministic Data

Top-k queries in deterministic data have been studied extensively. A detailed survey can be found in [50]. In general, it is to find the top-k answers with respect to a user specified score function by joining and aggregating multiple inputs(or relations).

The top-k algorithms by Fagin et al. are the most influential [32, 33]. They consider both random access and/or sequential access of the lists of base scores, where each list of a base score can be viewed as a separate relation. There are many works considering the scenario that random access is not supported by the underlying sources. The No Random Access (NRA) algorithm [33], the Stream-Combine algorithm [39], and the LARA-j algorithm [66] answer a top-k query by sequential accesses on the lists of base scores. The J^* algorithm [69], algorithms in [49], and the family of PBRJ algorithms [80] retrieve the join answers with top-k scores, using sequential access on the base relations. Marian et al. propose Upper and Pick algorithm to answer top-k queries, when sequential access is provided and also controlled random accesses is provided [67]. But, these work consider deterministic data, and can not be directly applied to probabilistic data. In probabilistic ranking, each tuple has both a score and a probability, the tuples are ranked based on the possible worlds semantics.

CHAPTER 3

FAST COMPUTING POSITIONAL PROBABILITY AND TOP-K PROBABILITY

3.1. Introduction

The probabilistic top-k queries based on the interplay of score and probability, under the possible worlds semantic, become an important research issue that considers both score and uncertainty on the same basis. In the literature, many different probabilistic top-k queries are proposed. As discussed in Chapter. 2.1, most of the ranking semantics need to compute the positional probability, $p_{i,j}$, which is the probability of a tuple t_i to be ranked at the j-th position across the entire set of possible worlds. The cost of computing $p_{i,j}$ is the dominant cost and is known as $O(kn^2)$, where n is the size of dataset. In this chapter, we propose a new novel algorithm that computes such probability efficiently based on conditional probability and the system of linear equations. We prove the correctness of our approach, and show that the time complexity is $O(kn)$. Based on the formulation of conditional probability, we prove a tight upper bound of the top-k probability of tuples, which is then used to stop the top-k computation earlier. We confirm the efficiency by comparing our approach with the up-to-date heuristics and find that our approach can be at least $3,000$ times faster.

The remainder of this chapter is organized as follows. In Sec. 3.2, we review the existing solution to compute the positional probability. We propose a new novel algorithm and prove its correctness in Sec. 3.3. In Sec. 3.4, we propose an algorithm to compute the top-k results based on the top-k probability. Finally, performance studies are reported in Sec. 3.5.

3.2. Existing Solution

We discuss $p_{i,j}$ computing for a given k and an x-Relation $\mathcal{X} = \{t_1, \cdots, t_n\}$ sorted in the descending score order. For simplicity and without loss of generality, in the following discussions, we further assume there are no tie scores in $\mathcal{X}$ such that $score(t_i) \neq score(t_j)$ for any $t_i \neq t_j$ in $\mathcal{X}$. Note that all algorithms including our algorithm to be discussed can deal with tie scores with minor modification for computing $p_{i,j}$.

Yi et al. [93] showed that the time complexity of computing $p_{i,j}$ for all $t_i \in \mathcal{X}$ and $j = 1, \cdots, k$ is $O(kn^2)$. We introduce it in brief below.

Given an x-Relation $\mathcal{X} = \{t_1, \cdots, t_n\}$ sorted in the decreasing score order. Let $\mathcal{X}_i = \{t_1, \cdots, t_i\}$ denote a reduced x-Relation on the largest i tuples (based on score), together with the projected (exclusive/independent) relationship between tuples. It is obvious that $p_{i,j}$ is the same to be computed either on $\mathcal{X}$ or $\mathcal{X}_i$, under the x-Relation model. Formally, let $\Pr(\tau|\mathcal{X}_i)$ be the existence probability of an x-tuple τ with respect to $\mathcal{X}_i$ as follows.

$$\Pr(\tau|\mathcal{X}_i) = \sum_{t \in \tau, t \in \mathcal{X}_i} p(t) \tag{3.1}$$

Then, $\Pr(\tau) = \Pr(\tau|\mathcal{X})$.

We highlight the main idea of computing $p_{i,j}$ in $O(kn^2)$ [93] below. First, consider a special case, where every x-tuple contains only one tuple (single-alternative), or equivalently, all the tuples are independent. Then, $p_{i,j}$ is equal to the probability that a randomly generated possible world from $\mathcal{X}_i$ contains t_i and there are j tuples in total. In other words, $p_{i,j}$ is the sum of the probabilities of the possible worlds that contain t_i and there are exactly $j-1$ tuples taken from the set $\mathcal{X}_{i-1} = \{t_1, \cdots, t_{i-1}\}$.

τ_1	$\{t_1(0.3), t_4(0.4)\}$
τ_2	$\{t_2(0.5), t_8(0.2)\}$
τ_3	$\{t_3(0.5), t_6(0.5)\}$
τ_4	$\{t_5(0.6), t_7(0.3)\}$

Table 3.1: Multi-alternative x-Relation

Let $r_{i,j}$ denote the probability that a randomly generated possible world from $\mathcal{X}_i$ has exactly j tuples, then $p_{i,j} = p(t_i) \cdot r_{i-1,j-1}$. For the totally independent case, the set of all $r_{i,j}$ values can be computed efficiently by the following dynamic programming equation, in time complexity $O(kn)$.

$$r_{i,j} = \begin{cases} p(t_i) \cdot r_{i-1,j-1} + (1 - p(t_i)) \cdot r_{i-1,j}, & \text{if } i \geq j > 0; \\ (1 - p(t_i)) \cdot r_{i-1,j}, & \text{if } i > j = 0; \\ 1, & \text{if } i = j = 0; \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

Second, consider the case where some x-tuples may contain multiple tuples (multi-alternative). The noticeable difference is that $p_{i,j} \neq p(t_i) \cdot r_{i-1,j-1}$ in the multi-alternative case, because an x-tuple contains multiple-alternatives that are mutually exclusive. When it needs to compute $p_{i,j}$ for a tuple t_i, the x-tuple that contains t_i may have other alternatives been computed already. It needs to remember whether an alternative of an x-tuple has already been computed in $\mathcal{X}_{i-1}$ using a set denoted $\mathcal{S}$. Let $\mathcal{S} = \{\tau_1, \cdots, \tau_s\}$ be the set of x-tuples, that have at least one alternative computed in $\mathcal{X}_{i-1}$ already, with probability $\Pr(\tau_l|\mathcal{X}_{i-1})$ for $1 \leq l \leq s$ (Refer to Eq. (3.1)). When t_i appears and the x-tuple τ_x that contains t_i has already appeared in $\mathcal{S}$, it computes $p_{i,j}$ as $p_{i,j} = p(t_i) \cdot r'_{s,j-1}$. Here, $r'_{i,j-1}$, for $1 \leq i \leq s$ and $1 \leq j \leq k$, need to be recomputed based on $\mathcal{S} = \{\tau_1, \cdots, \tau_s\}$ with $\Pr(\tau_x|\mathcal{X}_{i-1}) = 0$ using Eq. (3.2), and takes $O(s \cdot k)$ time. In the worst case, it takes $O(i \cdot k)$ to compute $p_{i,j}$ for a specific i. The time complexity to compute $p_{i,j}$ values, for $1 \leq i \leq n$ and $1 \leq j \leq k$, is $O(kn^2)$.

Example 1: Consider an x-Relation, $\mathcal{X}$, in Table 3.1 with four x-tuples, $\{\tau_1, \tau_2, \tau_3, \tau_4\}$

and 8 tuples $\{t_1, \cdots, t_8\}$. Each x-tuple contains two tuples (alternatives). We assume $score(t_i) > score(t_j)$ if $i < j$, and give the probability of each tuple t_i, $p(t_i)$, in the corresponding parentheses. For example, τ_1 has two tuples t_1 and t_4 where $p(t_1) = 0.3$ and $p(t_4) = 0.4$. Let $k = 2$. We show how to compute $p_{i,j}$ for all tuples t_i, for $1 \leq i \leq 8$ and $j = 1, 2$.

Let all 8 tuples in $\mathcal{X}$ be sorted in the decreasing score order, and let $\mathcal{S}$ be the set of x-tuples that have multi-alternatives in $\mathcal{X}_{i-1}$. Initially, $\mathcal{X}_0 = \emptyset$, $\mathcal{S} = \emptyset$.

First, consider t_1 which is the tuple that has the largest score, and $\mathcal{S} = \emptyset$ implies that t_1 has no preceding alternatives. Because $r'_{0,0} = 1$ and $r'_{0,1} = 0$, thus $p_{1,1} = p(t_1) \cdot r'_{0,0} = 0.3$ and $p_{1,2} = p(t_1) \cdot r'_{0,1} = 0$. $\mathcal{X}_1 = \{t_1\}$. Based on Eq. (3.1), the current existence probability of τ_1 in $\mathcal{X}_1$ is $\Pr(\tau_1|\mathcal{X}_1) = p(t_1) = 0.3$. $\mathcal{S}$ is updated to be $\mathcal{S} = \{\tau_1\}$, because the x-tuple τ_1 contains t_1 that has been computed. For simplicity, we use $\mathcal{S} = \{\tau_1(0.3)\}$ to indicate that $\mathcal{S}$ contains τ_1 whose current existence probability is 0.3.

Second, consider the second largest score tuple t_2, which has no preceding alternatives computed, because the x-tuple τ_2 that contains t_2 does not appear in $\mathcal{S} = \{\tau_1(0.3)\}$. Because $r'_{1,0} = 0.7$ and $r'_{1,1} = 0.3$, thus $p_{2,1} = p(t_2) \cdot r'_{1,0} = 0.35$ and $p_{2,2} = 0.15$. $\mathcal{X}_2 = \{t_1, t_2\}$. Based on Eq. (3.1), the current existence probability of τ_2 in $\mathcal{X}_2$ is $\Pr(\tau_2|\mathcal{X}_2) = p(t_2) = 0.5$. $\mathcal{S} = \{\tau_1(0.3), \tau_2(0.5)\}$.

In a similar fashion, the third largest score tuple t_3 is computed which has no preceding alternatives in $\mathcal{S}$. Because $r'_{2,0} = 0.35$ and $r'_{2,1} = 0.5$, thus $p_{3,1} = 0.5 \cdot 0.35 = 0.175$ and $p_{3,2} = 0.25$. $\mathcal{X}_3 = \{t_1, t_2, t_3\}$. Based on Eq. (3.1), the current existence probability of τ_3 in $\mathcal{X}_3$ is $\Pr(\tau_3|\mathcal{X}_3) = p(t_3) = 0.5$. $\mathcal{S} = \{\tau_1(0.3), \tau_2(0.5), \tau_3(0.5)\}$.

Fourth, consider the fourth largest score tuple t_4. Note that the current $\mathcal{S} = \{\tau_1(0.3), \tau_2(0.5), \tau_3(0.5)\}$. But because tuple t_4 has a preceding alternative t_1 in x-tuple τ_1 which appears in $\mathcal{S}$ already, the existence probability of $\Pr(\tau_1|\mathcal{X}_3) = 0$ is reset. Therefore, $\mathcal{S}$ is updated to be $\mathcal{S} = \{\tau_1(0), \tau_2(0.5), \tau_3(0.5)\}$. In order to compute $r'_{3,0}$ and $r'_{3,1}$, all the $r'_{i,j}$ values, for $i = 1, 2$ and $j = 0, 1$, need to be recomputed as well based on the updated $\mathcal{S}$. Because $r'_{1,0} = 1$, $r'_{1,1} = 0$, $r'_{2,0} = 0.5$, $r'_{2,1} = 0.5$, $r'_{3,0} = 0.25$, and $r'_{3,1} = 0.5$, thus $p_{4,1} = p(t_4) \cdot r'_{3,0} = 0.1$ and $p_{4,2} = 0.2$. $\mathcal{X}_4 = \{t_1, t_2, t_3, t_4\}$. Based on Eq. (3.1), the current existence probability of τ_1 in $\mathcal{X}_4$ is $\Pr(\tau_1 | \mathcal{X}_4) = p(t_1) + p(t_4) = 0.3 + 0.4 = 0.7$. Therefore, $\mathcal{S} = \{\tau_1(0.7), \tau_2(0.5), \tau_3(0.5)\}$, which will be used in the next iteration.

The same procedure repeats until all $p_{i,j}$ for all $t_i \in \mathcal{X}$ and $j = 1, 2$ are computed. □

Note that, between consecutive computations of $p_{i,j}$ and $p_{i+1,j}$, some $r'_{s,j}$ computing cost can be shared [45, 93]. Hua et al. [45] also studied several heuristics to fast compute $p_{i,j}$ but in the worst case it is $O(kn^2)$.

3.3. A New Novel Algorithm

In this section, we propose a novel $O(kn)$ algorithm using a newly introduced conditional probability $c_{i,j}$ given below,

$$c_{i,j} = \Pr(\text{Exactly } j \text{ tuples appear in } \{t_1, \cdots, t_i\} \mid t_{i+1} \text{ appears}) \tag{3.3}$$

to fast compute $p_{i,j}$. Consider a general multi-alternative case. Let $\mathcal{X}_i = \{t_1, \cdots, t_i\}$ be the set computed already. Now, we consider t_{i+1}, assume t_{i+1} appears. Among the tuples computed already in $\mathcal{X}_i$, there may exist several tuples in $\mathcal{X}_i$ that are contained in the same x-tuple that contains t_{i+1}. Those tuples need to be removed in order to compute for t_{i+1}, as we discussed in the previous section by setting the existence probability to be zero. Eq. (3.3) is the conditional probability of having exactly j tuples in $\mathcal{X}_i = \{t_1, \cdots, t_i\}$ after removing those tuples in $\mathcal{X}_i$ that are contained in the same x-tuple that contains t_{i+1}, given t_{i+1} appears. It is interesting to note that

$$\begin{aligned} p_{i,j} &= \Pr(t_i \text{ appears}) \cdot \Pr(\text{Exactly } j\text{-1 tuples appear in } \{t_1, \cdots, t_{i-1}\} \mid t_i \text{ appears}) \\ &= p(t_i) \cdot c_{i-1,j-1} \end{aligned} \tag{3.4}$$

And the problem becomes how to compute $c_{i,j}$ efficiently. Note that there is no obvious relationship between $c_{i,j}$ and $c_{i-1,j}$ (refer to Eq. (3.2)). However, we observe that there is a similar relationship between $c_{i,j}$ and $r_{i,j}$. Let τ_x be the x-tuple that contains t_{i+1}. Then, the relationship between $c_{i,j}$ and $r_{i,j}$ becomes as follows,

$$r_{i,j} = \begin{cases} (1 - \Pr(\tau_x|\mathcal{X}_i)) \cdot c_{i,j}, & \text{if } j = 0; \\ (1 - \Pr(\tau_x|\mathcal{X}_i)) \cdot c_{i,j} + \Pr(\tau_x|\mathcal{X}_i) \cdot c_{i,j-1}, & \text{if } j > 0; \end{cases} \tag{3.5}$$

Lemma 3.3.1: *Eq. (3.5) correctly computes $r_{i,j}$, given $c_{i,j}$.* □

Proof Sketch: Assume that $c_{i,j}$ for $0 \le j \le k-1$ are correct as defined, the probability that a randomly generated possible world has exactly j tuples from $\mathcal{X}_i$ is conditioned by the appearance of t_{i+1}. Let τ_x be the x-tuple that has t_{i+1}, and ρ denote $\Pr(\tau_x|\mathcal{X}_i)$. There are two cases.

First, t_{i+1} has no preceding alternative, equivalently $\rho = 0$. Then the two parts in the conditional probability $c_{i,j}$ are independent, $c_{i,j} = \Pr(\text{Exactly } j \text{ tuples appear in } \{t_1, \cdots, t_i\})$, where the latter part of the equation is actually $r_{i,j}$. Hence, Eq. (3.5) correctly computes $r_{i,j}$, given that $c_{i,j}$ are correct.

Second, t_{i+1} has some preceding alternatives, equivalently $\rho > 0$. Assume that $\mathcal{S} = \{\tau_1, \cdots, \tau_s, \tau_x\}$ is the set of x-tuples that have alternatives appearing in $\mathcal{X}_i = \{t_1, \cdots, t_i\}$, where $\Pr(\tau_l|\mathcal{X}_i) > 0$ for all $\tau_l \in \mathcal{S}$. Then $c_{i,j}$ is the probability that a randomly generated possible world from $\{\tau_1, \cdots, \tau_s\}$ ($= \mathcal{S} \setminus \{\tau_x\}$) has exactly j x-tuples, and $r_{i,j}$ is the probability that a randomly generated possible world from $\{\tau_1, \cdots, \tau_s, \tau_x\}$ has exactly j x-tuples. Hence, Eq. (3.5) is correct based on the same idea shown in Eq. (3.2). □

Given $c_{i,j}$ we can compute $r_{i,j}$ using Eq. (3.5). The reverse also holds such that, given $r_{i,j}$, we can compute $c_{i,j}$ correctly by the system of linear equations defined in Eq. (3.5). A general system of linear equations with n equations and n variables needs time $O(n^3)$. But the system of linear equations defined by Eq. (3.5) is in a special form, there are only two diagonals of the coefficient matrix which are non-zero, so it can be solved in $O(n)$ time [58]. In our problem, there are k linear equations with k variables, it can be solved in time $O(k)$, using $c_{i,0} = r_{i,0}/(1-\rho)$ and $c_{i,j} = (r_{i,j} - \rho \cdot c_{i,j-1})/(1-\rho)$ where $\rho = \Pr(\tau_x|\mathcal{X}_i)$, for $1 \le j \le k-1$. Note that

$0 < \Pr(\tau_x|\mathcal{X}_i) < 1$. In addition, given $c_{i,j}$, $r_{i+1,j}$ can also be computed using Eq. (3.5), by replacing $\Pr(\tau_x|\mathcal{X}_i)$ with $\Pr(\tau_x|\mathcal{X}_{i+1})$, where τ_x is the x-tuple that contains t_{i+1}.

Algorithm 1 CondProb($\mathcal{S}$, $\mathcal{R}_{i-1}$,t_i)

```
Input:  the probability for $x$-tuples $\mathcal{S} = \{\tau_1(\Pr(\tau_1)), \cdots, \tau_s(\Pr(\tau_s))\}$
        $\mathcal{R}_{i-1} = \{r_{i-1,0}, \cdots, r_{i-1,k-1}\}$ and a tuple $t_i$.
Output: $r_{i,j-1}$ and $p_{i,j}$, for $1 \le j \le k$.

 1: Let $\tau_x$ be the $x$-tuple that has $t_i$;
 2: $\rho \leftarrow \Pr(\tau_x)$ if $\tau_x(\Pr(\tau_x))$ appears in $\mathcal{S}$ otherwise 0;
    // compute $c_{i-1,j}$ and $p_{i,j}$ for $0 \le j \le k-1$
 3: $c_{i-1,0} \leftarrow r_{i-1,0}/(1-\rho)$;
 4: for $j \leftarrow 1$ to $k-1$ do
 5:     $c_{i-1,j} \leftarrow (r_{i-1,j} - \rho \cdot c_{i-1,j-1})/(1-\rho)$;
 6: end for
 7: $p_{i,j} \leftarrow p(t_i) \cdot c_{i-1,j-1}$, for $1 \le j \le k$;
    // compute $r_{i,j}$ for $0 \le j \le k-1$
 8: $\rho \leftarrow \rho + p(t_i)$;
 9: $r_{i,0} \leftarrow (1-\rho) \cdot c_{i-1,0}$;
10: for $j \leftarrow 1$ to $k-1$ do
11:     $r_{i,j} \leftarrow (1-\rho) \cdot c_{i-1,j} + \rho \cdot c_{i-1,j-1}$;
12: end for
13: if $\tau_x \notin \mathcal{S}$ then
14:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{\tau_x(\rho)\}$;
15: else
16:     update $\mathcal{S}$ by changing $\Pr(\tau_x)$ to be $\rho$;
17: end if
18: return ($\mathcal{S}$, $\{r_{i,0}, \cdots, r_{i,k-1}\}$, $\{p_{i,1}, \cdots, p_{i,k}\}$);
```

The algorithm to compute $r_{i,j}$ and $p_{i,j}$ values for a tuple t_i is shown in Algorithm 1. It takes three inputs, namely, the tuple t_i, the $r_{i-1,j}$ values, and a set of x-tuples, $\mathcal{S} = \{\tau_1, \cdots, \tau_s\}$, that have been computed with their probability

$\Pr(\tau_l) = \Pr(\tau_l|\mathcal{X}_{i-1})$. It first computes ρ (line 1-2). Then, it computes the $c_{i-1,j}$ values by solving a system of linear equations defined by Eq. (3.5) (line 3-5), and computes the $p_{i,j}$ values (line 6). In line 7-10, it computes the $r_{i,j}$ values using Eq. (3.5). Finally, it updates the probability $\Pr(\tau_x)$ (line 11-14). Note that, in our algorithm, the only values needed to compute $p_{i,j}$ values are $r_{i-1,j}$ values and $\Pr(\tau_x|\mathcal{X}_{i-1})$.

Theorem 3.3.1: *Algorithm 1 correctly computes the $p_{i,j}$ values with time complexity of $O(k)$.* □

Proof Sketch: It is obvious from the discussions above. □

In order to compute all $p_{i,j}$, we enumerate all $t_i \in \mathcal{X}$, which is sorted in the descending score order, such as $score(t_i) > score(t_j)$ if $i < j$ as given below.

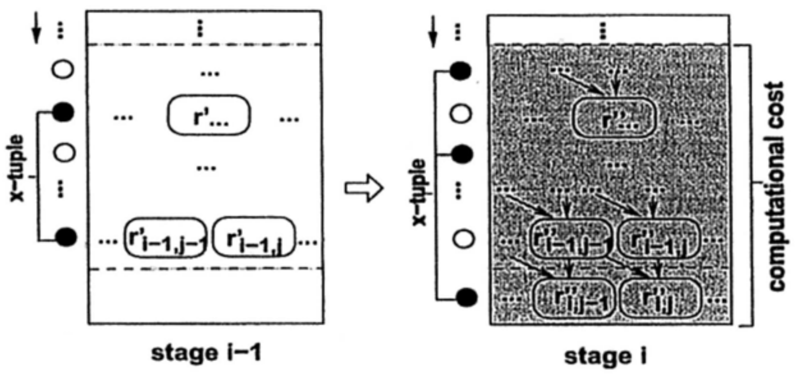
Algorithm 2 PositionalProbabilities()

```
1: Let $\mathcal{S} = \emptyset$;
2: Let $\mathcal{R}_0 = \{r_{0,0}, r_{0,1}, \cdots, r_{0,k-1}\}$ where $r_{0,j}$, for $0 \le j \le k-1$, are computed;
3: for $i = 1$ to $n$ do
4:    $(\mathcal{S}, \mathcal{R}_i, \mathcal{P}_i) \leftarrow$ CondProb($\mathcal{S}$, $\mathcal{R}_{i-1}$, $t_i$);
5:    output $\mathcal{P}_i = \{p_{i,1}, p_{i,2}, \cdots, p_{i,k}\}$;
6: end for
```

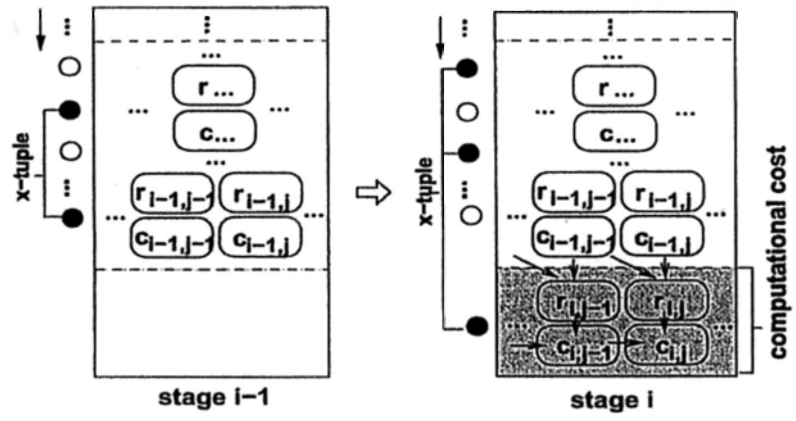
It is obvious that the time complexity to compute all $p_{i,j}$ is $O(kn)$.

Fig. 3.1(a) illustrates the existing $O(kn^2)$ approach to compute $r'_{i,j}$ in the stage i based on the stage i-1. Note that the stage i is the i-iteration to compute for the i-th largest score tuple in $\mathcal{X}_i$. On the left side in the stage i-1 and the stage i, it indicates that some x-tuple (marked by •) contains several tuples (alternatives). On the other hand, Fig. 3.1(b) illustrates our $O(kn)$ approach to compute $r_{i,j}$, using $c_{i,j}$, in the stage i based on the stage i-1. The shaded parts in Fig. 3.1(a)(b) indicate the equations needed to compute, and the difference between the two shaded regions confirms the significant cost saving of our approach.

Example 2: Consider the example x-Relation in Table 3.1. We show the steps of our algorithm to compute $p_{i,j}$. Let $k = 2$. We denote the sequence of x-tuples that have



(a) The Existing $O(kn^2)$ Approach



(b) Our New $O(kn)$ Approach

Figure 3.1: Computational Cost

been scanned as $\mathcal{S}$. Initially, $\mathcal{X}_0 = \emptyset$, $\mathcal{S} = \emptyset$, $r_{0,0} = 1$ and $r_{0,1} = 0$.

First, consider t_1 which is the largest score tuple. It has no preceding alternatives, $\Pr(\tau_1) = 0$, $c_{0,0} = 1$ and $c_{0,1} = 0$. Then, $p_{1,1} = p(t_1) \cdot c_{0,0} = 0.3$ and $p_{1,2} = 0$. After computing t_1, $\mathcal{X}_1 = \{t_1\}$, $\mathcal{S} = \{\tau_1(0.3)\}$, and we have $r_{1,0} = (1 - \Pr(\tau_1)) \cdot c_{0,0} = 0.7$ and $r_{1,1} = (1 - \Pr(\tau_1)) \cdot c_{0,1} + \Pr(\tau_1) \cdot c_{0,0} = 0.3$.

The second largest score tuple t_2 has no preceding alternatives, $\Pr(\tau_2) = 0$, $c_{1,0} = r_{1,0} = 0.7$ and $c_{1,1} = 0.3$. Then, $p_{2,1} = p(t_2) \cdot c_{1,0} = 0.35$ and $p_{2,2} = 0.15$. After computing t_2, $\mathcal{X}_2 = \{t_1, t_2\}$, $\mathcal{S} = \{\tau_1(0.3), \tau_2(0.5)\}$, and in addition we have $r_{2,0} = 0.35$ and $r_{2,1} = 0.5$.

The third largest score tuple t_3 has no preceding alternatives $\Pr(\tau_3) = 0$, $c_{2,0} = 0.35$ and $c_{2,1} = 0.5$. Then, $p_{3,1} = p(t_3) \cdot c_{2,0} = 0.175$ and $p_{3,2} = 0.25$. After computing t_3, $\mathcal{X}_3 = \{t_1, t_2, t_3\}$, $\mathcal{S} = \{\tau_1(0.3), \tau_2(0.5), \tau_3(0.5)\}$, and in addition we have $r_{3,0} = 0.175$ and $r_{3,1} = 0.425$.

The fourth largest score tuple t_4 has a preceding alternative t_1 that is contained in x-tuple τ_1 which appears in $\mathcal{S}$. Therefore, $\rho = \Pr(\tau_1|\mathcal{X}_3) = 0.3$, $c_{3,0} = r_{3,0}/(1-\rho) = 0.25$, $c_{3,1} = (r_{3,1} - \rho \cdot c_{3,0})/(1-\rho) = 0.5$, $p_{4,1} = p(t_4) \cdot c_{3,0} = 0.1$ and $p_{4,2} = 0.2$. After computing t_4, $\mathcal{X}_4 = \{t_1, t_2, t_3, t_4\}$, $\mathcal{S} = \{\tau_1(0.7), \tau_2(0.5), \tau_3(0.5)\}$, and in addition we have $r_{4,0} = 0.075$ and $r_{4,1} = 0.325$.

The same procedure repeats until all $p_{i,j}$ for all $t_i \in \mathcal{X}$ and $j = 1, 2$ are computed.

□

3.4. Top-k Computation

Algorithm 1 returns the set of $p_{i,j}$, which can be used to compute the top-k probability of a tuple, e.g. $tkp(t_i) = \sum_{j=1}^{k} p_{i,j}$. A naive way to get the top-k result is to first compute the top-k probabilities for all tuples, then report the top-k tuples with respect to their top-k probabilities. In the following, we will first discuss an upper bound, and then propose an early stop condition, which avoids to retrieve all the tuples.

Lemma 3.4.1: *Let $\{t_1, \cdots, t_i, \cdots\}$ be the order we scan the tuples, or equivalently it is the decreasing score order, and $r_{i,j}$ is defined as above. Then $tkp(t_{i+1}) \leq \sum_{j=1}^{k} r_{i,j}$, for all $i \geq 1$. This upper bound is also tight for an arbitrary sequence of tuples.* □

Proof Sketch: Let τ_x be the x-tuple that have t_{i+1}, and $p = \Pr(\tau_x|\mathcal{X}_i)$. Note that p may be zero, or equivalently t_{i+1} has no preceding alternative. By Eq. (3.5), sum up the $r_{i,j}$'s, $\sum_{j=0}^{k-1} r_{i,j} = \sum_{j=0}^{k-1} c_{i,j} - p \cdot c_{i,k-1}$. We have

$$\begin{aligned}
tkp(t_{i+1}) &= \textstyle\sum_{j=1}^{k} p_{i+1,j} \\
&= p(t_{i+1}) \cdot \textstyle\sum_{j=0}^{k-1} c_{i,j} \\
&\leq (1-p) \cdot \textstyle\sum_{j=0}^{k-1} c_{i,j} \\
&= \textstyle\sum_{j=0}^{k-1} c_{i,j} - p \cdot \sum_{j=0}^{k-1} c_{i,j} \\
&\leq \textstyle\sum_{j=0}^{k-1} c_{i,j} - p \cdot c_{i,k-1} = \sum_{j=0}^{k-1} r_{i,j}
\end{aligned}$$

where the third inequality holds because $\Pr(\tau_x|\mathcal{X}_i) + p(t_{i+1}) \leq 1$, as t_{i+1} is an alternative of x-tuple τ_x. So $tkp(t_{i+1}) \leq \sum_{j=1}^{k} r_{i,j}$. When $p(t_{i+1}) = 1$, $\Pr(\tau_x|\mathcal{X}_i) = 0$, the above inequalities hold with equality, and therefore $tkp(t_{i+1}) = \sum_{j=1}^{k} r_{i,j}$. Hence this upper bound is tight. □

Lemma 3.4.2: *$\sum_{j=0}^{k-1} r_{i,j}$ are in decreasing order, i.e., $\sum_{j=0}^{k-1} r_{i,j} \geq \sum_{j=0}^{k-1} r_{i+1,j}$, for any $i \geq 1$.* □

Proof Sketch: There are two cases, t_{i+1} has preceding alternatives or not.

First, if t_{i+1} does not have preceding alternatives, then $r_{i+1,j}$ can be computed by Eq. (3.2). Summing up $r_{i+1,j}$, we have $\sum_{j=0}^{k-1} r_{i+1,j} = \sum_{j=0}^{k-1} r_{i,j} - p(t_{i+1})r_{i,k-1} \leq \sum_{j=0}^{k-1} r_{i,j}$. Second, if t_{i+1} has preceding alternatives, assuming t_{i+1} is in the x-tuple τ_x, then $\Pr(\tau_x|\mathcal{X}_i) > 0$. Assume that $\{\tau_1, \cdots, \tau_x, \cdots, \tau_s\}$ is the set of x-tuples that have alternatives in $\{t_1, \cdots, t_i\}$, with probability $\Pr(\tau|\mathcal{X}_i)$. Then $\{\tau_1, \cdots, \tau_x, \cdots, \tau_s\}$ is also the set of x-tuples that have alternatives in $\{t_1, \cdots, t_i, t_{i+1}\}$, and their probability is $\Pr(\tau|\mathcal{X}_{i+1})$, with $\Pr(\tau|\mathcal{X}_{i+1}) = \Pr(\tau|\mathcal{X}_i)$ for all x-tuple τ except τ_x, which has $\Pr(\tau_x|\mathcal{X}_{i+1}) > \Pr(\tau_x|\mathcal{X}_i)$. Let $r_{*,j}$ be the probability that a random generated possible world from $\{\tau_1, \cdots, \tau_x, \cdots, \tau_s\}/\tau_x$, with probabilities $\Pr(\tau|\mathcal{X}_i)$, has exactly j x-tuples. The relationship between $r_{i,j}$ and $r_{*,j}$, or between $r_{i+1,j}$ and $r_{*,j}$, is the same as Eq. (3.2) or Eq. (3.5). Then $\sum_{j=0}^{k-1} r_{i,j} = \sum_{j=0}^{k-1} r_{*,j} - \Pr(\tau_x|\mathcal{X}_i) \cdot r_{*,k-1}$, and $\sum_{j=0}^{k-1} r_{i+1,j} = \sum_{j=0}^{k-1} r_{*,j} - \Pr(\tau_x|\mathcal{X}_{i+1}) \cdot r_{*,k-1}$. So $\sum_{j=0}^{k-1} r_{i+1,j} \leq \sum_{j=0}^{k-1} r_{i,j}$, as $\Pr(\tau_x|\mathcal{X}_{i+1}) > \Pr(\tau_x|\mathcal{X}_i)$. □

Theorem 3.4.2: *If all the top-k probabilities of the current top-k result, e.g. from the set $\{t_1, \cdots, t_i\}$, are greater than or equal to $\sum_{j=0}^{k-1} r_{i,j}$, then we can stop, and guarantee that any potential results in $\{t_{i+1}, \cdots, t_n\}$ can not be in the top-k result.* □

With Theorem 3.4.2, we can develop an algorithm to compute the top-k tuples with respect to their top-k probabilities, which is shown in Algorithm 3. It initializes in line 1-5, and *upBound* denotes the upper bound of the top-k probabilities of the remaining tuples (line 5). While the stop condition is not satisfied (line 6), it retrieves

Algorithm 3 Top-k (k)

Input: an integer k, specify the top-k value,

Output: top-k tuples.

```
1: Let $\{\tau_1, \cdots, \tau_m\}$ be the set of all the $x$-tuples;
2: Initialize $\Pr(\tau_i) \leftarrow 0$, for $1 \leq i \leq m$;
3: $top\text{-}k \leftarrow \emptyset$;
4: Initialize $r_0 = 1$ and $r_j = 0$ for $1 \leq j \leq k-1$;
5: $upBound \leftarrow \sum_{j=0}^{k-1} r_j$;
6: while $top\text{-}k[k].tkp < upBound$ do
7:     $t \leftarrow Next()$;
8:     $r_j, p_j \leftarrow CondProb(\{\Pr(\tau_1), \cdots, \Pr(\tau_m)\}, \{r_0, \cdots, r_{k-1}\}, t)$;
9:     $tkp(t) \leftarrow \sum_{j=1}^{k} p_j$;
10:    Insert $t$ into $top\text{-}k$;
11:    $upBound \leftarrow \sum_{j=0}^{k-1} r_j$;
12: end while
13: return $top\text{-}k$;
```

the next largest score tuple (line 7), computes its top-k probability, inserts it into the top-k set (line 8-10), and update the upper bound (line 11). The top-k set is maintained as a min-heap with size of k, $top\text{-}k[k].tkp$ (line 6) is the minimum top-k probability in the min-heap. When inserting a new tuple associate with its top-k probability into min-heap, if its top-k probability is smaller than that at the top of the min-heap, we do not need to insert it. Otherwise, we replace the top tuple of the min-heap with the new tuple and update the heap structure.

Theorem 3.4.3: *Algorithm 3 correctly returns the top-k tuples with highest top-k probabilities. The top-k generator takes time $O(l \cdot (k + log(k))$, where l is scan depth, or equivalently the number of calls $Next()$.* □

Proof Sketch: The correctness directly follows from the above discussions.

The time complexity of $O(l \cdot (k + log(k))$ does not take $Next()$ into consideration.

The initial of line 1-5 takes constant time. Each call of CondProb() (Algorithm 1) takes $O(k)$ time, based on Theorem 3.3.1. Line 9, 11 take time $O(k)$. Line 10 takes time $O(log(k))$, due to the min-heap of size k. Line 6-11 are only executed l times, so the total time complexity is $O(l \cdot (k + log(k)))$. □

3.5. Performance Studies

We have implemented our algorithm in Visual C++. We compare our CondProb algorithm, denoted CP, for computing $p_{i,j}$, with the heuristics proposed in [45] which are RC (rule-tuple compression only), RC+AR (RC with aggressive reordering), and R-C+LR (RC with lazy reordering). The heuristics proposed can improve the efficiency but they are algorithms in $O(kn^2)$, where n is the number of tuples and k is the top-k value. The executable code and data generator used in [45] are downloadable[1]. We use exactly the same synthetic dataset as used in [45], which is also included in the package.

Parameter	**Range**	**Default**
mem-p	0.1, 0.3, 0.5, 0.7, 0.9	0.5
p	0.1, 0.3, 0.5, 0.7, 0.9	0.3
k	200, 400, 600, 800, 1000	200
$\|rule\|$	5, 10, 15, 20, 25	10
#tuple	20000, 40000, 60000, 80000, 100000	20000
#rule	500, 1000, 1500, 2000, 2500	2000

Table 3.2: Parameters and Default Values

The parameters and default values are shown in Tab. 3.2. Here, *mem-p* is the expectation of the membership probability of tuples, p is the threshold specifying the minimum top-k probability of the result tuples returned, k is the top-k value, $|rule|$ is the average number of tuples in a rule (x-tuple), *#tuple* is the total number of tuples,

[1] `http://www.cs.sfu.ca/~jpei/Software/PTKLib.rar`

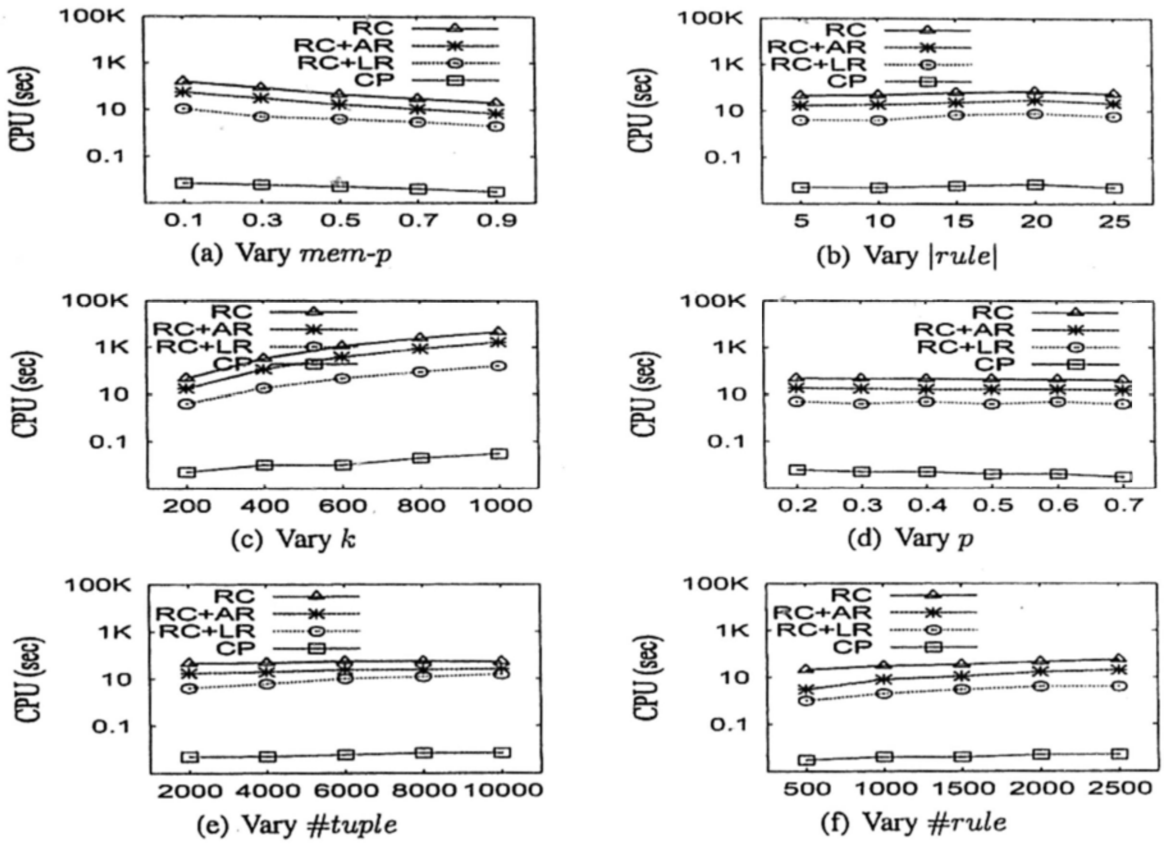


Figure 3.2: Computing $p_{i,j}$

and $\#rule$ is the total number of rules (x-tuples).

The experimental results are shown in Fig. 3.2. In all figures, the shape of the curves for all the four algorithms are all similar, our CP algorithm is $3,000$ times faster than RC+LR on average, and $30,000$ times faster than RC on average.

CHAPTER 4

PROBABILISTIC RANKING OVER RELATIONS

4.1. Introduction

Probabilistic top-k ranking queries have been extensively studied due to the fact that data obtained can be uncertain in many real applications as discussed in Chapter. 2.1. A probabilistic top-k ranking query ranks objects by the interplay of score and probability, with an implicit assumption that both scores based on which objects are ranked and probabilities of the existence of the objects are stored in the same relation. However, we observe that, in general, scores and probabilities are highly possible to be stored in different relations, e.g. in column-store database [1, 48, 7], data integration [76], and data warehouse [16].

In a column-store database, unlike the tuple-based approach taken in conventional relational DBMSs, information is stored in column relations. For example, one column relation stores object identifier and object scores, and another column relation stores object identifier and object existence probability. It is reported that column-oriented DBMSs can perform much better than conventional relational DBMSs in many real applications such as business intelligence applications [1]. Column storage has been successfully used for many years in OLAP (Online Analytical Processing) [88], and is also adapted to perform OLTP (Online Transactional Processing) recently [73]. As another example, in a data warehouse, data are stored in a fact table and a collection of dimension tables using a star schema. The object identifiers and the probability of the existence of the objects may be stored in the fact table or a dimensional table, whereas the scores based on which users want to rank objects may be stored in another dimension table. In such an environment, it needs to join different relations into one relation to have both score and probability together, and to apply one of the existing approaches to probabilistically rank the objects, which can be costly.

Consider a data warehouse that stores textual information, e.g. reviews, shape, price, weight, about products extracted from online shops and forums. The fact table stores the probability of each fact (e.g. review) to be true, and the shape, price and weight information will be stored in other dimensional tables. In order to analyze such kinds of uncertain information, users may want to rank the facts based on a user-specified score function by combining shape, price, weight, and text information. These scoring attributes and the probability attribute are stored in different tables. Also the users may want to specify selection constraints on the facts that should be ranked, e.g. specify the areas where the products are sold, the countries where the products are made, the time interval the facts are extracted. Users are interested in different portions of the whole data, and also in different ranking criteria. In such cases, it is difficult to materialize data for all possible queries.

In this chapter, we study top-k probabilistic ranking queries with joins when scores and probabilities are stored in different relations. To the best of our knowledge, this is the first work in probabilistic ranking under possible worlds semantic that take join issues into consideration. Our work is different from the existing work on rank joins which deal with deterministic data [50, 80]. The main difference is that, for deterministic data, the score of an object can be computed by itself, whereas, for probabilistic data, the probabilistic score of an object cannot be computed by itself, and needs to be computed based on scores and probabilities of other objects.

The main contributions of this chapter are summarized below. We study top-k probabilistic ranking queries when scores and probabilities are stored in differen-

OID	$score$	$prob$
o_1	100	0.3
o_2	95	0.15
o_3	90	0.4
o_4	85	0.1
o_5	80	0.45
o_6	75	0.2
o_7	70	0.2

Table 4.1: An Example Relation (SP)

t relations, and focus on reducing the join cost in probabilistic top-k ranking. We investigate two probabilistic score functions, namely, expected rank value [28] and probability of highest ranking [12]. We give upper/lower bounds of such probabilistic score functions in random access and sequential access, and discuss the advantages/disadvantages of random and sequential accesses. We propose new I/O efficient algorithms to find top-k objects with probabilistic ranking functions, using random access, sequential access, and the combination of random and sequential access by taking the advantages from both random/sequential access. We conduct extensive performance studies, and confirm the effectiveness of our approaches.

The remainder is organized as follows. Section 4.2 gives out our problem statement. In Section 4.3, we discuss bounding schema for two top-k probabilistic ranking queries. In Section 4.4, we propose algorithms to find the top-k answers with respect to the probabilistic ranking function, with random and/or sequential accesses. Experimental studies are reported in Section 4.5.

4.2. Problem Statement

In uncertain data, each tuple o_i contains both a score ($score(o_i)$) and a probability ($p(o_i)$), which play different roles in probabilistic ranking. The $score(o_i)$ is used

rank	tuple	R_E
1	o_3	1.02
2	o_1	1.05
3	o_5	1.17
4	o_2	1.4475
5	o_6	1.56
6	o_7	1.6
7	o_4	1.615

(a) Rank with R_E

rank	tuple	P_{HR}
1	o_1	0.3
2	o_3	0.238
3	o_5	0.144585
4	o_2	0.105
5	o_4	0.0375
6	o_6	0.035343
7	o_7	0.0282744

(b) Rank with P_{HR}

Table 4.2: Two Rankings

to define the relative rank of tuples in a possible instance, whereas $p(o_i)$ is used to measure the probability in all possible instances. In the following, we discuss probabilistic ranking based on a probabilistic score function that combines both $score(o_i)$ and $p(o_i)$, denoted as pscore(o_i). Both R_E (Equ. (2.2)) and P_{HR} (Equ. (2.3)) are such probabilistic score functions.

Based on the probabilistic score function R_E, a Top-k Expected Rank Value (Top-kERV) query returns top-k tuples with lowest R_E values. Based on the probabilistic score function P_{HR}, a Top-k Probable Highest Ranking (Top-kPHR) query returns top-k tuples with highest P_{HR} values.

Table 4.1 shows a relation with 7 tuples $\{o_1, o_2, \cdots, o_7\}$. A tuple o_i is associated with a score ($score(o_i)$) and a probability ($p(o_i)$). For example, tuple o_3 has a score value, $score(o_3) = 90$, and a probability $p(o_3) = 0.4$. Assume an x-Relation $\mathcal{X}$ has 7 x-tuples $\{\tau_1, \tau_2, \cdots, \tau_7\}$, and each x-tuple τ_i has only one alternative tuple o_i in Table 4.1. The ranking based on R_E and P_{HR} are shown in Table 4.2.

All the existing work assume that there is an x-Relation $\mathcal{X}$ which contains both score and probability. However, in real applications, the scores and the probabilities may be stored in different relations. As an example, the same information stored in relation SP (Table 4.1) may be stored in two separated relations, S and P, as shown

OID	$score$
o_1	100
o_2	95
o_3	90
o_4	85
o_5	80
o_6	75
o_7	70

(a) Relation S

OID	$prob$
o_5	0.45
o_3	0.4
o_1	0.3
o_6	0.2
o_7	0.2
o_2	0.15
o_4	0.1

(b) Relation P

Table 4.3: Two Relations (S and P)

in Table 4.3 where $SP = S \bowtie P$.

A naive approach to compute a probabilistic ranking query, when score and probability are not stored in the same relation, is to join the relations followed by applying an existing approach to compute the probabilistic ranking query in the relation. But this naive approach will incur both high computational cost and I/O cost, because it needs to join the whole relations.

Problem Statement: In this chapter, we study how to compute a probabilistic ranking query (Top-kERV or Top-kPHR) by reducing the total I/O cost, when score and probability are not stored in the same relation. We aim at computing the top-k tuples by accessing tuples as least as possible.

In the following, we discuss our approaches using the two relations, S and P, as shown in Table 4.3. We consider two access methods, namely, random access and sequential access.

- For the random access, it assumes that the relation S is sorted in descending order based on the scores. It sequentially accesses the tuples in relation S one-by-one. When it accesses a tuple o_i in relation S in an iteration, it obtains the score value ($score(o_i)$) from the same tuple in relation S, and obtains the probability of the tuple o_i, $p(o_i)$, in relation P using a SQL selection with the same `OID` o_i, which results in a random access of relation P.

- For the sequential access, it assumes that both relation S and relation P are sorted in descending order based on the scores and probabilities, respectively. It sequentially accesses the tuples in relation S and relation P following the descending order of score and probability, respectively. In every iteration, it accesses an additional tuple from relation S and an additional tuple from relation P, respectively. For example, as shown in Table 4.3, in the first iteration, it accesses the tuple identified by o_1 from S and the tuple identified by o_5 from P; in the second iteration, it accesses the tuple identified by o_2 from S and the tuple identified by o_3 from P.

In summary, we consider that the relation S is accessed sequentially in descending score order, which is the access method used in all the existing algorithms when both scores and probabilities are stored in the same relation [86, 93, 45, 28]. The two access methods, namely random/sequential accesses, are about how to access relation P.

The key issue is how many tuples it needs to access in order to compute Top-kERV/Top-kPHR using random access and sequential access. We show that we do not need to compute the exact pscore values (R_E values and P_{HR} values) with the assistance of upper bounds and lower bounds for pscore values. We can compute Top-kERV/Top-kPHR by its relative orders.

4.3. Bounding Ranking Functions

A probabilistic ranking query ranks a tuple, o_i, with a score function pscore(o_i). It is worth noting that the probabilistic score function pscore(o_i) is completely different from those score functions that can be evaluated by the tuple in question and is independent from other tuples. In other words, the probabilistic score function pscore(o_i) needs to be evaluated depending on the $score(o_i)$ and $p(o_i)$ for the tuple o_i itself as well as $score(o_j)$ and $p(o_j)$ for the other tuples o_j. It becomes very important to identify characteristics of the probabilistic score function pscore, especially the monotonicity, upper bounds, and lower bounds.

In this section, first, for simplicity, we focus on independent case such that in an x-Relation $\mathcal{X}$, every x-tuple has only one tuple (alternative). Then we will discuss mutually exclusive case. We assume that all the scores are of different values, it is straightforward to extend to tie scores. We use the notation in a way that the tuples, $o_1, \cdots, o_n$, are in descending score order such that $score(o_i) > score(o_j)$ if $i < j$.

4.3.1. Ranking Function R_E

Assume the tuples $o_1, \cdots, o_n$ are in descending score order, and they are totally independent. The expected size of possible instances is $E[|I|] = \sum p(o_j)$ for all o_j in relation P, and the probabilistic score function $R_E(o_i)$ can be simplified as follows.

$$\begin{aligned}
R_E(o_i) &= \sum_{I \in pwd(\mathcal{X})} \Pr(I) \cdot rank_I(o_i) \\
&= \sum_{o_i \in I} \Pr(I) \cdot rank_I(o_i) + \sum_{o_i \notin I} \Pr(I) \cdot |I| \\
&= p(o_i) \cdot \sum_{j=1}^{i-1} p(o_j) + (1 - p(o_i)) \cdot \sum_{j \neq i} p(o_j) \\
&= p(o_i) \cdot \sum_{j=1}^{i-1} p(o_j) + (1 - p(o_i)) \cdot (E[|I|] - p(o_i)) \\
&= E[|I|] - p(o_i) \cdot (E[|I|] - \sum_{j=1}^{i-1} p(o_j) - p(o_i) + 1)
\end{aligned} \tag{4.1}$$

The details and correctness of Eq. (4.1) are given in [28]. Eq. (4.1) suggests that we cannot simply compute $R_E(o_i)$ even if we have already known its $score(o_i)$ and $p(o_i)$, because it requests us to know $p(o_j)$ for those tuples that $score(o_j) > score(o_i)$. In order to find the top-k tuples without accessing all tuples, we need to bound the R_E value for each seen or unseen tuple. Before discussing bounds, we first prove the monotonicity of R_E below.

Lemma 4.3.1: *The R_E function, on which the Top-kERV query is based, is a monotone function, i.e. for any two tuples o_i and o_j, if $score(o_i) \geq score(o_j)$ and $p(o_i) \geq p(o_j)$, then $R_E(o_i) \leq R_E(o_j)$ and o_i ranks higher than o_j.* □

Proof Sketch: We simplify $p(o_l)$ as p_l in the following proof. Consider $R_E(o_i) - R_E(o_j)$. We have

$$\begin{aligned}
& R_E(o_i) - R_E(o_j) \\
= \ & p_i \sum_{l=1}^{i-1} p_l + (1-p_i)(E[|I|] - p_i) - p_j \sum_{l=1}^{j-1} p_l - (1-p_j)(E[|I|] - p_j) \\
= \ & (p_i - p_j) \sum_{l=1}^{i-1} p_l - p_j \sum_{l=i}^{j-1} p_l - E[|I|](p_i - p_j) + (p_i - p_j)(p_i + p_j - 1) \\
= \ & (p_i - p_j)(\sum_{l=1}^{i} p_l - E[|I|] + p_j - 1) - p_j \sum_{l=i}^{j-1} p_l \\
\leq \ & 0
\end{aligned}$$

where the last inequality holds because $\sum_{l=1}^{i} p_l + p_j \leq E[|I|]$ and $p_i \geq p_j$. Therefore, $R_E(o_i) \leq R_E(o_j)$ and o_i ranks higher than o_j. □

We can bound $R_E(o_i)$, under the random access and sequential access of relation P respectively, where relation S is accessed sequentially in descending score order. We denote the upper and lower bounds as R_E^{up} and R_E^{low}. It is reasonable to assume that $E[|I|]$ is available in advance, because $E[|I|] = \sum p(o_j)$ for all o_j in relation P.

Random access on relation P: For a tuple o_i, we obtain its score $score(o_i)$ when accessing relation S in descending score order and obtain its probability $p(o_i)$ using a SQL selection to access relation P randomly at the same time. Because relation S is sorted in descending score order, we know the probability $p(o_j)$ for all tuples whose scores are larger than that of the tuple o_i in question ($score(o_j) > score(o_i)$).

For each seen tuple o_j, we can compute its exact $R_E(o_j)$ value by Eq. (4.1). Assume that o_i is the last seen tuple after retrieving the tuples $o_1, \cdots, o_{i-1}$. For the unseen tuples o, we can lower bound $R_E(o)$ by the following equation.

$$\begin{aligned}
R_E(o) &= p(o) \cdot \sum_{score(o_j) > score(o)} p(o_j) + (1 - p(o)) \cdot \sum_{o_j \neq o} p(o_j) \\
&\geq p(o) \cdot \sum_{j=1}^{i} p(o_j) + (1 - p(o)) \cdot \sum_{j=1}^{i} p(o_j) \\
&= \sum_{j=1}^{i} p(o_j)
\end{aligned} \tag{4.2}$$

Intuitively, it is lower bounded by the expected size of the possible worlds generated by the tuples $\{o_1, \cdots, o_i\}$. Note that for the Top-kERV query the smaller R_E value the better.

Example 1: Consider relation S and relation P in Table 4.3. Assume the first three tuples in relation S have been retrieved. We get all the scores for o_1, o_2, and o_3, and also get the probability for the three type by random accesses on relation P. The set of seen tuples is $\{o_1(100, 0.3), o_2(95, 0.15), o_3(90, 0.4)\}$, where each entry represents $o_i(score(o_i), p(o_i))$. $E[|I|] = 1.8$. Based on Eq. (4.1), we have $R_E(o_1) = 1.05$, $R_E(o_2) = 1.4475$, and $R_E(o_3) = 1.02$. The lower bound for any unseen tuple o is $R_E^{low}(o) = 0.3 + 0.15 + 0.4 = 0.85$. □

It is difficult to bound $R_E(o)$ tight, if we do not know all the tuples o_j whose scores are larger ($score(o_j) > score(o)$). In other words, all the unseen tuples may have larger scores or none of them have larger score.

Sequential access on relation P: In this scenario, each time we retrieve one entry from S and P in descending score and probability order respectively. For each seen tuple, we may know its score and/or probability. In other words, we may not know both score and probability for every seen tuple. However, in the sequential access, unlike the random access, we have one additional piece of information, the upper bound of all the unknown probabilities, denoted as $\overline{p}$. It is the last retrieved probability from relation P.

Let $\mathcal{H}_s$ denote the set of seen tuples that we know their scores, and let $\mathcal{H}_{\neg s}$ denote the set of seen tuples that we know their probabilities but do not know their scores. In other words, we know $score(o_i)$ for those tuples $o_i \in \mathcal{H}_s$ and $p(o_i)$ only for those tuples $o_i \in \mathcal{H}_{\neg s}$. Furthermore, the tuples o_i that we know both $score(o_i)$ and $p(o_i)$ are also kept in $\mathcal{H}_s$. In particular, we have $\mathcal{H}_s = \mathcal{H}_s^+ \bigcup \mathcal{H}_s^-$, where $\mathcal{H}_s^+$ contains the tuples that we know both score and probability, and $\mathcal{H}_s^-$ contains the tuples we only know their score. In summary, we need to bound R_E for the tuples in $\mathcal{H}_s^+$, $\mathcal{H}_s^-$, $\mathcal{H}_{\neg s}$, and those tuples we have not seen. For the tuples in $\mathcal{H}_s^+$, we need to get both the lower bound and the upper bound, to find the top-k tuples earlier. For the other tuples, we only need to get its lower bound, because its upper bound can be very loose, and we

Set	S	P	Type	Bounds for $E[\lvert I \rvert] - R_E$
$\mathcal{H}_s^+$	✓	✓	*lower*	$p(o_i) \cdot (E[\lvert I \rvert] - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} p(o_j) - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^-}} \bar{p} + 1)$
$\mathcal{H}_s^+$	✓	✓	*upper*	$p(o_i) \cdot (E[\lvert I \rvert] - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} p(o_j) + 1)$
$\mathcal{H}_s^-$	✓	×	*upper*	$\bar{p} \cdot (E[\lvert I \rvert] - \sum_{\substack{j \le i \\ o_j \in \mathcal{H}_s^+}} p(o_j) + 1)$
$\mathcal{H}_{\neg s}$	×	✓	*upper*	$p(o_i) \cdot (E[\lvert I \rvert] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) - p(o_i) + 1)$
U	×	×	*upper*	$\bar{p} \cdot (E[\lvert I \rvert] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) + 1)$

Table 4.4: $E[|I|] - R_E$ Bounds for Tuple o_i (independent)

can not determine any of the other tuples to be in top-k results at this step. If the upper bound R_E of any tuples in $\mathcal{H}_s^+$ is no larger than the lower bound of all the other tuples, then this tuple can be determined in the top-k results. In order to bound R_E, we need the following information.

For the tuples in $\mathcal{H}_s^+$ with known score and probability, we need both lower bound and upper bound of R_E. Consider Eq. (4.1). When $p(o_i)$ is known, the formula can be simplified to the form of $R_E(o_i) = c \cdot \sum_{j=1}^{i-1} p(o_j) + c'$, where $c > 0$ and c' are constants. The lower bound and upper bound are obtained by replacing those unknown $p(o_j)$'s with 0 and $\bar{p}$ respectively.

For those tuples in $\mathcal{H}_s^-$ with known score only, we need to compute its lower bound. It is lower bounded by $E[|I|] - \bar{p} \cdot (E[|I|] - \sum_{j<i, o_j \in \mathcal{H}_s^+} p(o_j) + 1)$.

For those tuples $o_i \in \mathcal{H}_{\neg s}$, we need its lower bound, which can be obtained in a similar way as discussed above. It is lower bounded by $E[|I|] - p(o_i) \cdot (E[|I|] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) - p(o_i) + 1)$. Similarly, we can lower bound $R_E(o_i)$ for the unseen tuples by $E[|I|] - \bar{p} \cdot (E[|I|] - \sum_{o_j \in \mathcal{H}_s^+} p(o_j) + 1)$.

In a summary, the bounds for tuples in different sets are listed in Tab. 4.4. Where the `Set` column is the name of the set that the tuple belongs to, and U denotes the set of unseen tuples. The S and P columns means whether we know the score and probability for the tuple respectively. We show bounds for $E[|I|] - R_E(o_i)$ in Tab. 4.4. All the lower bounds for $E[|I|] - R_E(o_i)$ become upper bounds for $R_E(o_i)$, and upper

Tuple	Randomly Access P		Sequentially Access P	
	$R_E^{low}(o_i)$	$R_E^{up}(o_i)$	$R_E^{low}(o_i)$	$R_E^{up}(o_i)$
o_1	1.05	1.05	1.05	1.05
o_2	1.4475	1.4475	1.05	-
o_3	1.02	1.02	0.96	1.08
o_5	-	-	1.0575	-
unseen	0.85	-	1.17	-

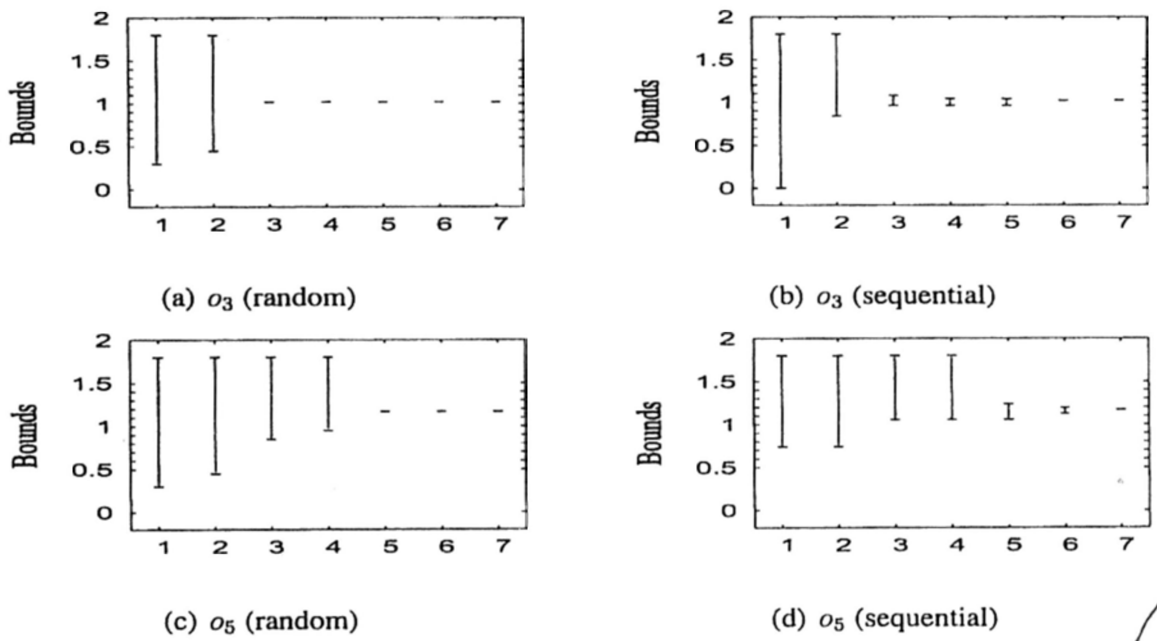
Table 4.5: Bounds of $R_E(o_i)$ in Ran/Seq Access

bounds for $E[|I|] - R_E(o_i)$ become lower bounds for $R_E(o_i)$.

Example 2: Assume that we have retrieved 3 tuples from both relation S and relation P in Table 4.3, respectively. Then, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4)\}$ and $\mathcal{H}_{\neg s} = \{o_5(-, 0.45)\}$. $\mathcal{H}_s$ can be further partitioned into $\mathcal{H}_s^+ = \{o_1(100, 0.3), o_3(90, 0.4)\}$ and $\mathcal{H}_s^- = \{o_2(95, -)\}$. "-" means the value of that field is unknown. Here, $\bar{p} = p(o_1) = 0.3$, which is the last probability we have seen. $E[|I|] = 1.8$. For tuple o_1, no tuple has a larger score. $p(o_1) = 0.3$, and then $R_E(o_1) = 1.8 - 0.3 \times (1.8 - 0.3 + 1) = 1.05$. For tuple o_2, tuple o_1 is the only tuple with a larger score, and we do not know the probability of o_2. We compute its lower bound, which is $R_E^{low}(o_2) = 1.8 - 0.3 \times (1.8 - 0.3 + 1) = 1.05$. For tuple o_3, tuple o_1 and o_2 are the tuples with a larger score, and $p(o_3) = 0.4$. $R_E(o_3) = 1.8 - 0.4 \times (1.8 - 0.3 - p(o_2) - 0.4 + 1) = 0.96 + 0.4 \times p(o_2)$, then $R_E^{up}(o_3) = 0.96 + 0.4 \times 0.3 = 1.08$ and $R_E^{low}(o_3) = 0.96$. For tuple o_5, we do not know its exact score, and only know that the tuples in $\mathcal{H}_s$ are with a larger score. Hence, $R_E^{low}(o_5) = E[|I|] - p(o_5) \times (E[|I|] - p(o_1) - p(o_2) - p(o_3) - p(o_5) + 1) \geq 1.8 - 0.45 \times (1.8 - 0.3 - 0 - 0.4 - 0.45 + 1) = 1.0575$, with a lower bound 1.0575. For all the unseen tuples o_i, $R_E^{low}(o_i) = E[|I|] - \bar{p} \times (E[|I|] - p(o_1) - p(o_2) - p(o_3) + 1) \geq 1.17$, with a lower bound 1.17. □

Tab. 4.5 summarizes the bounds for seen and unseen tuples, after three iterations of random/sequential accesses on relation P respectively. The $R_E(o_i)$ for o_1, o_2, and o_3, are exact values in random access, which is tighter compared to the bounds in sequential access. But the lower bound for unseen tuples in random access is looser, i.e. $R_E^{low}(o) = 0.85$, whereas $R_E^{low}(o) = 1.17$ in sequential access, which is tighter.

Fig. 4.1 shows the [lower bound, upper bound] interval for tuple o_3 and o_5 in every iteration from 1 to 7. As we get more information, the lower bound goes non-decrease, and the upper bound goes non-increase, eventually we get the exact R_E values.



(a) o_3 (random)
(b) o_3 (sequential)
(c) o_5 (random)
(d) o_5 (sequential)

Figure 4.1: R_E Bound Changes for o_3 and o_5

Lemma 4.3.2: *Our bounding scheme is correct.* □

The bounding scheme is correct based on Lemma 4.3.1.

4.3.2. Ranking Function P_{HR}

If the tuples $o_1, \cdots, o_n$ are in descending score order and are totally independent, the $P_{HR}(o_i)$ function can be simplified as below.

$$P_{HR}(o_i) = p(o_i) \times \prod_{j=1}^{i-1}(1 - p(o_j)) \tag{4.3}$$

It is the product of the nonexistence probability of tuples o_j that have a larger score than the score of o_i and the probability of tuple o_i itself. We prove that P_{HR} is a monotone function in the following.

Lemma 4.3.3: *The P_{HR} function, on which the Top-kPHR query is based, is a monotone function. For any two tuples, o_i and o_j, if $score(o_i) \geq score(o_j)$ and $p(o_i) \geq p(o_j)$, then $P_{HR}(o_i) \geq P_{HR}(o_j)$.* □

Proof Sketch: As all the tuples are totally independent, the following equation holds.

$$\frac{P_{HR}(o_j)}{P_{HR}(o_i)} = \frac{p(o_j)}{p(o_i)} \times \prod_{l=i}^{j-1}(1 - p(o_l))$$

Here, the first part $p(o_j)/p(o_i) \leq 1$ because $p(o_i) \geq p(o_j)$, and the second part is no larger than one too. Therefore the probabilistic score function P_{HR} is monotone, $P_{HR}(o_i) \geq P_{HR}(o_j)$, if $score(o_i) \geq score(o_j)$ and $p(o_i) \geq p(o_j)$. □

Let the upper and lower bounds of $P_{HR}(o_i)$ be $P_{HR}^{up}(o_i)$ and $P_{HR}^{low}(o_i)$, respectively. We consider random and sequential accesses on relation P respectively.

Random access on relation P: In this scenario, we get the probability for each seen tuple by a random access on relation P. For any seen tuple we know its exact P_{HR} value. For all the unseen tuples we can upper bound it by $\prod_{j=1}^{i}(1 - p(o_j))$, where o_i is the last accessed tuple from relation S. Note that this upper bound is tight, because an unseen tuple o_{i+1} may have probability 1.

Example 3: Consider relation S and relation P in Table 4.3. Assume the first three tuples in relation S have been retrieved. The set of seen tuples is $\{o_1(100, 0.3), o_2(95, 0.15), o_3(90, 0.4)\}$. Based on Eq. (4.3), we have $P_{HR}(o_1) = 0.3$,

Set	S	P	Type	Bounds for P_{HR}
$\mathcal{H}_s^+$	✓	✓	*lower*	$p(o_i) \cdot \prod_{\substack{j \leq i \\ o_j \in \mathcal{H}_s^+}} (1 - p(o_j)) \cdot \prod_{\substack{j \leq i \\ o_j \in \mathcal{H}_s^-}} (1 - \bar{p})$
$\mathcal{H}_s^+$	✓	✓	*upper*	$p(o_i) \cdot \prod_{\substack{j \leq i \\ o_j \in \mathcal{H}_s^+}} (1 - p(o_j))$
$\mathcal{H}_s^-$	✓	×	*upper*	$\bar{p} \cdot \prod_{\substack{j \leq i \\ o_j \in \mathcal{H}_s^+}} (1 - p(o_j))$
$\mathcal{H}_{\neg s}$	×	✓	*upper*	$p(o_i) \cdot \prod_{o_j \in \mathcal{H}_s^+} (1 - p(o_j))$
U	×	×	*upper*	$\bar{p} \cdot \prod_{o_j \in \mathcal{H}_s^+} (1 - p(o_j))$

Table 4.6: P_{HR} Bounds for Tuple o_i (independent)

$P_{HR}(o_2) = 0.105$, and $P_{HR}(o_3) = 0.238$. The upper bound for any unseen tuple, o, is $P_{HR}^{up}(o) = (1 - 0.3) \cdot (1 - 0.15) \cdot (1 - 0.4) = 0.357$. This upper bound is achieved by giving the unseen tuple with highest score a probability 1. That is, o_4 is estimated to have a probability 1. □

Sequential access on relation P: In this scenario, in every iteration, we retrieve a tuple from relation S and a tuple from relation P in descending score/probability order respectively. We know the score and/or probability for the retrieved tuples from both relations. Let $\bar{p}$ denote the last retrieved probability from P.

For a tuple $o_i \in \mathcal{H}_s$, all the tuples o_j such that $score(o_j) > score(o_i)$ are in $\mathcal{H}_s$ already. We compute $P_{HR}(o_i)$ using Eq. (4.3). However, because the probability for some tuples may be unknown, we need to upper bound and lower bound $P_{HR}(o_i)$. If $p(o_i)$ is unknown, then we upper bound it by $\bar{p}$, and lower bound it by 0. For each tuple o_j involved in Eq. (4.3) to compute $P_{HR}(o_i)$, the upper/lower bounds of $(1 - p(o_j))$ are 1 and $(1 - \bar{p})$ respectively.

For a tuple $o_i \in \mathcal{H}_{\neg s}$, we upper/lower bound $P_{HR}(o_i)$ value. Note that its lower bound is 0. All the tuples in $\mathcal{H}_s$ have a score larger than $score(o_i)$. We upper bound $P_{HR}(o_i)$ by multiplying $p(o_i)$ and $(1 - p(o_j))$ for all the tuples in $\mathcal{H}_s$ as discussed above. Note that there may exist some tuple o_j that has a larger score than o_i but has not been retrieved from relation S yet. Similarly, we upper bound $P_{HR}(o)$ for the

Tuple	Randomly Access P		Sequentially Access P	
	$P_{HR}^{up}(o_i)$	$P_{HR}^{low}(o_i)$	$P_{HR}^{up}(o_i)$	$P_{HR}^{low}(o_i)$
o_1	0.3	0.3	0.3	0.3
o_2	0.105	0.105	0.21	-
o_3	0.238	0.238	0.28	0.196
o_5	-	-	0.189	-
unseen	0.357	-	0.126	-

Table 4.7: Bounds of $P_{HR}(o_i)$ in Ran/Seq Access

unseen tuples by multiplying $\bar{p}$ and $(1 - p(o_j))$ for all the tuples in $\mathcal{H}_s$ as discussed above.

In a summary, the bounds of P_{HR} for tuples in different sets are listed in Tab. 4.6.

Example 4: Assume that we have retrieved 3 tuples from both relation S and relation P in Table 4.3, respectively. Then, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4)\}$ and $\mathcal{H}_{\neg s} = \{o_5(-, 0.45)\}$. $\mathcal{H}_s$ can be further partitioned into $\mathcal{H}_s^+ = \{o_1(100, 0.3), o_3(90, 0.4)\}$ and $\mathcal{H}_s^- = \{o_2(95, -)\}$. Here, $\bar{p} = 0.3$, which is the last probability we have seen. In the following, we compute upper bounds and lower bounds for all the partial/full seen tuples. For tuple o_1, no tuple has a larger score. $p(o_1) = 0.3$, then $P_{HR}(o_1) = p(o_1) = 0.3$. For tuple o_2, tuple o_1 is the only tuple with a larger score, but we do not know the probability of o_2. $P_{HR}(o_2) = p(o_2) \times (1 - p(o_1))$, then $P_{HR}^{up}(o_2) = \bar{p} \times (1 - p(o_1)) = 0.21$ and $P_{HR}^{low}(o_2) = 0$. For tuple o_3, tuple o_1 and o_2 are the tuples with larger scores, and $p(o_3) = 0.4$. $P_{HR}(o_3) = p(o_3) \times (1 - p(o_1)) \times (1 - p(o_2)) = 0.4 \times (1 - 0.3) \times (1 - p(o_2)) = 0.28 \times (1 - p(o_2))$, then $P_{HR}^{up}(o_3) = 0.28$ and $P_{HR}^{low}(o_3) = 0.28 \times (1 - \bar{p}) = 0.196$. For tuple o_5, we do not know its exact score, but we know that the tuples in $\mathcal{H}_s$ are with a larger score. So $P_{HR}^{up}(o_5) = p(o_5) \times (1 - p(o_1)) \times (1 - p(o_2)) \times (1 - p(o_3)) \leq 0.45 \times (1 - 0.3) \times (1 - 0) \times (1 - 0.4) = 0.189$, with an upper bound 0.189. For all the unseen tuples o_i, $P_{HR}^{up}(o_i) = \bar{p} \times (1 - p(o_1)) \times (1 - p(o_2)) \times (1 - p(o_3)) \leq 0.126$, with an upper bound 0.126. □

Tab. 4.7 summarizes the bounds for seen and unseen tuples, after three iterations of random/sequential accesses on relation P respectively. The $P_{HR}(o_i)$ for o_1, o_2, and o_3, are exact values in random access, which is tighter compared to the bounds in sequential access. But the upper bound for unseen tuples is a little looser, i.e., $P_{HR}^{up}(o) = 0.357$, whereas $P_{HR}^{up}(o) = 0.126$ in sequential access, which is tighter.

Lemma 4.3.4: *Our bounding scheme is correct and tight among all possible bounding schema provided that relation P is sorted in descending probability order.* □

Proof Sketch: The correctness of our bounding scheme directly follows from Lemma 4.3.3. For the tightness, we assume that a tuple may have zero probability. There does not exist any other bounding scheme (without random guess) that is more tight than ours. Based on Eq. (4.3), the lower bound is achieved, when all the tuples with a larger score has probability $\bar{p}$; and the upper bound is achieved when all the tuples with larger score has probability 0. Note that the upper bound and lower bound are achievable individually. □

4.3.3. Ranking Function with Exclusive Relationship

In the previous section, we discussed the bounds for R_E and P_{HR} respectively, assuming that all the tuples are independent. In this section, in a general, we discuss bounds for R_E and P_{HR} in an x-Relation with exclusive relationships.

Let $o_i \diamond o_j$ denote that tuple o_i and o_j are mutually exclusive, i.e. they belong to the same x-tuple τ, $o_i \in \tau$ and $o_j \in \tau$, and let $o_i \bar{\diamond} o_j$ denote that tuple o_i and o_j are from different x-tuples (independent). Note that o_i and o_j are different tuples. Let τ_{o_i} denote the x-tuple that o_i belongs to, i.e., $\tau_{o_i} = \{o_j \mid o_j \diamond o_i\} \cup \{o_i\}$.

R_E Function

Assume the tuples $o_1, \cdots, o_n$ are in descending score order, the $R_E(o_i)$ with exclusive relationship is as follows.

$$
\begin{aligned}
& R_E(o_i) \\
= \;& p(o_i) \cdot \sum_{o_j \bar{\diamond} o_i, j<i} p(o_j) + (1 - p(o_i)) \cdot (\frac{\sum_{o_j \diamond o_i} p(o_j)}{1 - p(o_i)} + \sum_{o_j \bar{\diamond} o_i} p(o_j)) \\
= \;& p(o_i) \cdot (\sum_{j<i} p(o_j) - \sum_{o_j \diamond o_i, j<i} p(o_j)) + \sum_{o_j \diamond o_i} p(o_j) \\
& +(1 - p(o_i)) \cdot (E[|I|] - p(o_i) - \textstyle\sum_{o_j \diamond o_i} p(o_j)). \\
= \;& E[|I|] - p(o_i) \cdot (E[|I|] - \sum_{j<i} p(o_j) - p(o_i) + 1) + p(o_i) \cdot \sum_{o_j \diamond o_i, j>i} p(o_j)
\end{aligned}
\tag{4.4}
$$

where the first equation is from [28]. Compared with R_E in the independent case, there is one extra term $p(o_i) \cdot \sum_{o_j \diamond o_i, j>i} p(o_j)$.

When randomly accessing relation P, the lower bound for the unseen tuples is $R_E(o_i) \geq \sum_{j \leq i} p(o_j)$, which is the same as Eq. (4.2). However, the $R_E(o_i)$ for the seen tuples can not be bounded tightly. Even though we have retrieved all the tuples with higher score and their probabilities, we still do not know those tuples in the same x-tuple with o_i, i.e., the term $\sum_{o_j \diamond o_i, j>i} p(o_j)$ is unknown.

When sequentially accessing relation P, the bounding scheme is more complicated compared to that discussed in Section 4.3.1. In Section 4.3.1, $p(o_i)$ for unknown probability is bounded by $0 \leq p(o_i) \leq \bar{p}$. But, when mutually exclusive exists, it is upper bounded by

$$\min\{\bar{p}, 1 - \sum_{o_j \diamond o_i, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j)\}$$

When $\sum_{o_j \diamond o_i, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j) > 1 - \bar{p}$, $p(o_i)$ must be less than $\bar{p}$. In order to bound $R_E(o_i)$ when mutually exclusive exists, we reorganize Eq. (4.4) in the granularity of x-tuples. It is possible to get better bounds, because $\sum_{o_j \in \tau} p(o_j) \leq 1$. Eq. (4.4) is reorganized as follows.

$$\begin{aligned} & R_E(o_i) \\ = \; & E[|I|] - p(o_i) \cdot (E[|I|] + 1 - \sum_{j<i} p(o_j) - p(o_i) - \sum_{\substack{o_j \circ o_i \\ j>i}} p(o_j)) \\ = \; & E[|I|] - p(o_i) \cdot (E[|I|] + 1 - \sum_{\substack{\tau \in \mathcal{X} \\ o_i \notin \tau}} \sum_{\substack{o_j \in \tau \\ j<i}} p(o_j) - \sum_{o_j \in \tau_{o_i}} p(o_j)) \end{aligned} \tag{4.5}$$

The x-tuples are independent, so are their bounds. For the term $\sum_{o_j \in \tau, j<i} p(o_j)$, the lower bound is obtained by replacing all the unknown $p(o_j)$ with 0, i.e., $\sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+, j<i} p(o_j)$. There exist two possible upper bounds. One is to replace all of the unknown $p(o_j)$ with $\bar{p}$, i.e., $\sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+, j<i} p(o_j) + \bar{p} \cdot |\{o_j \in \tau \mid o_j \in \mathcal{H}_s^-, j < i\}|$, where $|\cdot|$ is the size of a set. The other is one minus the summation of the probabilities for tuples in τ that has a larger score than $score(o_i)$, i.e., $1 - \sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+ \&\& j>i || o_j \in \mathcal{H}_{\neg s}} p(o_j)$. The upper bound is the minimum of the two. Similarly, for the term $\sum_{o_j \in \tau_{o_i}} p(o_j)$, its lower bound can be obtained by replacing all the unknown $p(o_j)$ with 0, i.e., $\sum_{o_j \in \tau_{o_i}, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j)$, the upper bound can be obtained by replacing all the unknown $p(o_j)$ for tuples in τ_{o_i} with $\bar{p}$, if we can get the size information about x-tuple τ_{o_i}, otherwise, it can only be trivially upper bounded by 1. By combining the corresponding lower bounds and upper bounds for each term, we get the lower bound and upper bound for Eq. (4.5).

In a summary, the bounds of $E[|I|] - R_E$ for tuples in different sets are shown in Tab. 4.8.

Example 5: Consider the two relations S and P in Tab. 4.3, assume there is a mutually exclusive relationship between tuple o_1 and o_6, i.e. $\tau_1 = \{o_1, o_6\}$, and the other tuples are independent. The x-tuple information can be possibly maintained in relation S, in an additional column named XID. The tuples identified by unique OID share the same XID if they belong to the same x-tuple. In addition, we add an additional column called Xcnt which records the number of alternatives an x-tuple has. With this additional column Xcnt, we can achieve tighter bound. It is achieved by the following information. For example, when retrieving o_6 from rela-

Set	**S**	**P**	**Type**	**Bounds for $E[\|I\|] - R_E$**
$\mathcal{H}_s^+$	✓	✓	*upper*	$p(o_i) \cdot (E[\|I\|] + 1 - \sum_{\tau \in \mathcal{X}, o_i \notin \tau} \sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+, j<i} p(o_j)$ $- \sum_{o_j \diamond o_i, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j) - p(o_i))$
$\mathcal{H}_s^+$	✓	✓	*lower*	$p(o_i) \cdot (E[\|I\|] + 1$ $- \sum_{\substack{\tau \in \mathcal{X} \\ o_i \notin \tau}} \min\{\sum_{\substack{o_j \in \tau \\ o_j \in \mathcal{H}_s^+ \\ j<i}} p(o_j) + \bar{p} \cdot \|\{o_j \in \tau \mid o_j \in \mathcal{H}_s^-, j < i\}\|,$ $1 - \sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+, j>i} p(o_j) - \sum_{o_j \in \tau, o_j \in \mathcal{H}_{\neg s}} p(o_j)\}$ $- \min\{\sum_{o_j \diamond o_i, o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j) + p(o_i)$ $+\bar{p} \cdot (\|\tau_{o_i}\| - \|\{o_j \diamond o_i \mid o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}\}\| - 1), 1\})$
$\mathcal{H}_s^-$	✓	×	*upper*	$\min\{\bar{p}, 1 - \sum_{\substack{o_j \diamond o_i \\ o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}}} p(o_j)\}$ $\cdot (E[\|I\|] + 1 - \sum_{\substack{\tau \in \mathcal{X} \\ o_i \notin \tau}} \sum_{\substack{o_j \in \tau \\ o_j \in \mathcal{H}_s^+ \\ j<i}} p(o_j) - \sum_{\substack{o_j \diamond o_i \\ o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}}} p(o_j))$
$\mathcal{H}_{\neg s}$	×	✓	*upper*	$p(o_i) \cdot (E[\|I\|] + 1 - \sum_{\substack{o_j \bar{\diamond} o_i \\ o_j \in \mathcal{H}_s^+}} p(o_j) - \sum_{\substack{o_j \in \tau_{o_i} \\ o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}}} p(o_j))$
U	×	×	*upper*	$\bar{p} \cdot (E[\|I\|] + 1 - \sum_{o_j \in \mathcal{H}_s^+} p(o_j))$

Table 4.8: $E[|I|] - R_E$ Bounds for Tuple o_i (independent/exclusive)

tion S by sequential access, we get the information that all the alternatives of x-tuple τ_1 have been retrieved, because τ_1 has only two alternatives and the other alternative o_1 has been retrieved. $E[|I|] = 1.8$. After three sequential accesses on both S and P, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4)\}$, $\mathcal{H}_{\neg s} = \{o_5(-, 0.45)\}$, and $\bar{p} = 0.3$. We also know that tuple o_2 and o_3 have no other alternatives from the same x-tuple, and there are more alternatives from the x-tuple that contains o_1. For tuple o_1, $R_E(o_1) = E[|I|] - p(o_1)(E[|I|] + 1 - \sum_{o_j \in \tau_{o_1}} p(o_j)) = 0.96 + 0.3 \sum_{o_j \in \tau_{o_1}} p(o_j)$, where $\sum_{o_j \in \tau_{o_1}} p(o_j)$ is lower bounded by $p(o_1) = 0.3$ and upper bounded by $p(o_1) + \bar{p} = 0.6$. Then $R_E^{low}(o_1) = 1.05$ and $R_E^{up}(o_1) = 0.96 + 0.3 \times 0.6 = 1.14$. For tuple o_2, $R_E(o_2) = E[|I|] - p(o_2)(E[|I|] - p(o_1) - p(o_2) + 1)$, and then $R_E^{low}(o_2) = 1.8 - 0.3 \times (1.8 - 0.3 + 1) = 1.05$. For tuple o_3, $R_E^{low}(o_3) = 0.96$ and $R_E^{up}(o_3) = 1.08$. For tuple o_5, $R_E^{low}(o_5) = 1.0575$. And $R_E^{low}(o) = 1.17$ for unseen tuples. □

In this example, although we know both score and probability for o_1 and o_1 has the highest score, we can not get the exact $R_E(o_1)$, because there may exist some tuples with a smaller score and in the same x-tuple where o_1 belongs to.

P_{HR} Function

Assume the tuples $o_1, \cdots, o_n$ are in descending score order, the $P_{HR}(o_i)$ with exclusive relationship is as follows.

$$P_{HR}(o_i) = p(o_i) \times \prod_{\tau \in \mathcal{X}, o_i \notin \tau} (1 - \sum_{o_j \in \tau, j<i} p(o_j)) \tag{4.6}$$

which is the multiplication of the existence probability of o_i and the nonexistence probabilities of other tuples with higher score. Note that, the multiplication is in the granularity of x-tuple, because the tuples from an x-tuple are mutually exclusive.

When randomly accessing relation P, we only need to upper bound the unseen tuples. Let o_i be the last seen tuple, the P_{HR} for unseen tuples can be upper bounded by $\prod_{\tau \in \mathcal{X}}(1 - \sum_{o_j \in \tau, j<i} p(o_j))$.

When sequentially accessing relation P in descending probability order, we need both upper and lower bounds for seen tuples. In Eq. (4.6), if $p(o_i)$ is unknown, then the upper bound is $\min\{\bar{p}, 1 - \sum_{o_j \diamond o_i} p(o_j)\}$ and the lower bound is 0. For each unknown $p(o_j)$ in the second part of Eq. (4.6), we replace it by 0 for the upper bound, and by $\bar{p}$ for the lower bound. If the lower bound is negative, then its lower bound is 0.

For each $o_i \in \mathcal{H}_{\neg s}$ whose score is unknown, we upper bound it by $P_{HR}^{up}(o_i) = p(o_i) \times \prod_{\tau \in \mathcal{X}, o_i \notin \tau}(- \sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+} p(o_j))$. For the unseen tuples, we can upper bound it by

$$\max_{\tau \in \mathcal{X}}\{\frac{\min\{\bar{p}, 1 - \sum_{o_j \in \mathcal{H}_s^+ \cup \mathcal{H}_{\neg s}} p(o_j)\}}{1 - \sum_{o_j \in \mathcal{H}_s^+} p(o_j)}\} \prod_{\tau \in \mathcal{X}} (1 - \sum_{o_j \in \tau, o_j \in \mathcal{H}_s^+} p(o_j))$$

4.3.4. Discussions

In this section, we discuss two issues. One is the advantages and disadvantages related to random/sequential access. The other is the bounding scheme for other possible top-k probabilistic ranking queries.

Random vs Sequential access: We discuss the advantages and disadvantages of the bounds for random and sequential accesses. Consider at the iteration i, we distinguish the whole set of (seen or unseen) tuples into two sets $\mathcal{H}_s$ and $\mathcal{H}_{\neg s} \cup U$.

When randomly accessing relation P, we retrieve the probability by a random access each time we retrieve the corresponding tuple from relation S. $\mathcal{H}_{\neg s} = \emptyset$ and $\mathcal{H}_s = \mathcal{H}_s^+$. For tuples $o_i \in \mathcal{H}_s^+$, we can get the exact pscore(o_i) values, both upper bound and lower bound are pscore(o_i) which is absolutely tight. But, for tuples $o_i \in U$, we do not know any information about the score and probability. Then, the upper bound for pscore(o_i) can be arbitrarily loose.

When sequentially accessing relation P, we have additional information $\bar{p}$, which is the upper bound for all the unknown probabilities. We can upper bound pscore(o_i) for tuples $o_i \in \mathcal{H}_{\neg s} \cup U$ more tighter than that in random access. But, among tuples $o_i \in \mathcal{H}_s$, the probability for some tuples may be unknown, we can only bound it by 0 from below and $\bar{p}$ from above. The lower/upper bound for pscore(o_i) is a little looser than that in random access.

In summary, with random access on relation P, we can get better bounds of pscore(o_i) for tuples in $\mathcal{H}_s$. With sequential access on relation P, we can get better bounds of pscore(o_i) for tuples in $\mathcal{H}_{\neg s} \cup U$. But, we can not get better bounds for both tuples in $\mathcal{H}_s$ and tuples in $\mathcal{H}_{\neg s} \cup U$ at the same time, in either random access or sequential access.

The bounds for other functions: There are also other probabilistic ranking functions used in the literature, e.g., top-k probability [45], or the probability for a tuple to be ranked at the j-th position in possible worlds [86]. The same approach in the literature can be used to bound pscore in random access, because it is the same as to process in the same relation. However, it is very difficult to find an upper bound or nontrivial lower bound (other than 0) for these probabilistic ranking functions, in sequential access of relation P. Below, we discuss why it is difficult to find an upper bound for the probability that a tuple ranked at the j-th position in possible worlds.

Suppose we have done 4 sequential accesses, and we have $\mathcal{H}_s = \{o_1(100, p_1), o_2(95, p_2), o_3(90, p_3), o_4(85, 0.4)\}$, where p_i is short for $p(o_i)$ and is unknown. We only know that p_1, p_2, and p_3, are in the range of $[0, \bar{p}]$, where $\bar{p}$ is the last seen probability. Then the probability for o_4 to be ranked at the second place in possible worlds is as follows.

$$p(o_4) \cdot (p_1(1-p_2)(1-p_3) + p_2(1-p_1)(1-p_3) + p_3(1-p_1)(1-p_2))$$

which is $0.4 \cdot (3p_1p_2p_3 - 2p_1p_2 - 2p_1p_3 - 2p_2p_3 + p_1 + p_2 + p_3)$. It is a polynomial of degree 3, it is hard to get the upper bound. A very loose upper bound is $0.4 \cdot (3p_1p_2p_3 + p_1 + p_2 + p_3)$, where p_1, p_2, p_3 is replaced by $\bar{p}$. This upper bound can be very loose, and can be arbitrarily large when there are more than 3 tuples with unknown probability. Even worse, in order to get such upper bound, it takes $O(2^n)$ time in order to compute the probability for a tuple to be ranked at the second position, where n is the number of probabilities that are unknown.

4.4. I/O Efficiency

We discussed the bounding schema for both R_E and P_{HR} in Section 4.3. In this section, we discuss two algorithms for random access and sequential access of relation P respectively for Top-kERV queries when all x-tuples are independent. It is straightforward to extend the algorithms to support the general mutually exclusive case, and it is straightforward to extend the algorithms to compute probability of highest rank queries using the bounding scheme of P_{HR} instead of R_E. We also discuss how to combine the advantages from both random/sequential access.

Algorithm 4 PRR(S, P, k)

Input: relation S, relation P, and a number k

Output: Top-k tuples in sorted order based on R_E.

```
1: initialize a priority queue of size $k$, $\mathcal{Q}$, to be empty;
2: while less than $k$ tuples reported do
3:     $(o_i, s_i) \leftarrow next(S)$;
4:     $(o_i, p_i) \leftarrow find(P, o_i)$;
5:     compute $R_E(o_i)$ using Eq. (4.1);
6:     insert $(o_i, R_E(o_i))$ into $\mathcal{Q}$;
7:     compute the lower bound of all the unseen tuples as $R_E^{low}(o)$;
8:     while less than $k$ tuples reported do
9:         let $o_i$ be the tuple with largest $R_E(o_i)$ in $\mathcal{Q}$.
10:        if $R_E(o_i) \leq R_E^{low}(o)$ then
11:            report $o_i$ as the next tuple in the top-k answers;
12:            delete $o_i$ from $\mathcal{Q}$;
13:        else
14:            break;
15:        end if
16:    end while
17: end while
```

4.4.1. Random Access on Relation P

The algorithm for random access on relation P is similar to the algorithms for probabilistic rank queries in a single relation.

Algorithm 4 shows the detailed steps of computing Top-kERV queries. It takes three inputs: a relation S which is sorted in descending score order, a relation P, and a number k. It uses a priority queue $\mathcal{Q}$ of size k which is initialized to be empty (line 1). It outputs the top-k answers in a while loop (line 2-14), and will stop when top-k answers are output. In the while loop, it gets the next pair (o_i, s_i) from relation S

where $s_i = score(o_i)$ (line 3). The score will be the largest among those unseen tuples in relation S, because relation S is sorted in descending score order. Then, it obtains $p(o_i)$ by calling $find(P, o_i)$ with the OID o_i by a random access (line 4). It computes $R_E(o_i)$ using Eq. (4.1) (line 5), and inserts the pair $(o_i, R_E(o_i))$ into the priority queue $\mathcal{Q}$ where tuples are sorted in ascending $R_E(o_i)$ order (line 6). If its size exceeds k when inserting a pair into $\mathcal{Q}$, we delete the pair with the largest R_E value from $\mathcal{Q}$. It also computes the lower bound for all the unseen tuples denoted as $R_E^{low}(o)$, using the bounding approach discussed in Section 4.3.1 (line 7). All the tuples o_i in $\mathcal{Q}$ with $R_E(o_i) \leq R_E^{low}(o)$ can be determined to be the top-k answers (line 8-14). Note that in Algorithm 4 the numbers of tuples retrieved from relation S and relation P are the same.

Example 6: Consider the two relations, S and P, in Table 4.3. Let $k = 2$. Algorithm 4 executes as follows. In the first two iterations, $(o_1, 100)$ and $(o_1, 0.3)$, and $(o_2, 95)$ and $(o_2, 0.15)$, are retrieved from relation S and relation P. In the third iteration, $(o_3, 90)$ and $(o_3, 0.4)$ are retrieved. We have $R_E(o_1) = 1.05$, $R_E(o_2) = 1.4475$, and $R_E(o_3) = 1.02$. The lower bound is computed as $R_E^{low}(o) = 0.85$. All these $R_E(o_1)$, $R_E(o_2)$, and $R_E(o_3)$ are larger than the lower bound $R_E^{low}(o)$. Therefore, no tuples can be determined to be in the top-k answers in this iteration. In the fourth iteration, we retrieve o_4 from both relations: $(o_4, 85)$ and $(o_4, 0.1)$. $R_E(o_4) = 1.615$. The lower bound computed is $R_E^{low}(o) = 0.95$. No tuple can be determined to be the top-k answers. In the fifth iteration, we retrieve o_5 from both relations: $(o_5, 80)$ and $(o_5, 0.45)$. $R_E(o_5) = 1.17$. The lower bound computed is $R_E^{low}(o) = 1.4$. $R_E(o_3)$ and $R_E(o_1)$ are smaller than $R_E^{low}(o)$, then o_3 and o_1 can be determined to be the top-2 results. □

Theorem 4.4.1: *Algorithm 4 correctly finds the top-k tuples with respect to R_E, with sequential access on relation S and random access on relation $\mathcal{P}$.* □

Proof Sketch: For all the seen tuples o_i, we can compute $R_E(o_i)$ exactly by Eq. (4.1). For the unseen tuples o, it can be lower bounded as $R_E^{low}(o)$. All the tuples output by Algorithm 4 are guaranteed to be no larger than $R_E^{low}(o)$ (line 10). Then it is guaranteed to be in the top-k answers. □

4.4.2. Sequential Access on Relation P

Because random access usually is much expensive compared to sequential access, in this section, we consider sequential accessing relation P provided that relation P is sorted in descending order in terms of probability, as well as sequential accessing relation S which is sorted in descending score order.

We sequentially access relation S and relation P. In every iteration, we retrieve (o_i, s_i) from relation S, and (o_j, p_j) from relation P, update $\bar{p}$ to be p_j, where $\bar{p}$ is the upper bound for all the unseen probabilities. We update $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$ which may require joining (o_i, s_i) and (o_j, p_j) with the existing retrieved tuples in $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$. We also update the upper bound and lower bound for all the seen tuples, and compute the lower bound for all the unseen tuples $R_E^{low}(o)$. Let o_i be the tuple with smallest lower bound among all seen tuples. If $R_E^{up}(o_i)$ is no larger than all the other lower bounds, then tuple o_i can be determined to be the next tuple in the top-k answers. Algorithm 5 shows the detailed steps. We explain it using an example.

Example 7: Consider the two relations, S and P, in Table 4.3. Let $k = 2$. Algorithm 5 executes as follows. In the first iteration, $(o_1, 100)$ and $(o_5, 0.45)$ are retrieved from relation S and relation P (line 2-3). $\bar{p} = 0.45$ (line 4). Here, $\mathcal{H}_s = \{(o_1(100, -)\}$ and $\mathcal{H}_{\neg s} = \{(o_5(-, 0.45)\}$, where every entry in $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$ represents $o_i(score(o_i), p(o_i))$. In the second iteration, $(o_2, 95)$ and $(o_3, 0.4)$ are retrieved from relation S and relation P. $\bar{p} = 0.4$. Here, $\mathcal{H}_s = \{(o_1(100, -), o_2(95, -)\}$ and $\mathcal{H}_{\neg s} = \{(o_5(-, 0.45), o_3(-, 0.4)\}$. The upper bounds for R_E of all the seen tuples are $E(|I|)$, no tuples can be the top-k answers.

In the third iteration, after retrieving $(o_3, 90)$ from relation S and $(o_1, 0.3)$ from relation P, we update $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$ (line 5). Here, $\mathcal{H}_s = \{(o_1(100, 0.3), o_2(95, -),$ $o_3(90, 0.4)\}$ and $\mathcal{H}_{\neg s} = \{(o_5(-, 0.45)\}$. Note that the entry $o_3(-, 0.4)$ is deleted from $\mathcal{H}_{\neg s}$ and its probability is added into $o_3(90, 0.4)$ in $\mathcal{H}_s$. The same is applied to o_1.

Algorithm 5 PRS(S, P, k)

Input: relation S, relation P, and a number k

Output: Top-k tuples in sorted order based on R_E.

```
1: while less than $k$ tuples reported do
2:    $(o_i, s_i) \leftarrow next(S)$;
3:    $(o_j, p_j) \leftarrow next(P)$;
4:    $\bar{p} \leftarrow p_j$;
5:    update $\mathcal{H}_s$ and $\mathcal{H}_{\neg s}$;
6:    for all the tuples $o_i \in \mathcal{H}_s$, compute its upper bound and lower bound $R_E^{up}(o_i)$ and $R_E^{low}(o_i)$;
7:    compute the lower bound for all the tuples in $\mathcal{H}_{\neg s}$ and all the unseen tuples;
8:    while less than $k$ tuples reported do
9:       let $o_i$ be the unreported tuple in $\mathcal{H}_s$ with smallest lower bound;
10:      if $R_E^{up}(o_i)$ is no larger than all the other lower bounds then
11:         report $o_i$ as the next tuple in the top-k answers;
12:      else
13:         break;
14:      end if
15:   end while
16: end while
```

The upper bounds and lower bounds in the third iteration are shown in Table 4.9. The tuple with smallest lower bound is o_3 such that $R_E^{low}(o_3) = 0.96$. Its upper bound ($R_E^{up}(o_3) = 1.08$) is larger than the lower bound of o_2 because $R_E^{low}(o_2) = 1.05$, so we continue for the next iteration.

In the fourth iteration, after retrieving $(o_4, 85)$ from relation S and $(o_6, 0.2)$ from relation P, we have $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4), o_4(85, -)\}$ and $\mathcal{H}_{\neg s} = \{o_5(-, 0.45), o_6(-, 0.2)\}$. $\bar{p} = 0.2$. We recompute the upper bounds and lower bounds R_E for the seen tuples, as shown in Table 4.9. The lower bound R_E value for the tuples in $\mathcal{H}_{\neg s}$ is $R_E^{low}(o_5) = 1.0575$, and the lower bound for the unseen tuples is

Tuple	Iteration 3		Iteration 4	
	$R_E^{up}(o_i)$	$R_E^{low}(o_i)$	$R_E^{up}(o_i)$	$R_E^{low}(o_i)$
o_1	1.05	1.05	1.05	1.05
o_2	-	1.05	-	1.3
o_3	1.08	0.96	1.04	0.96
o_4	-	-	-	1.38
o_5	-	1.0575	-	1.0575
o_6	-	-	-	1.42
$unseen$	-	1.17	-	1.38

Table 4.9: Upper/Lower Bounds in 3rd and 4th Iteration

$R_E^{low}(o) = 1.38$. The tuples with smallest lower bound is o_3, and its upper bound 1.04 is no larger than any other lower bounds. Therefore, o_3 can be determined to be the top-1 answer, although its exact $R_E(o_3)$ is still unknown. Then the unreported tuples with the smallest lower bound is o_1, and its upper bound 1.05 is no larger than any other unreported lower bounds. We can report o_1 as the top-2 answer. □

Compare Example 6 and Example 7. For random access on relation P, we report the top-2 answers after retrieving 5 tuples from relation S and 5 tuples from relation P. For sequential access on relation P, we can determine that tuples o_3 and o_1 must be in the top-2 answers, in the fourth iteration. It is does not only incur less expensive to conduct sequential access, but also retrieve less number of tuples compared to random access on relation P.

Theorem 4.4.2: *Algorithm 5 correctly computes the top-k tuples based on R_E, with sequential access on both relation S and relation P.* □

Proof Sketch: The correctness of Algorithm 5 directly follows from Lemma 4.3.1 and Lemma 4.3.2, and the correctness of the non-random access algorithms in [33, 80]. □

It is important to note that our sequential access is similar to the scenario discussed in [33, 80]. For each seen tuple, there is an upper bound and lower bound of

$R_E(o_i)$. If $R_E^{low}(o_i) \leq R_E^{up}(o_j)$, then tuple o_i is guaranteed to rank higher than o_j. For the unseen tuples, $R_E(o)$ is guaranteed to be less than or equal to some lower bound value. However, both work reported in [33, 80] are for deterministic datasets and cannot be directly applied to probabilistic query processing.

4.4.3. Sequential and Random Access

In Section 4.4.1 and Section 4.4.2, we discussed algorithms to find the top-k answers, with either random access or sequential access on relation P. We also discussed the advantages and disadvantages of these two access methods in Section 4.3.4. It is hard to get better bounds of pscore(o_i) for both the seen tuples and unseen tuples.

In this section, we discuss conducting both sequential and random access at the same time, to utilize both advantages of random access and sequential access. We can add random access of relation P into the framework of Algorithm 5, which is designed for sequential access only. In some iteration in Algorithm 5 (line 3), instead of sequentially retrieving the next tuple from relation P, we issue a random access to find the probability for the tuple in $\mathcal{H}_s$ with the largest score and unknown probability. Then, the bounds for all the tuples with smaller scores will become tighter. Note that, when a random access is issued, the $\bar{p}$ value will not be changed to the probability retrieved. Below, we give an example to show how random access helps bounding in sequential access.

Example 8: Consider the two relations, S and P, in Table 4.3. Let $k = 2$. Assume the probability of o_5 is changed to 0.5, i.e., $p(o_5) = 0.5$. $E[|I|] = 1.85$. After conducting four sequential accesses on relation S and relation P respectively, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, -), o_3(90, 0.4), o_4(85, -)\}$, $\mathcal{H}_{\neg s} = \{o_5(-, 0.5), o_6(-, 0.2)\}$, and $\bar{p} = 0.2$. Then the upper bounds and lower bounds are as follows. For tuple o_1, $R_E(o_1) = 1.085$. For tuple o_2, $R_E^{low}(o_2) = 1.34$. For tuple o_3, $R_E^{low}(o_3) = 0.99$, and $R_E^{up}(o_3) = 1.07$. For tuple o_4, $R_E^{low}(o_4) = 1.42$. For tuple o_5, $R_E^{low}(o_5) = 1.025$. For tuple o_6, $R_E^{low}(o_6) = 1.56$. For unseen tuple o, $R_E^{low}(o) = 1.42$. The upper bound of o_3, which is the tuple with smallest lower bound, is larger than the lower bound of o_5. Then no tuple can be determined to be in the top-k answers in this iteration.

If, in the fourth iteration, we issue a random access on P instead of sequential access to the probability of o_2. Then, $\mathcal{H}_s = \{o_1(100, 0.3), o_2(95, 0.15), o_3(90, 0.4), o_4(85, -)\}$, $\mathcal{H}_{\neg s} = \{o_5\ (-, 0.5)\}$, $\bar{p} = 0.3$. Note that $\bar{p}$ is larger than that of sequential access. The set of upper bounds and lower bounds are as follows. For tuple o_1, $R_E(o_1) = 1.085$. For tuple o_2, $R_E(o_2) = 1.49$. For tuple o_3, $R_E(o_3) = 1.05$. For tuple o_4, $R_E^{low}(o_4) = 1.25$. For tuple o_5, $R_E^{low}(o_5) = 1.1$. For unseen tuple o, $R_E^{low}(o) = 1.25$. Then, tuple o_3 can be determined with the highest rank, and tuple o_1 with the second highest rank. □

4.5. Performance Studies

We conducted extensive performance studies to get top-k answers using the two pscore functions, namely R_E and P_{HR}. We tested three algorithms, namely Random, Sequent, and Hybrid. All the three algorithms sequentially access relation S, and access relation P as the names imply. Random randomly accesses relation P, Sequent sequentially accesses relation P, and Hybrid may sequentially and randomly access relation P.

We use both real datasets and synthetic datasets. For the real datasets, we extracted several sets of x-tuples from the International Ice Patrol (IIP) Iceberg Sightings Database (`http://nsidc.org/data/g00807.html`) which is a database that collects the activities of the iceberg in the North Atlantic. The data are collected through airborne Coast Guard reconnaissance missions and information from radar and satellites to monitor iceberg danger near the Grand Banks of Newfoundland. There are some imprecise information for each record which is recorded as the confidence level according to the source of sighting. The 6 confidence levels are converted to confidence probabilities 0.8, 0.7, 0.6, 0.5, 0.4, and 0.3 respectively. Each drifting activity may be recorded several times by several types of sources. The x-tuples are the records that are obtained at the same time and the same location. We collect-

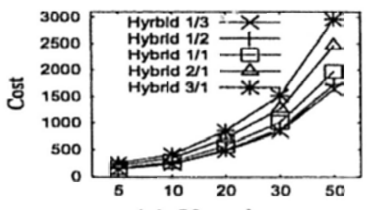
Parameter	**Range**	**Default**
k	5, 10, 20, 30, 50	20
$size$	$1, 2, 3, 4, 5$ ($\times 10k$)	3
$xsize$	$1, 2, 3, 4, 5$	3
$mean$	0.4, 0.5, 0.6, 0.7, 0.8	0.6

Table 4.10: Parameters for All Testings

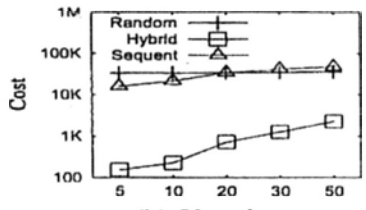
ed records from 1998 to 2007 and generated $17,505$ x-tuples which contain $50,879$ tuples. For each x-tuple τ, we normalize the probabilities for each tuple $o \in \tau$ as follows, $p(o) = \frac{conf(o)}{\sum_{o' \in \tau} conf(o')} \cdot max\{conf(o') | o' \in \tau\}$, where $conf(o)$ is the confidence probability for the tuple o. For each tuple extracted, we set its score to be the number of days drifted because it is important in determining the status of icebergs. We extracted 5 datasets from the whole dataset, which are sized $10,000$, $20,000$, $30,000$, $40,000$ and $50,000$ in terms of tuples respectively.

For the synthetic datasets, we have four types of distributions for the probabilities of x-tuples in the datasets, namely, uniform distribution, normal distribution, positive correlated distribution and negative correlated distribution. For the uniform distribution, given a mean value $0 < ave < 1$, suppose $d = min\{ave, 1 - ave\}$, all probabilities of x-tuples are distributed uniformly in the range $[ave - d, ave + d]$. For the normal distribution, for a mean value ave, all probabilities of x-tuples follow the normal distribution $N(\ ave, 0.2)$. For the positive and negative correlated distribution, the probabilities and the scores form a correlated bivariate with correlation 0.8 and -0.8 respectively.

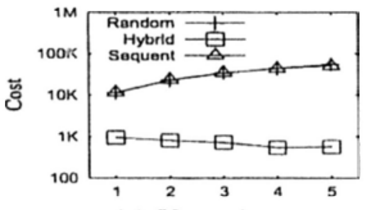
The parameters used and their default values for both real and synthetic data are given in Tab. 4.10. k is for the top-k value in a top-k probabilistic query. $size$ is the number of units for the dataset, where each unit contains $10,000$ tuples. $xsize$ is the average number of tuples in each x-tuple in the dataset. $mean$ is the mean value under a certain distribution discussed above. The k and $size$ parameters are used for both real and synthetic datasets, whereas $xsize$ and $mean$ parameters are used for the



(a) Vary k

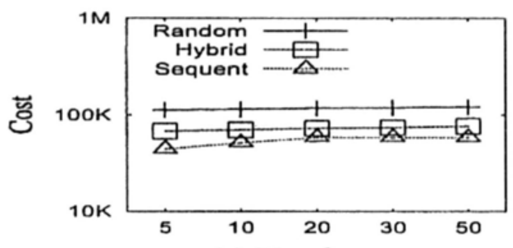


(b) Vary k

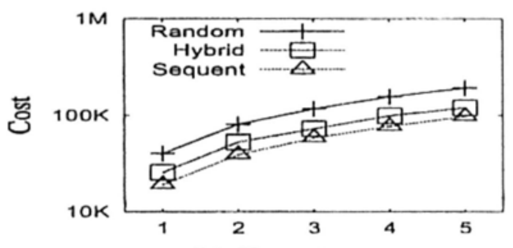


(c) Vary $size$

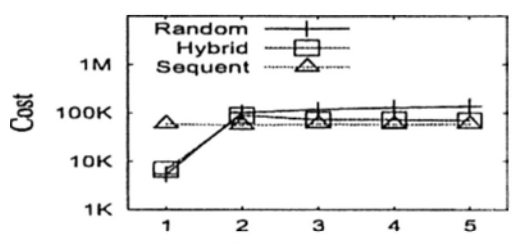
Figure 4.2: Real Data



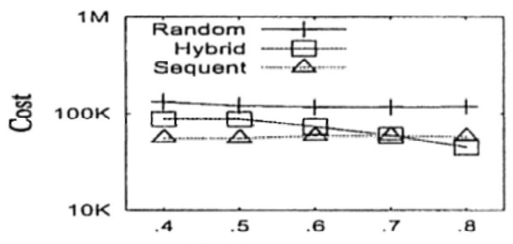
(a) Vary k



(b) Vary $size$



(c) Vary $xsize$



(d) Vary $mean$

Figure 4.3: Uniform Distribution

synthetic datasets only. When varying a certain parameter, all the other parameters are fixed to their default values. We report the I/O cost follow the same approach given in [83]. The cost is measured in unit, one sequential I/O contributes one unit, and one random I/O contributes 5 units. All algorithms are implemented in Visual C++, and all tests were conducted on a 2.8GHz CPU, 2GB memory and 80GB disk space PC running Windows XP.

Exp-1 Real Datasets for R_E: The testing results for R_E using the real datasets are shown in Fig. 4.2. In Fig. 4.2(a), we also test the Hybrid algorithm with different accessing patterns between the random access and the sequential access. In our testing, `Hybrid i/j` means that the Hybrid algorithm performs i sequential accesses fol-

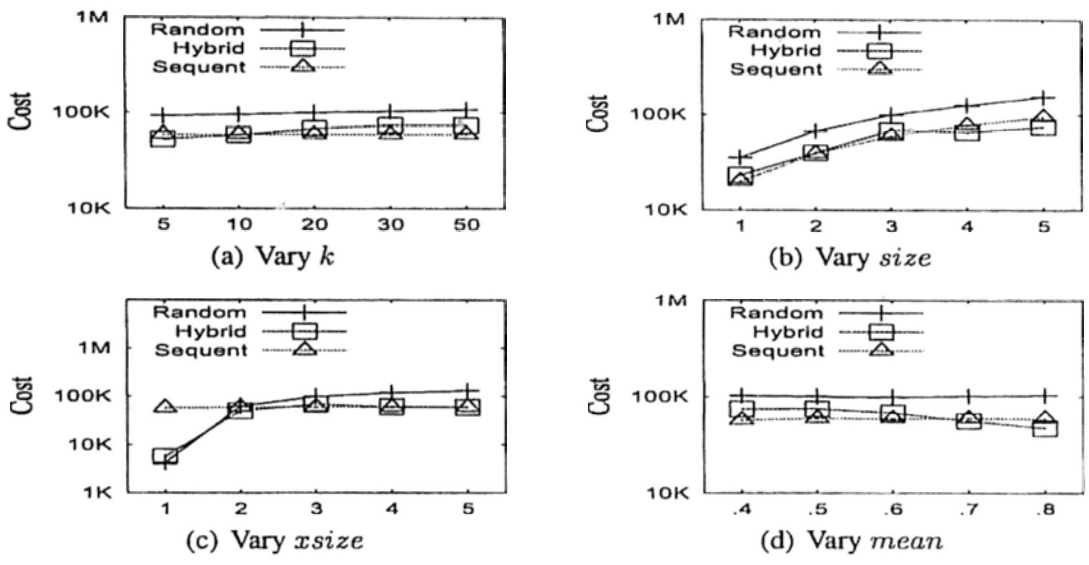
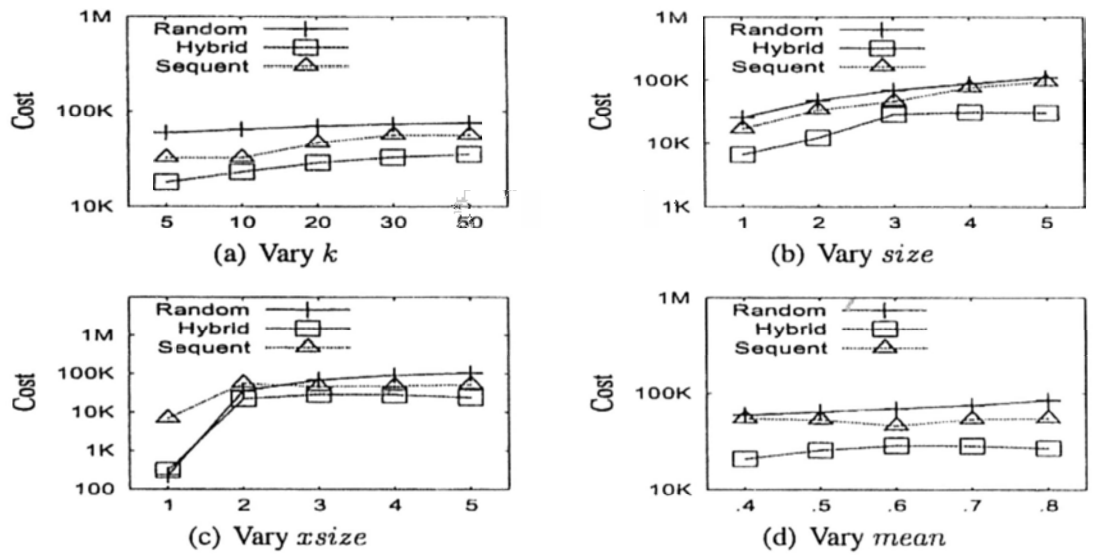


Figure 4.4: Normal Distribution

lowed by j random accesses alternatively. From Fig. 4.2(a), `Hybrid i/j` performs well when $i < j$, and there is no much difference between different Hybrid algorithms with $i < j$ regarding the cost. Because `Hybrid 1/1` performs well among these variants, we use `Hybrid 1/1` as Hybrid to conduct our testing below. In Fig. 4.2(b), when k increases, the numbers of tuples visited for all the 3 algorithms increase because the top-k answers for all the 3 algorithms are incrementally generated. Random and Sequent have similar costs. Hybrid is much better than both Random and Sequent. One of the bottlenecks for Sequent is that, although the lower bounds of R_E for the unseen tuples increase in every iteration, the set of seen tuples with both scores and probabilities is small. Note that the seen tuples with both scores and probabilities have an upper bound, and thus can satisfy the stop conditions. When random access is integrated into sequential access, the number of seen tuples with both scores and probabilities increases. Thus Hybrid can stop in an early stage. In Fig. 4.2(c), when the number of tuples in the datasets increases, the cost for both Random and Sequent increases, but it decreases for Hybrid. The reason is that, in the same iteration the upper bound of R_E for the seen tuples with both scores and probabilities tend to be smaller in a dataset with a larger size, where $p(o_i)$ tends to be larger. Hybrid is also



(a) Vary k (b) Vary $size$

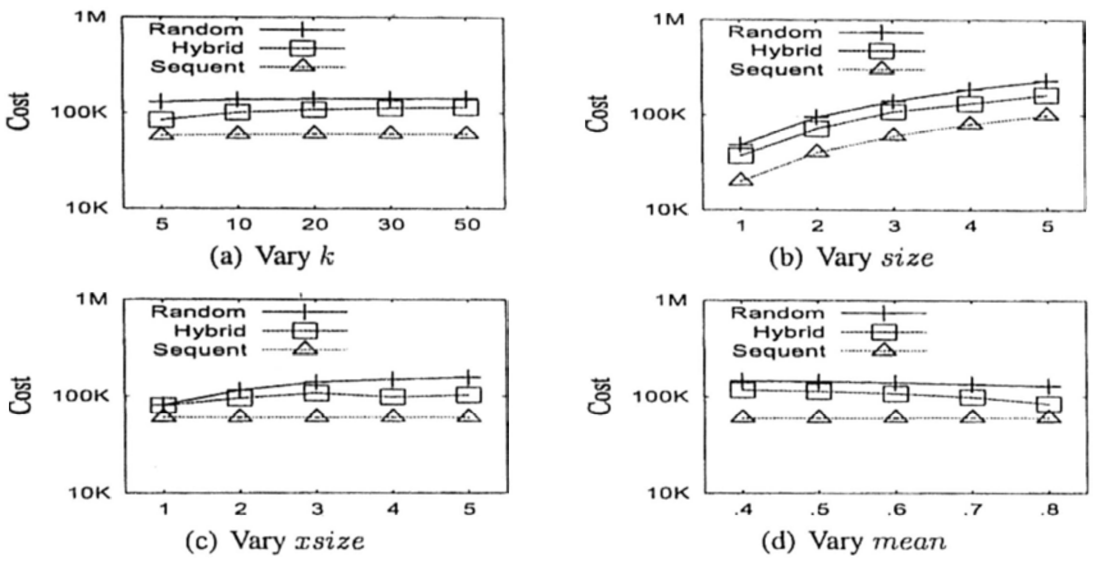
(c) Vary $xsize$ (d) Vary $mean$

Figure 4.5: Positive Correlated

much better than both Random and Sequent.

Exp-2 Synthetic Datasets for R_E: We tested all the three algorithms for R_E using synthetic data. For each of the four probability distributions, namely, uniform, normal, positive correlated, and negative correlated, we vary k, $size$, $xsize$, and $mean$, to test the performance for each algorithm. The results are shown in Fig. 4.3, Fig. 4.4, Fig. 4.5 and Fig. 4.6 respectively.

Under the uniform distribution, Fig. 4.3(a) shows that when k increases, the cost for all the 3 algorithms increases, Hybrid does not perform as good as in the real dataset, because in the real dataset, some of the first several tuples tend to have high probabilities, which make the upper bounds of R_E for those tuples small, and thus output in early iterations. In Fig. 4.3(b), when the number of tuples in the dataset increases, the cost for all the three algorithms increases, Hybrid is between Sequent and Random. Fig. 4.3(c) shows that when $xsize$ increases, the cost for all the 3 algorithms increases. Sequent performs badly when $xsize$ is small, because the average probability for each tuple in each x-tuple is large. In this situation, $\bar{p}$ will be large in each iteration, thus the lower bound for unseen tuples will be loose. In addition, when the average probability for tuples is large, the lower bound for R_E in Sequent is small.



(a) Vary k (b) Vary $size$

(c) Vary $xsize$ (d) Vary $mean$

Figure 4.6: Negative Correlated

Thus the performance is bad. On the other hand, the lower bound for the unseen tuples in Random increases faster, which makes it perform good. In Fig. 4.3(d), when the $mean$ value increases, the cost for all the 3 algorithms decreases, Hybrid algorithm decreases faster. As shown in Fig. 4.4, The algorithms under the normal distribution perform similar as in the uniform distribution, Sequent does not perform well in the normal distribution, because there are not many tuples with very high probabilities or very low probabilities.

Fig. 4.5 and Fig. 4.6 show the distributions in two opposite situations, positive correlated and negative correlated. In the positive correlated data, the tuples with large scores tend to have high probabilities, whereas, in the negative correlated data, the tuples with large scores tend to have low probabilities. The curves are all similar to those in the normal distributions. The performance for all testings in the positive correlated data is much better than those in the negative correlated data. In the positive correlated data, the lower bound of R_E in Random increases fast in the first several iterations, because the first several tuples tend to have a large probability. For the same reason, the upper bound of R_E for each tuple in Sequent decreases fast in the first several iterations. For Sequent, the number of tuples with both scores and probabilities

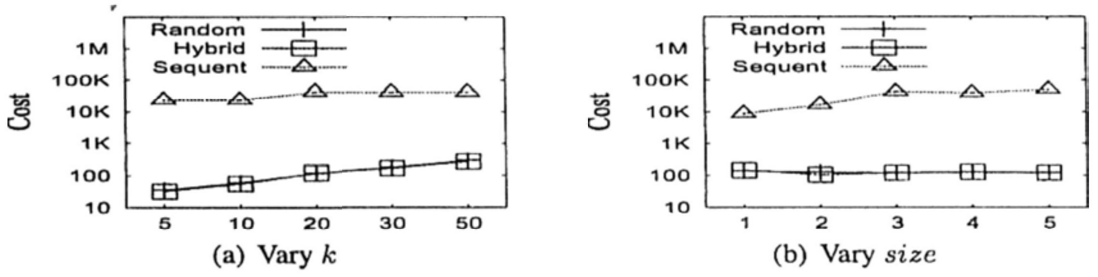


Figure 4.7: P_{HR} Testing

in the positive correlated data is much larger than that in the negative correlated data. It makes it faster in the positive correlated data.

Exp-3 Real Datasets for P_{HR}: We tested the P_{HR} function in the real dataset for all the 3 algorithms. The results are shown in Fig. 4.7. When either k or $size$ increases, the cost for all the 3 algorithms increases. Random and Hybrid have similar performance, and is much better than Sequent. This is because, the upper bound of P_{HR} in the Random algorithm decreases fast. For example, even if all of the first 30 tuples have very low probability, say 0.1, the upper bound for the unseen tuples after 30 iterations becomes $(1-0.1)^{30} = 0.04$ which is very small. This means that we can output the top-k answers in an early stage.

CHAPTER 5

QUERY RANKING IN PROBABILISTIC XML DATA

5.1. Introduction

Twig queries over traditional XML documents have been extensively studied [15]. The result of a twig query over an XML tree is a set of subtrees. In order to rank twig query results, some IR strategies are incorporated with other factors to score each subtree [34, 79, 5]. There are some works conducting keyword search over an ordinary XML tree [26, 40], and reporting the top-k results. In [14, 56], the authors treat XML trees as XML graphs, and assign weights to the nodes and edges of XML graphs, where the weight of a node indicates its importance and the weight of an edge represents the strength of its semantic connection in the XML tree. All these works are dealing with traditional (or ordinary) XML trees.

Probabilistic XML (PXML) have been extensively studied recently as discussed in Chapter. 2.4. The issues studied widely cover the PXML models [70, 46, 47], semantics, data integration [90], constraints [25], expressiveness [2, 82], query evaluation, query tractability [55], and complexity analysis. In this chapter, we study a new research issue, and we study PXML rank query, (Q, k), which is to rank top-k probabilities of the answers of a twig query Q in PXML data. The new challenging is how to compute top-k probabilities of answers of a twig query Q in PXML in the presence of containment (ancestor/descendant) relationships, where an answer of a twig query can be judged using any score function as studied in [34, 79, 5]. In the presence of the ancestor/descendant relationships, the existing dynamic programming approaches [92, 93, 44, 45] to compute top-k probabilities over a set of tuples cannot be directly applied, because in the context of PXML any node/edge may possibly have impacts on the top-k probabilities of answers. To the best of our knowledge, it is the first work which studies ranking of twig query results in the context of PXML.

We study three types of PXML-RANK queries, (Q, k), where Q can be a node query ($/\!/A$), a path query ($/\!/A/\!/B$), or a tree query ($/\!/A[./\!/C]/\!/B$). In our study, we consider all the three issues, namely, ranking, probability, and structures. The main contributions of this chapter are summarized below. First, we focus on node queries, and propose a new dynamic programming algorithm which can compute top-k probabilities for the answers of node queries based on the previously computed results in PXML data. Our algorithm considers the containment issue (ancestor/descendant) as well as the top-k probability and the score ranking (score functions) issues. We further propose optimization techniques to share the computational cost. Second, we show that our techniques can be used to support any path queries, and certain tree queries efficiently without enumerating all the possible worlds. We give conditions on the tree queries, and discuss our approaches. Third, we conduct extensive performance studies using both real and large benchmark datasets, and confirm the efficiency of our algorithms.

The remainder of this chapter is organized as follows. Section 5.2 reviews the definition of probabilistic XML, and gives our problem statement. In Section 5.3, we discuss the technique details of answering a PXML-RANK node query, and in Section 5.4, we discuss how to extend the algorithms of node query to process all path queries and certain tree queries. Experimental studies are given in Section 5.5.

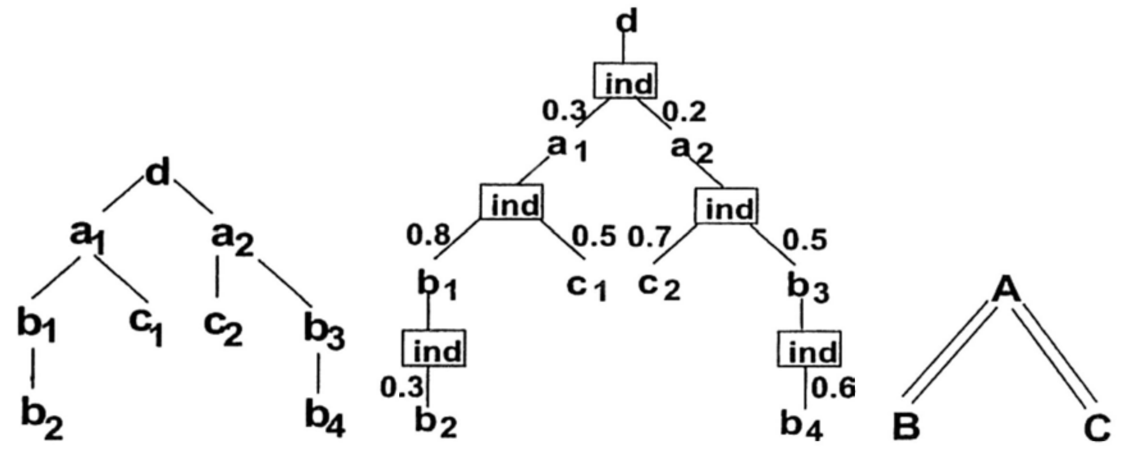
5.2. PXML and PXML-RANK

An XML document can be modeled as a rooted, unordered, and node-labeled tree, $T_X(V_X, E_X)$, where V_X represents a set of XML elements (nodes), and E_X represents a set of parent/child relationships (edges) between elements in XML. In an XML tree, a node is associated with a value x_i which belongs to a type (tag-name) X, denoted as $x_i \in X$. An XML tree is weighted if nodes and edges in the XML tree, $T_X(V_X, E_X)$, are associated with non-negative weights, denoted as $w_v(v)$ for $v \in V_X$ and $w_e(e)$ for $e \in E_X$, respectively. In the following, an XML tree is a weighted XML tree unless otherwise specified.

A probabilistic XML (or PXML for short) defines a probability distribution over XML trees. Following the model given in [70], which is the PrXML$^{\{\text{ind,mux}\}}$ model in [55], in this work, we define a PXML, $T_P(V_P, E_P)$, over a weighted XML tree $T_X(V_X, E_X)$. Here, V_P is a set of nodes $V_P = V_X \cup V_D$, where V_X is a set of ordinary nodes that appear in an XML tree, and V_D is a set of distribution nodes (e.g. independent, mutually exclusive). Consider a node u, which has a set of child nodes, V_u, in an XML tree T_X. In PXML, T_P, the ordinary node, u, may have several distribution nodes, as its child nodes, which specify the probability distributions over the disjoint subsets of the children of u, V_u. And E_P is a set of edges $E_P = E_{XX} \cup E_{XD} \cup E_{DD} \cup E_{DX}$ where E_{XX} is a set of edges that appear in E_X, E_{XD} is a set of edges from V_X nodes to V_D nodes, E_{DD} is a set of edges from V_D nodes to V_D nodes, and E_{DX} is a set of edges from V_D nodes to V_X nodes. Below, we call an E_{XX} edge an ordinary edge, and an edge in $E_{XD} \cup E_{DD} \cup E_{DX}$ a distribution edge. A positive probability is only associated with an edge, $e \in E_{DD} \cup E_{DX}$, denoted as $\rho_e(e)$. Note that a node in V_X has a node-weight, and an edge in $E_{XX} \cup E_{DX}$ is associated with an edge weight.[1]

Example 1: An XML tree, T'_X, is shown in Fig. 5.1(a). There is a D-typed node d, two A-typed nodes (a_1 and a_2), four B-typed nodes (b_i for $1 \leq i \leq 4$), and two C-typed nodes (c_1 and c_2). A PXML tree, T'_P, based on the XML tree T'_X, is shown in

[1]For simplicity, we assumed that default weights are zero.



(a) XML (b) PXML (c) Twig Query

Figure 5.1: XML, PXML, and Twig Query

Fig. 5.1(b). In T'_P, d has an independent distribution node as its child, which specifies that its two child nodes, a_1 and a_2 are independent. The probabilities of having a_1 and a_2 are 0.3 and 0.2, as indicated in the incoming edges to a_1 and a_2, respectively. In a similar fashion, there are other four independent distribution nodes. A node-weight, say $w_v(d)$, in T'_X can be specified as the node-weight associated with $w_v(d)$ in T'_P, and an edge-weight, say $w_e(d, a_1)$ can be specified in the incoming edge to a_1 in T'_P. □

A PXML tree, T_P, is a compact representation of probability distribution over a collection of XML trees, $T_{X_1}, T_{X_2}, \cdots$, which is generated in two steps.

First, we traverse the PXML tree, T_P, in a top-down fashion. When we visit an independent distribution node, ind_i, which has l children, we divide T_P into 2^l subtrees where each of them has a subset of the l children. When we visit a mutually exclusive distribution node, mux_i, which has l children, we divide T_P into l subtrees where each of them has one child. We repeat the same procedure for each of the divided subtrees recursively, and obtain the set of PXML subtrees, where every connected PXML subtree shares the same root node of the PXML tree. Let T'_P be one PXML subtree. The probability of T'_P, denoted as $\Pr(T'_P)$, is computed in Eq. (5.1).

$$\Pr(T'_P) = \textstyle\prod_{u \in V'_P} \Pr(u) \tag{5.1}$$

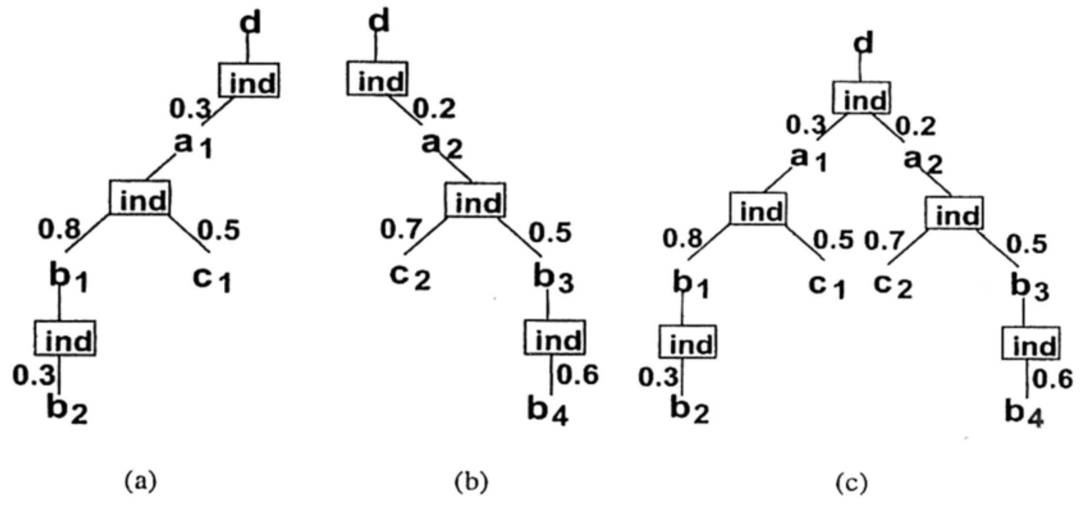


Figure 5.2: PXML Subtrees

Here, if u is an ordinary node, $\Pr(u) = 1$. If u is a distribution node, $\Pr(u)$ is computed as follows. Let u be a mutually exclusive node, and suppose u has l children. There are only two cases, selecting one of l children or none because it is mutually exclusive. For the former, $\Pr(u)$ is the probability associated with its outgoing edge to the selected child node. For the latter, it is one minus the summation of all the l existence probabilities. Let u be an independent node. Suppose u has n children, $u_1, u_2, \cdots u_n$, out of l children in total in the PXML tree T_P, $u_1, u_2, \cdots u_n, u_{n+1}, \cdots u_l$. $\Pr(u)$ is computed by multiplying the existence probability associated with u_i, for $1 \leq u_i \leq u_n$, and the absence probability (one minus the existence probability) for u_i for $u_n < u_i \leq u_l$.

Following the first step, for the PXML tree (Fig. 5.1(b)) 4 intermediate PXML subtrees will be generated when visiting the first independent distribution node under the root node. Three are shown in Fig. 5.2 in addition to the PXML subtree which contains the root node d only. Then, 6 PXML subtrees will be generated from the PXML subtrees Fig. 5.2(a)(b), respectively, and 36 PXML subtrees will be generated from the PXML subtree Fig. 5.2(c). In total, 49 PXML subtrees will be generated.

Second, for each of the PXML subtrees, T'_P, where $\Pr(T'_P) > 0$, we construct an XML tree, denoted as tree(T'_P), by removing all distribution nodes/edges and connecting two ordinary nodes if there are distribution nodes/edges in between. The entire set of such XML trees for a PXML tree is then uniquely identified. We denote it as $pwd(T_P) = \{T_{X_1}, T_{X_2}, \cdots\}$. The probability of T_{X_i} is given by

$$\Pr(T_{X_i}) = \sum_{\mathsf{tree}(T_{P_j})=T_{X_i}} \Pr(T_{P_j}) \tag{5.2}$$

because the same XML tree T_{X_i} can be constructed from several PXML subtrees, T_{P_j}. The set $pwd(T_P)$ forms the possible worlds of the probabilistic XML (PXML), T_P, and it satisfies the condition that $\sum_{T_{X_i} \in pwd(T_P)} \Pr(T_{X_i}) = 1$.

A twig query is a fragment of XPATH queries that can be represented as a query tree, $Q(V, E)$. Here, $V = (V_1, V_2, \cdots, V_n)$ is a set of nodes representing types (tag-names), and E is a set of edges. An edge between two typed nodes, for example, A and D, is either associated with an XPATH axis operator // or / to represent $A/\!/D$ or A/D. Given an XML tree T_X, the former is to retrieve all A and D typed nodes that satisfy the ancestor/descendant relationships, and the latter is to retrieve all A and D typed nodes that satisfy parent/child relationships. We call the former //-edge and the latter /-edge in short. As a special case, the root node in the query tree has an incoming //- or /-edge to represent an XPATH query, $/\!/A$ or $/A$, suppose the root node is A-typed. The answer of a n-node twig query, $Q(V, E)$, against an XML tree T_X, is a set of connected subtrees, where a connected subtree consists of n nodes $(v_1, v_2, \cdots, v_n)$ in T_X, for $v_i \in V_i$ $(1 \leq i \leq n)$, that satisfy all the structural relationships imposed by Q, and the minimal additional nodes/edges connecting the n nodes as a connected subtree. An example of an XPATH query is $Q = /\!/A[./\!/C]/\!/B$ (Fig. 5.1(c)). In this work, we consider three classes of twig queries: (1) node query, (2) path query, and (3) tree query. For example, $/\!/A$, $/\!/A/\!/B$, and $/\!/A[./\!/C]/\!/B$ are examples of node query, path query, and tree query, respectively.

A twig query, Q, against a PXML tree, T_P, can be processed by ignoring the existence of the distribution nodes/edges in T_P. The result is a set of XML trees, $M(Q, T_P) = \{\varphi_1, \varphi_2, \cdots, \varphi_N\}$. Let φ_i be an XML tree in the result for a twig query. The score of φ_i, denoted as $\omega(\varphi_i)$, can be computed using any score function as stud-

Algorithm 6 PXML-RANK (T_P, Q, k)

Input: a PXML tree T_P, a twig query Q, and an integer k.

Output: XML trees, $\varphi_1, \cdots, \varphi_M$, with top-$k$ probabilities s.t. $\rho(\varphi_1) \geq \cdots \geq \rho(\varphi_M)$.

```
1: $M \leftarrow$ twigQuery($Q$, $T_P$);
2: sort $M = \{\varphi_1, \cdots, \varphi_N\}$ in the non-increasing order of their scores;
3: $\mathcal{M} \leftarrow$ P-RANK ($T_P$, $k$, $M$);
4: remove all $\varphi_i$ from $\mathcal{M}$ if $\rho(\varphi_i) = 0$;
5: sort $\mathcal{M}$ in the non-increasing order of their top-$k$ probabilities ($\rho(\varphi_i)$);
6: return $\mathcal{M}$;
```

ied in [34, 79, 5]. For simplicity we define it as the total sum of its node/edge weights, i.e. $\omega(\varphi_i) = \sum_{u \in \varphi_i} w_v(u) + \sum_{e \in \varphi_i} w_e(e)$. The top-$k$ probability of φ_i, $\rho(\varphi_i)$, is given below.

$$\rho(\varphi_i) = \sum_{\substack{T_{X_j} \in pwd(T_P) \\ \varphi_i \in topk(T_{X_j})}} \Pr(T_{X_j}) \tag{5.3}$$

Here, T_{X_j} is one XML tree in the possible worlds of the PXML tree T_P ($pwd(T_P)$), and the probability of T_{X_j}, $\Pr(T_{X_j})$, is computed using Eq. (5.2). The probability of φ_i in the possible world, T_{X_j}, is $\Pr(T_{X_j})$ if φ_i is contained in T_{X_j} and the score of φ_i, $\omega(\varphi_i)$, is at least the k-th largest value in T_{X_j} ($\varphi_i \in topk(T_{X_j})$). It is important to note that several answers may appear in one possible world simultaneously. The $\rho(\varphi_i)$ is defined as the sum of such probability for every possible world where φ_i is contained.

Problem Statement [Top-k PXML Ranking (PXML-RANK)]: Let T_P be a PXML tree with possible worlds $pwd(T_P)$. A PXML-RANK query, (Q, k), is specified by a twig query, Q, and a positive number k, against T_P. It ranks the top-k probabilities for the answers, φ_i, that satisfy the twig query Q.

The algorithm for processing a PXML-RANK query, (Q, k), is outlined in Algorithm 6. First, it obtains a set of XML trees, $M = \{\varphi_1, \varphi_2, \cdots, \varphi_N\}$, that satisfy Q, against T_P (line 1). It can be done over an XML tree which virtually treats every distribution path between two ordinary nodes in T_P as an edge between the two ordinary nodes. Any efficient existing algorithms that process twig query can be adapted [75]. Second, it sorts M in the non-increasing order using the scores, such as φ_i appears before φ_j on the sorted M if $\omega(\varphi_i) \geq \omega(\varphi_j)$ (line 2). Third, it calls P-RANK to compute the top-k probabilities for all answers in M (line 3). P-RANK returns $\mathcal{M}$, which is a set of pairs $(\varphi_i, \rho(\varphi_i))$ for every answer φ_i in M. Finally, it removes all answers φ_i from $\mathcal{M}$ if their top-k probabilities are zero ($\rho(\varphi_i) = 0$) (line 4), and sorts $\mathcal{M}$ in the non-increasing order of their top-k probabilities ($\rho(\varphi_i)$) (line 5). Such $\mathcal{M}$ is returned in line 6. It is worth noting that P-RANK is a time-consuming task in computing PXML-RANK queries. Given a set of answers, $\{\varphi_1, \varphi_2, \cdots, \varphi_N\}$, a naive approach needs to compute $\rho(\varphi_i)$ by enumerating all the possible worlds, $pwd(T_P)$, using Eq. (5.3).

Below, we will first discuss how to process node queries (e.g. $/\!/A$), and then based on our techniques to process node queries we will discuss how to process any path queries (e.g. $/\!/A/\!/B$), and certain tree queries (e.g. $/\!/A[./\!/C]/\!/B$).

5.3. Node Query

In this section, we discuss processing PXML-RANK queries, (Q, k), where Q is a node query in the form of $/\!/A$. A node query is to find all A-typed nodes in PXML T_P to be ranked. Let the answer set M be $M = \{\varphi_1, \varphi_2, \cdots, \varphi_N\}$ which is processed by twigQuery(Q, T_P) in line 1 of Algorithm 6. Note that, here, an answer φ_i is an ordinary node in PXML tree T_P.

In the following, we first introduce some existing algorithms for processing ranking queries in a similar but different setting and discuss their deficiencies for processing PXML-RANK queries, followed by discussions of our new approaches.

5.3.1. New Containment Issues

In [45], Hua et al. discussed how to answer ranking queries in x-Relation uncertain data. In the x-Relation model, there is a set of independent x-tuples where an x-tuple consists of a set of mutually exclusive tuples (called alternatives). Each tuple in an x-tuple is associated with a score and a probability. A possible world is generated by choosing at most one tuple from each x-tuple. Under the x-Relation model, to process a ranking query, algorithms based on dynamic programming are proposed. The main issue is how to compute the probability that a tuple, t_i, to be the j-th largest ranked tuple in possible worlds, denoted as $p_{i,j}$. Assume all tuples are sorted in the decreasing order based on their scores, $\{t_1, t_2, \cdots, t_N\}$. The existing algorithms compute $p_{i,j}$, for $1 \leq i \leq N$ and $1 \leq j \leq k$, where k is the top-k value. First, consider every x-tuple has exactly one alternative, or equivalently, all the tuples are independent. The probability that t_i ranks j-th in a randomly generated possible world from the sorted tuple set $\{t_1, \cdots, t_i\}$ is $p_{i,j} = \Pr(t_i) \cdot r_{i-1,j-1}$. Here, $\Pr(t_i)$ is the existence probability of tuple t_i. $r_{i,j}$ is the probability that a randomly generated possible world from the tuple set $\{t_1, \cdots, t_i\}$ has exactly j tuples, and can be computed by the following dynamic programming equations.

$$r_{i,j} = \begin{cases} \Pr(t_i) \cdot r_{i-1,j-1} + (1 - \Pr(t_i)) \cdot r_{i-1,j} & \text{if } i \geq j \geq 0; \\ 1 & \text{if } i = j = 0; \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

With the above equations, all $r_{i,j}$ can be computed for $1 \leq i \leq N$ and $j = 0, 1, \cdots, k-1$, based on the previous values, namely, $r_{i-1,j-1}$ and $r_{i-1,j}$. When x-tuples represent multiple alternatives, the same dynamic programming equations can be applied with additional tuple transformations [45].

Like the x-Relation model, in our PXML model, we consider independent and/or mutually exclusive nodes, as well as the scores and top-k probabilities. Unlike the x-Relation model, we consider one additional criterion, containment. In other words, for a node query, an answer φ_i can be an ancestor/descendant of another answer φ_j in the PXML tree T_P. The additional criterion makes it difficult to apply the

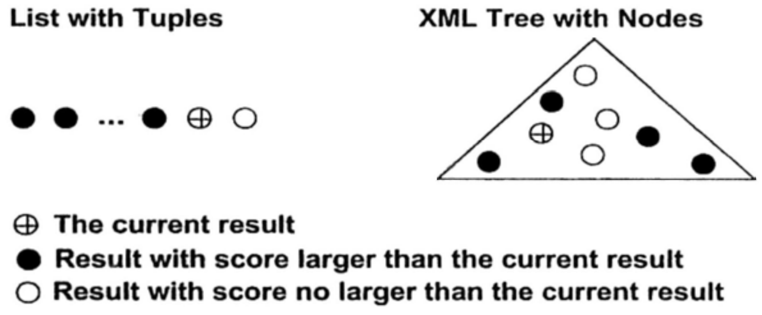

List with Tuples
XML Tree with Nodes
⊕ The current result
● Result with score larger than the current result
○ Result with score no larger than the current result

Figure 5.3: List v.s. XML Tree

existing techniques [93, 45, 94] to solve the problem in our setting, even for node queries. Fig. 5.3 illustrates the main differences. First, in an x-Relation model, the tuples, $\{t_1, t_2, \cdots, t_N\}$, themselves are the context in which the top-k probabilities are computed. The current $r_{i,j}$ for the sorted set of tuples $\{t_1, t_2, \cdots t_{i-1}, t_i\}$ can be computed by the previously computed $r_{i-1,j-1}$ and $r_{i-1,j}$ for the sorted set of tuples $\{t_1, t_2, \cdots t_{i-1}\}$. Note that all tuples are sorted based on their scores in a decreasing order. Every time for computing $r_{i,j}$ the algorithm only needs to consider an additional tuple t_i. The tuples $\{t_{i+1}, \cdots, t_N\}$, which have smaller scores than the current tuple t_i, are not needed in the x-Relation model, because they do not affect $r_{i,j}$ computing. However, in our problem setting, it becomes invalid that the nodes which have smaller scores than the current node are not relevant. As shown on the right side of Fig. 5.3, a node ("o") with a smaller score than the current node ("⊕") under consideration can be an ancestor or descendant of the current node. The existence/absence of every node may have impacts on the current node.

Remark 5.3.1: *The top-k probabilities for answers, $\{\varphi_1, \varphi_2, \cdots, \varphi_N\}$, need to be determined in the context of the entire* PXML *tree.* □

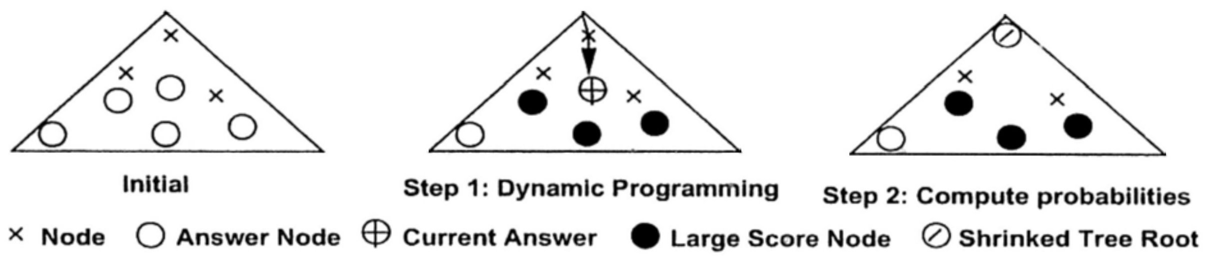


Figure 5.4: An Overview of Our Approach

5.3.2. An Overview of Our Approach

We outline our basic ideas for processing node queries in Fig. 5.4. We will discuss how to extend the basic ideas to process path queries and certain tree queries, and our optimization techniques later. Let the set of answers, M, be $M = \{\varphi_1, \varphi_2, \cdots, \varphi_N\}$ which is processed by twigQuery(Q, T_P) in Algorithm 6. All such answers in M are identified in the PXML tree T_P. It is shown in Fig. 5.4, in the initial stage, where tree nodes (not answers) and answers in T_P are indicated as "×" and "o", respectively. Then, we compute $p_{i,j}$ for every answer $\varphi_i \in M$, for $1 \leq j \leq k$. The answer φ_i to be computed at an iteration is called the current. For the current φ_i, we compute $p_{i,j}$ in two steps, computing a $r_{i,j}$-like variable (step 1) and computing $p_{i,j}$ (step 2).

In step 1, given the current answer φ_i (indicated as "⊕" in Fig. 5.4), the $r_{i,j}$-like variable we compute is $r^{\omega(\varphi_i)}_{\varphi_i,j}$. There exist main differences between $r_{i,j}$ and $r^{\omega(\varphi_i)}_{\varphi_i,j}$. Recall that $r_{i,j}$ is the probability that a randomly generated possible world from the sorted tuple set $\{t_1, \cdots, t_i\}$ has exactly j tuples. In Eq. (5.4), $r_{i,j}$ is computed for the current tuple t_i using the answers that have a larger score than t_i's by utilizing the sorted tuple set, $\{t_1, \cdots, t_{i-1}\}$, in the decreasing order of the scores. The value of i in $r_{i,j}$ means the position of the i-th tuple itself on the sorted tuple set. In our problem setting, there does not exist such a sorted set. In order to simulate the sorted set, in other words, the set of answers that have a larger score than the score of the current φ_i ($\omega(\varphi_i)$), the superscript of $r^{\omega(\varphi_i)}_{\varphi_i,j}$ is introduced. Also, because the sorted tuple set does not exist, the indicator of i used in $r_{i,j}$ for x-Relation model needs to be reconsidered.

In our model, instead of i, we use a subscript φ_i to indicate a subtree in PXML tree T_P rooted at node φ_i. The meaning of j in $r^{\omega(\varphi_i)}_{\varphi_i,j}$ remains unchanged. In summary, $r^{\omega(\varphi_i)}_{\varphi_i,j}$ is the probability that a randomly generated possible world from the subtree of the PXML T_P rooted at φ_i has exactly j answers whose score is larger than the score of φ_i, $\omega(\varphi_i)$.

In addition, there is a new issue on containment. For computing $r^{\omega(\varphi_i)}_{\varphi_i,j}$, we need compute all nodes $v \in T_P$ including the answers in M as well as the current node φ_i itself. For this purpose, we introduce a general variable $r^h_{v,j}$ where v is a node in the PXML tree T_P and h is a score. We compute $r^{\omega(\varphi_i)}_{v,j}$ for every node $v \in T_P$, based on the score $\omega(\varphi_i)$, using dynamic programming. It is important to note that $r^{\omega(\varphi_i)}_{v,j}$ can be computed based on the subtrees of the subtree rooted at v in T_P. Upon completion of the computation, $r^{\omega(\varphi_i)}_{v,j}$ are known for every node (including the current) and for $0 \leq j \leq k-1$. An answer $\varphi_l \in M$ is marked as "•" in Fig. 5.4, if it has a larger score than the current's ($\omega(\varphi_l) > \omega(\varphi_i)$).

In fact, up to this stage, $r^{\omega(\varphi_i)}_{\varphi_i,j}$ computed is local, since it is computed based on the subtree rooted at φ_i and is not computed in the entire PXML tree T_P globally. Note that there is a path from the root of PXML tree T_P to the current φ_i as indicated by "$\rightarrow\oplus$" in Fig. 5.4. The $r^{\omega(\varphi_i)}_{\varphi_i,j}$ needs to be computed globally under the condition that the path "$\rightarrow\oplus$" must exist. The condition of the existence of such a path "$\rightarrow\oplus$" may affect some other $r^h_{v,j}$ which in turn affect $r^{\omega(\varphi_i)}_{\varphi_i,j}$ for the current φ_i.

In step 2, based on the condition that the path "$\rightarrow\oplus$" must exist, we compute global $r^{\omega(\varphi_i)}_{\varphi_i,j}$ and $p_{i,j}$ for the current φ_i for $1 \leq j \leq k$. This is done by condensing the path "$\rightarrow\oplus$" into a node indicated as "$\oslash$" in Fig. 5.4. In other words, the PXML tree T_P is virtually transformed into another PXML tree $\mathcal{T}_P$ where the path "$\rightarrow\oplus$" in T_P becomes a node "$\oslash$" in $\mathcal{T}_P$ and all nodes that are connected to the nodes along the path "$\rightarrow\oplus$" in T_P are connected to the node "$\oslash$" in $\mathcal{T}_P$. It is worth noting that the global $r^{\omega(\varphi_i)}_{\varphi_i,j}$ and therefore $p_{i,j}$ can be computed using the same dynamic programming because the subtree rooted at φ_i is the entire PXML tree. Fig. 5.5 illustrates the main idea. The left tree is T_P where the path "$\rightarrow\oplus$" consists of an ordinary node (ord),

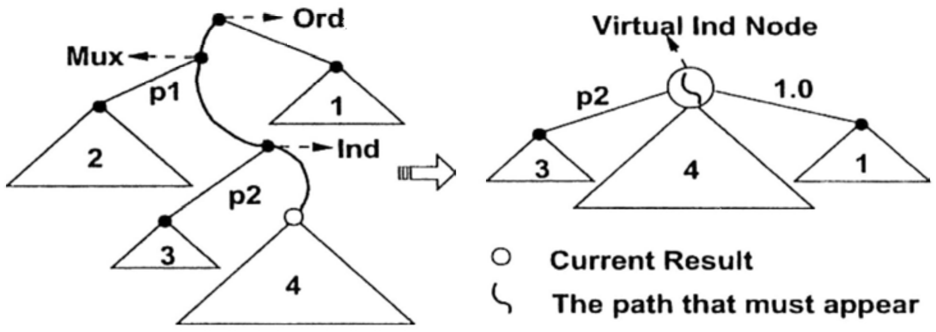


Figure 5.5: Computing $p_{i,j}$

a mutually exclusive node (mux), an independent node ind, and the current node φ_i (the root of the subtree (marked 4)). The right tree is $\mathcal{T}_P$. The subtree (marked 2) and its incoming edge are removed, because the mux node implies that the subtree (marked 2) cannot exist. The subtree (marked 3) in T_P is directly linked to the root node in $\mathcal{T}_P$ with the same probability. The ord node is treated as an independent node with probability one to the subtree (marked 1), which is connected to the root node in $\mathcal{T}_P$.

We also use $p_{i,j}$ to denote the probability that φ_i ranked j-th according to their scores in a randomly generated document. Finally, given $p_{i,j}$, $\rho(\varphi_i)$ can be computed using the following Eq. (5.5) instead of Eq. (5.3).

$$\rho(\varphi_i) = \textstyle\sum_{j=1}^{k} p_{i,j} \tag{5.5}$$

Below, we discuss how to compute $\rho(\varphi_i)$ in a conditional probability viewpoint. Suppose M' is the set of answers where every $\varphi_l \in M'$ has a larger score than φ_i such as $(\omega(\varphi_l) > \omega(\varphi_i))$. The probability of φ_i to appear in the top-k answers, $\rho(\varphi_i)$, can be formulated as follows.

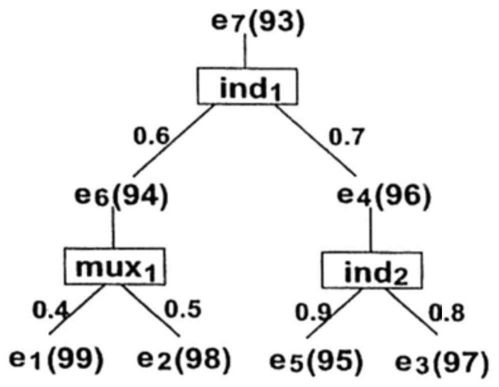
$$\begin{aligned} & \Pr(\varphi_i \text{ appears in the top-}k\text{ answers}) \\ = \ & \Pr(\varphi_i \text{ appears and at most } k\text{-1 answers in } M') \\ = \ & \Pr(\varphi_i \text{ appears}) \times \Pr(\text{at most } k\text{-1 answers in } M' \mid \varphi_i \text{ appears}) \end{aligned}$$

In [25] Cohen et al studied probabilistic XML with constraints (constraint satisfaction, query evaluation, and sampling), and the computation of $\Pr(\text{at most } k\text{-1 answers in } M' \mid \varphi_i \text{ appears})$ can be transformed to a constraint satisfaction problem. The constraint satisfaction problem can be specified by modifying the PXML T_P as follows: along the path from the root to φ_i, for each edge (u, v), (i) if u is a distribution node, then change the probability $\rho_e(u, v)$ to one, (ii) if u is a mutually exclusive node, then remove other children and the corresponding subtrees. Let the modified PXML be T'_P. Then, $\Pr(\text{at most } k\text{-1 answers in } M' \mid \varphi_i \text{ appears})$ is equal to the probability that a random generated XML tree from T'_P satisfies the constraints that it contains at most k-1 answers in M'. Cohen et al. show that the constraint satisfaction problem is polynomial time solvable, and propose an algorithm to solve it. In this work, we compute $\rho(\varphi_i)$, for $1 \leq i \leq N$ for the following main reasons. Although the constraint satisfaction problem is polynomial time solvable, it is proposed for general constraints, and is still time-consuming. For a different φ_i, there is a different T'_P, and the algorithm [25] needs to compute $\rho(\varphi_i)$ individually. Instead we mainly consider how to share the costs of computing different $\rho(\varphi_i)$'s using specific constraints as discussed above.

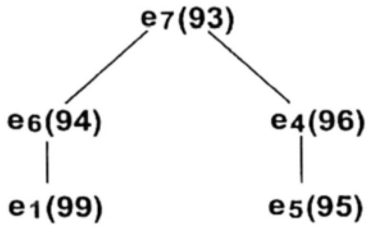
5.3.3. An Example

In this section, from a different viewpoint (conditional probability viewpoint), we explain how to compute $\rho(\varphi_i)$ using an example. Suppose φ_i is the current answer and M' is the set of answers where every $\varphi_l \in M'$ has a larger score than φ_i such as $(\omega(\varphi_l) > \omega(\varphi_i))$. The probability of φ_i to appear in the top-k answers can be formulated as follows.

$$\begin{aligned}
&\Pr(\varphi_i \text{ appears in the top-}k \text{ answers}) \\
=\;& \Pr(\varphi_i \text{ appears and at most } k-1 \text{ answers in } M') \\
=\;& \textstyle\sum_{j=0}^{k-1} \Pr(\varphi_i \text{ appears and exact } j \text{ answers in } M') \\
=\;& \textstyle\sum_{j=0}^{k-1} \Pr(\varphi_i \text{ appears}) \times \Pr(\text{exact } j \text{ answers in } M' \mid \varphi_i \text{ appears}).
\end{aligned}$$



(a) A PXML T'_P



(b) A random document T'_X

Figure 5.6: An Example

Here, $\Pr(\varphi_i \text{ appears})$ can be easily computed by multiplying all probabilities, $\rho_e(.)$, along the path from root node of the PXML tree T_P to φ_i ("$\rightarrow\oplus$"). The conditional probability of $\Pr(\text{exact } j \text{ answers in } M' \mid \varphi_i \text{ appears})$ is computed upon the condensed new PXML tree $\mathcal{T}_P$.

Fig. 5.6(a) shows a PXML T'_P that specifies the relationships among E-products (E-typed). There are 7 E-products e_i for $1 \leq i \leq 7$. An E-product has a score (indicated in the brackets) as its performance. There are some uncertainties. The distribution node ind_1 implies that e_7 is a part of e_6 with probability 0.6 and is a part of e_4 with probability 0.7. The two are independent. The distribution node mux_1 implies that either e_6 is used in e_1 with probability 0.4 or is used in e_2 with probability 0.5, but cannot be used in both. The two are mutually exclusive to each other. Suppose a PXML-RANK query (Q, k) is issued against T'_P (Fig. 5.6(a)), where $Q = /\!/E$ and $k = 1$. The set of E-products to be ranked is $M = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ which is computed using twigQuery(Q, T'_P) in Algorithm 6. Note that M is sorted in the decreasing order of the scores (performance indicators). Next, all E-products in M will be ranked based on top-k probabilities, $\rho(e_i)$, for $1 \leq i \leq 7$, against the possible words $pwd(T'_P)$.

One of the possible worlds (e.g. XML tree T'_X) is shown in Fig. 5.6(b). T'_X is with the conditions that e_6 and e_4 coexist under the independent node ind_1, e_5 is present alone under the independent node ind_2, and e_1 is present under the mutually exclusive node mux_1. The probability of T'_X is $\Pr(T'_X) = (0.4 \times 0.6) \times ((0.9 \times (1 - 0.8)) \times 0.7) = 0.03024$ where 0.4 is the probability of the subtree rooted at e_6, and $(0.9 \times (1 - 0.8))$ is the probability of the subtree rooted at e_4 in T'_X, respectively. It is infeasible to compute $\rho(e_i)$ using Eq. (5.3) because it needs to enumerate all possible worlds $pwd(T'_P)$ and summarize the top-k probabilities for e_i to be ranked top. Note that the size of possible worlds can be exponential. Instead we efficiently compute $\rho(e_i)$ using Eq. (5.5).

There are several obvious cases. (1) e_7 has the smallest score (93) in M. The only possible world for e_7 to be ranked top is the XML tree with e_7 only. $\rho(e_7) = (1 - 0.6) \times (1 - 0.7) = 0\ 12$. (2) e_1 has the largest score in M. If it appears in a possible world, it will be ranked top $\rho(e_1) = 0.6 \times 0.4 = 0.24$. (3) e_2 is ranked top if and only if e_1 does not appear in the possible worlds where e_2 appears. Note that e_1 and e_2 are mutually exclusive. In other words, if e_2 appears, then e_1 will not appear. $\rho(e_2) = 0.6 \times 0.5 = 0.3$. (4) e_5 can not be ranked top, because its ancestor e_4 has a higher score than e_5, and whenever e_5 appears e_4 will always appear. $\rho(e_5) = 0$.

Next consider a PXML-RANK query (Q, k) against T'_P (Fig. 5.6(a)) where Q is the same $/\!/E$ but $k = 3$. The set of answers to be ranked is the same $M = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$. We discuss computation of top-k probability, $\rho(e_6)$, for e_6 to be ranked top-k. Let e_6 to be the current with $\omega(e_6) = 94$. The E-products that have larger scores than $\omega(e_6) = 94$ are $M_h = \{e_1, e_2, e_3, e_4, e_5\}$ which we call h-answers.

Consider e_6. The PXML tree T'_P can be divided into several parts, P, T_1, and T_2, as shown in Fig. 5.7. Here, P represents "$\rightarrow\oplus$" which must appear because e_6 must appear. Obviously, Pr(e_6 appears) = Pr(P appears) = 0.6. We have $\rho(e_6) =$ Pr(P appears and at most 2 h-answers appear) $= \sum_{j=0}^{2}$ Pr(P appears and exact j h-answers appear) $= \sum_{j=0}^{2}$ Pr(P appears) $\times$ Pr(exact j h-answers appear $|$ P appears). Note that Pr(P appears) $= 0.6$. We explain how to compute

$$\Pr(\text{exact } j \text{ h-answers appear} \mid P \text{ appears}) \tag{5.6}$$

for $0 \leq j \leq 2$ below. Recall that, given the current e_6, in our notation, the probability

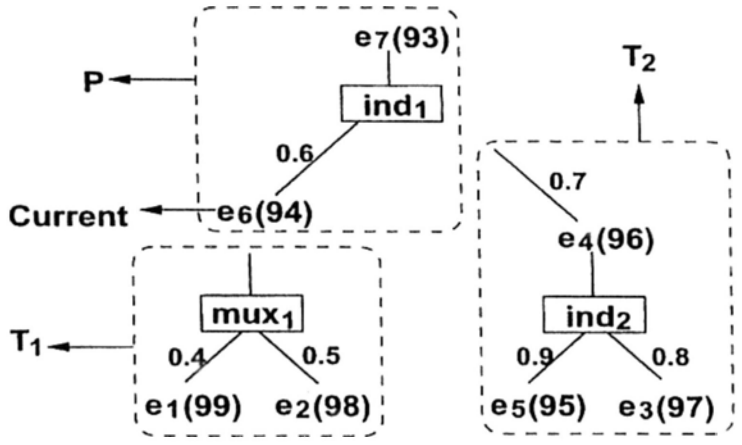


Figure 5.7: Cut the Tree into Several Parts

that exact j answers from M_h appear in the subtree rooted at v is denoted as $r^{\omega(e_6)}_{v,j} = r^{94}_{v,j}$. For simplicity, we use $r_{v,j}$ below.

Case-1 ($j = 0$): Eq. (5.6) equals to $\Pr(0$ h-answer in T_1 appears$) \times \Pr(0$ h-answer in T_2 appears$)$. For the first part, it is $r_{mux_1,0}$. The only situation that 0 h-answer appears in the subtree rooted at mux_1 (T_1) is that none of e_1 and e_2 appears. Since e_1 and e_2 are mutually exclusive, we have $r_{mux_1,0} = 1.0 - \rho_e(mux_1, e_1) - \rho_e(mux_1, e_2) = 1.0 - 0.4 - 0.5 = 0.1$. The second part is equal to $\Pr$(the absence of edge (ind_1, e_4))$+\Pr$(the existence of edge (ind_1, e_4)) $\times\, r_{e_4,0} = (1.0 - 0.7) + 0.7 \times r_{e_4,0}$. Here, $r_{e_4,0} = 0$ because e_4 must appear when considering the subtree rooted at e_4, so the second part is $0.3 + 0.7 \times 0 = 0.3$. Combining the two parts, Eq. (5.6), for $j = 0$, equals to $0.1 \times 0.3 = 0.03$.

Case-2 ($j = 1$): Eq. (5.6) equals to $\Pr(0$ h-answer in T_1 appears$) \times \Pr(1$ h-answers in T_2 appear$) + \Pr(1$ h-answers in T_1 appear$) \times \Pr(0$ h-answer in T_2 appears$)$. Note that $\Pr(0$ h-answer in T_1 appears$) = 0.1$ and $\Pr(0$ h-answer in T_2 appears$) = 0.3$ are computed in Case-1.

Here, $\Pr(1$ h-answer in T_1 appears$) = r_{mux_1,1}$. The only situation that 1 h-answer in the subtree rooted at mux_1 (T_1) appears is that either e_1 appears or e_2 appears. Since e_1 and e_2 are mutually exclusive, we have $\Pr(1$ h-answer in T_2 appears$) = r_{mux_1,1} =$

$\rho_e(mux_1, e_1) + \rho_e(mux_1, e_2) = 0.4 + 0.5 = 0.9$.

Also Pr(1 h-answer in T_2 appears) $=$ Pr(the existence of edge (ind_1, e_4)) $\times$ $r_{e_4,1} = 0.7 \times r_{e_4,1}$. The only situation that 1 h-answer appears in the subtree rooted at e_4 (T_2) is that 0 h-answer appears in the subtree rooted at ind_2, i.e., $r_{e_4,1} = r_{ind_2,0}$. It means that neither e_5 nor e_3 appears, i.e. $r_{ind_2,0} = (1 - \rho_e(ind_2, e_5)) \times (1 - \rho_e(ind_2, e_3)) = (1 - 0.9) \times (1 - 0.8) = 0.02$. Then, we have Pr(1 h-answer in T_2 appears) $= 0.7 \times 0.02 = 0.014$.

Therefore, Eq. (5.6), for $j = 1$, equals to $0.1 \times 0.014 + 0.9 \times 0.3 = 0.2714$.

Case-3 ($j = 2$): Eq. (5.6) equals to Pr(0 h-answer in T_1 appears) $\times$ Pr(2 h-answers in T_2 appear) $+$ Pr(1 h-answer in T_1 appears) $\times$ Pr(1 h-answer in T_2 appears) $+$ Pr(2 h-answers in T_1 appear) $\times$ Pr(0 h-answer in T_2 appears) $= 0.1 \times$ Pr(2 h-answers in T_2 appear) $+ 0.9 \times 0.014 +$ Pr(2 h-answers in T_1 appear) $\times 0.3$. The probabilities for $j < 2$ are computed already in Case-1 and Case-2.

Here, Pr(2 h-answers in T_1 appear) $= r_{mux_1,2}$. To have 2 h-answers appear in the subtree rooted at mux_1, both e_1 and e_2 must appear. This is impossible because e_1 and e_2 are mutually exclusive. We have Pr(2 h-answers in T_1 appear) $= r_{mux_1,2} = 0$.

On the other hand, Pr(2 h-answers in T_2 appear) $=$ Pr(existence of edge (ind_1, e_4)) $\times r_{e_4,2} = 0.7 \times r_{e_4,2}$. The only situation that 2 h-answers appear in the subtree rooted at e_4 is that 1 h-answer appears in the subtree rooted at ind_2, i.e., $r_{e_4,2} = r_{ind_2,1}$. It means that either (a) e_5 appears but e_3 does not appear or (b) e_5 does not appear but e_3 appears. We have $r_{ind_2,1} = \rho_e(ind_2, e_3) \times (1 - \rho_e(ind_2, e_5)) + (1 - \rho_e(ind_2, e_3)) \times \rho_e(ind_2, e_5) = 0.9 \times (1 - 0.8) + (1 - 0.9) \times 0.8 = 0.26$. We have Pr(2 h-answers in T_2 appear) $= 0.7 \times 0.26 = 0.182$. Therefore, Eq. (5.6), for $j = 2$, equals to $0.1 \times 0.182 + 0.9 \times 0.014 + 0 \times 0.3 = 0.0308$.

With all the three cases, $\rho(e_6) =$ Pr(P appears) $\times \sum_{j=0}^{2}$ Pr(exact j h-answers appear | P appears) $= 0.6 \times (0.03 + 0.2714 + 0.0308) = 0.19932$. For the PXML-RANK query $(/\!/E, 3)$ against T'_P (Fig. 5.6(a)), the ranking is shown below.

$\rho(e_4)$	$\rho(e_3)$	$\rho(e_7)$	$\rho(e_5)$	$\rho(e_2)$	$\rho(e_1)$	$\rho(e_6)$
0.7	0.56	0.37924	0.35784	0.3	0.24	0.19932

Algorithm 7 P-RANK (T_P, k, M)

Input: a PXML T_P, an integer k, and a sorted set of twig query answers $M = \{\varphi_1, \cdots, \varphi_N\}$, s.t. $\omega(\varphi_1) \geq \cdots \geq \omega(\varphi_N)$.

Output: $(\varphi_i, \rho(\varphi_i))$, for $1 \leq i \leq N$.

```
1: for $i \leftarrow 1$ to $N$ do
2:     H-PROB ($T_P$, $\varphi_i$, $k$);
3:     $P \leftarrow \mathsf{path}(T_P, \varphi_i)$;
4:     $\mathcal{T}_P \leftarrow$ PATH-CONDENSE ($T_P$, $\varphi_i$);
5:     $v' \leftarrow root(\mathcal{T}_P)$;
6:     H-TOPK ($v'$, $k$, $\omega(\varphi_i)$, children($v'$));
7:     $s \leftarrow$ count($P$);
8:     $p \leftarrow \prod_{e \in P} \rho_e(e)$;
9:     $p_{i,j} \leftarrow 0$, for $1 \leq j \leq s$;
10:    $p_{i,j} \leftarrow p \cdot r^{\omega(\varphi_i)}_{v',j-s-1}$, for $s+1 \leq j \leq k$;
11:    $\rho(\varphi_i) \leftarrow \sum_{j=1}^{k} p_{i,j}$;
12: end for
13: return $\{(\varphi_1, \rho(\varphi_1)), \cdots, (\varphi_N, \rho(\varphi_N))\}$;
```

5.3.4. Algorithms

The algorithm to compute P-RANK for node queries is given in Algorithm 7. It takes three inputs. The PXML tree T_P, the top-k value k, and a set of answers $M = \{\varphi_1, \varphi_2, \cdots, \varphi_N\}$ which is sorted in the decreasing order of their scores $\omega(\cdot)$. For each φ_i (the current), in a for-loop, it processes the following tasks. It computes its local $r^{\omega(\varphi_i)}_{\varphi_i,j}$ using dynamic programming (line 2). It identifies the path from the root of T_P to the current φ_i ("$\rightarrow\oplus$"), and assigns it to P (line 3). Then, it virtually reconstructs T_P to $\mathcal{T}_P$ by condensing the path P into a node which is the root of $\mathcal{T}_P$, v' (line 4-5). It computes the global $r^{\omega(\varphi_i)}_{\varphi_i,j}$ in line 6 using dynamic programming where children(v') indicates the children of the root node v'. In order to compute $p_{i,j}$ where i implies φ_i, it counts how many nodes on the path P ("$\rightarrow\oplus$") that are with a

Algorithm 8 H-PROB (T_P, φ, k)

Input: a PXML $T_P(V_P, E_P)$, an answer φ, and an integer k.

Output: $r_{v,j}^{\omega(\varphi)}$, for $v \in V_P$ and $0 \le j \le k-1$.

```
1: $h \leftarrow \omega(\varphi)$;
2: for every $v \in V_P$ in the post-order traversing order do
3:     if $v$ is a leaf node then
4:         $r_{v,j}^{h} \leftarrow 0$, for $0 \le j \le k-1$;
5:         if $v$ is an answer with $\omega(v) > h$ then
6:             $r_{v,1}^{h} \leftarrow 1$;
7:         else
8:             $r_{v,0}^{h} \leftarrow 1$;
9:         end if
10:    else
11:        let $\{v_1, \cdots, v_l\}$ be the set of children of $v$ in $T_P$;
12:        H-TOPK $(v, k, h, \{v_1, \cdots, v_l\})$;
13:    end if
14: end for
15: return $r_{v,j}^{\omega(\varphi)}$;
```

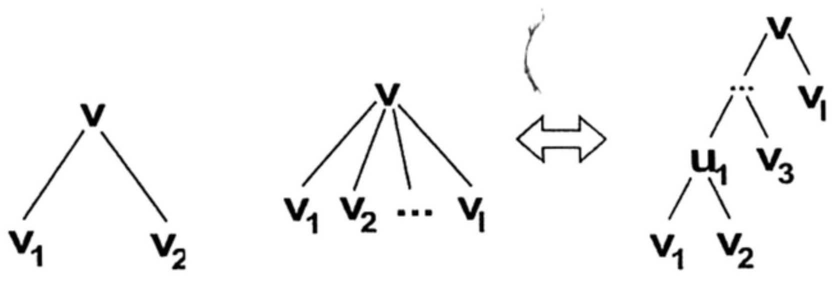
score greater than $\omega(\varphi_i)$ (line 7). The algorithm computes $p_{i,j}$ (line 8-10), and then computes $\rho(\varphi_i)$ in line 11.

We explain H-PROB (used in line 2, Algorithm 7). The H-PROB algorithm is given in Algorithm 8. It takes three inputs, the PXML tree T_P, the current answer φ, and the value of k. The main task of H-PROB is to compute local $r_{v,j}^{h}$ where $h = \omega(\varphi)$ in a bottom-up fashion. For non-leaf nodes, it further calls H-TOPK (Algorithm 9) to compute using dynamic programming. The H-TOPK algorithm takes four inputs to compute $r_{v,j}^{h}$, for $0 \le j < k$. Here, h is the score of the current φ, v is the non-leaf node in question, $\{v_1, \cdots, v_l\}$ are the children of v. It assumes that $r_{v_i,j}^{h}$, for $1 \le i \le l$, have already been computed. (Note that the H-PROB algorithm uses a bottom-up traversal to compute.) There are several cases handled in the H-TOPK algorithm: (1) v

is a mutually exclusive distribution node (line 1-3), (2) v is an independent distribution node (line 4-11), and (3) v is an ordinary node (line 12-20).

Algorithm 9 H-TOPK (v, k, h, $\{v_1, \cdots, v_l\}$)

```
1: if $v$ is mutually exclusive then
2:     $r^h_{v,j} \leftarrow \sum_{i=1}^{l} \rho_e(v, v_i) \cdot r^h_{v_i,j}$, for $0 \leq j \leq k-1$;
3:     $r^h_{v,0} \leftarrow r^h_{v,0} + (1 - \sum_{i=1}^{l} \rho_e(v, v_i))$;
4: end if
5: if $v$ is independent then
6:     for $i \leftarrow 1$ to $l$ do
7:         $B^h_{v_i,j} \leftarrow \rho_e(v, v_i) \cdot r^h_{v_i,j}$, for $0 \leq j \leq k-1$;
8:         $B^h_{v_i,0} \leftarrow B^h_{v_i,0} + (1 - \rho_e(v, v_i))$;
9:     end for
10:    $r^h_{u_0,j} \leftarrow B^h_{v_1,j}$, for $0 \leq j \leq k-1$;
11:    for $i \leftarrow 1$ to $l-1$ do
12:        $r^h_{u_i,j} \leftarrow \sum_{s=0}^{j} r^h_{u_{i-1},s} \cdot B^h_{v_{i+1},j-s}$, for $0 \leq j \leq k-1$;
13:    end for
14:    $r^h_{v,j} \leftarrow r^h_{u_{l-1},j}$, for $0 \leq j \leq k-1$;
15: end if
16: if $v$ is ordinary then
17:    $r^h_{u_0,j} \leftarrow r^h_{v_1,j}$, for $0 \leq j \leq k-1$;
18:    for $i \leftarrow 1$ to $l-1$ do
19:        $r^h_{u_i,j} \leftarrow \sum_{s=0}^{j} r^h_{u_{i-1},s} \cdot r^h_{v_{i+1},j-s}$, for $0 \leq j \leq k-1$;
20:    end for
21:    if $v$ is an answer with $\omega(v) > h$ then
22:        $r^h_{v,j} \leftarrow r^h_{u_{l-1},j-1}$, for $1 \leq j \leq k-1$;
23:        $r^h_{v,0} \leftarrow 0$;
24:    else
25:        $r^h_{v,j} \leftarrow r^h_{u_{l-1},j}$, for $0 \leq j \leq k-1$;
26:    end if
27: end if
28: return $r^h_{v,0}, \cdots, r^h_{v,k-1}$;
```



(a) 2 children (b) l children

Figure 5.8: Computing $r^h_{v,j}$

Below, we explain the case when v has 2 children ($l = 2$): $\{v_1, v_2\}$ (Fig. 5.8(a)). Note that both $r^h_{v_1,j}$ and $r^h_{v_2,j}$ have been computed.

If v is a mutually exclusive node, for $j > 0$, v_1 and v_2 can not appear simultaneously. $r^h_{v,j}$ can be computed in two cases, either with the subtree rooted at v_1 or with the subtree rooted at v_2. That is, $r^h_{v,j} = \rho_e(v, v_1) \cdot r^h_{v_1,j} + \rho_e(v, v_2) \cdot r^h_{v_2,j}$. For $j = 0$, it needs to consider an additional case that none of the two subtrees are selected. $r^h_{v,0}$ is computed as follows. $r^h_{v,0} = \rho_e(v, v_1) \cdot r^h_{v_1,0} + \rho_e(v, v_2) \cdot r^h_{v_2,0} + (1 - \rho_e(v, v_1) - \rho_e(v, v_2))$.

If v is an independent node, the existence of v_i is independent from each other where $i = 1, 2$. v chooses either v_1, or v_2, or both, or none. Let $B^h_{v_i,j}$ be the probability that a randomly generated XML tree, from the branch (v, v_i) which consists of the edge (v, v_i) and the subtree rooted at v_i, contains exactly j answers with score greater than h. When $j > 0$, $B^h_{v_i,j} = \rho_e(v, v_i) \cdot r^h_{v_i,j}$, and, when $j = 0$, $B^h_{v_i,0} = \rho_e(v, v_i) \cdot r^h_{v_i,0} + (1 - \rho_e(v, v_i))$. Then, $r^h_{v,j}$ includes the cases that, in a randomly generated XML tree, one branch, say (v, v_1), contains exactly s ($\leq j$) answers with a score greater than h, and the other branch, say (v, v_2), contains exactly $j - s$ answers with a score greater than h, and is computed as $r^h_{v,j} = \sum_{s=0}^{j} B^h_{v_1,s} \cdot B^h_{v_2,j-s}$, for $0 \leq j \leq k - 1$.

If v is an ordinary node, it can be computed by treating v as an independent node, i.e. v has two independent children v_1 and v_2, with probability $\rho_e(v, v_1) = 1$ and $\rho_e(v, v_2) = 1$, which means that both edges must exist with probability one. First compute $\tilde{r}^h_{v,j}$ by treating v as an independent node, i.e. $\tilde{r}^h_{v,j} = \sum_{i=0}^{j} r^h_{v_1,i} \cdot r^h_{v_2,j-i}$. Note that in this case $B^h_{v_i,s} = r^h_{v_i,s}$. If v itself is an answer with score greater than h, then

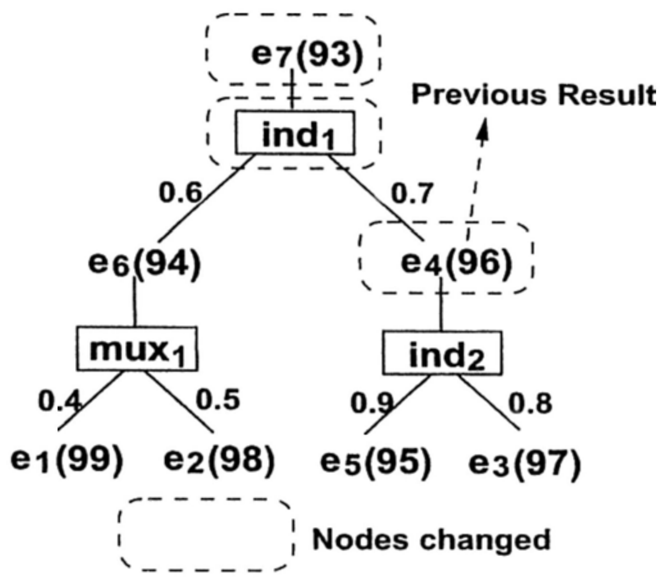


Figure 5.9: Computing H-PROB (T_P, φ_{i+1}, k)

$r^h_{v,j} = \bar{r}^h_{v,j-1}$ for $0 < j \leq k-1$ and $r^h_{v,0} = 0$, otherwise $r^h_{v,j} = \bar{r}^h_{v,j}$ for $0 \leq j \leq k-1$.

We design our algorithm for handling general l children of a given node v. If v is an independent node or an ordinary node, in order to compute $r^h_{v,j}$, we need to consider how many exact answers out of j answers are from which subtrees by enumerating all the sequences, $i_1, \cdots, i_l$, such that $\sum_{s=1}^{l} i_s = j$ for any fixed $j \in \{0, \cdots, k-1\}$. We handle l children of a given node v, if it is an independent/ordinary node, by transforming the node v with l children into a left-deep binary subtree, as shown in Fig. 5.8(b). There are $2l-1$ nodes in the transformed binary tree in total, $\{v_1, \cdots, v_l, u_1, \cdots, u_{l-2}, v\}$. Here, $v_1, \cdots, v_l$ are leaf nodes, u_1 has two children (v_1 and v_2), u_i, for $i > 1$, has two children (u_{i-1} and v_{i+1}), for $2 \leq i \leq l-2$. The node v has two children u_{l-2} and v_l. If v is an ordinary node, then $u_1, \cdots, u_{l-2}$ are ordinary nodes with weight 0. If v is an independent node, then $u_1, \cdots, u_{l-2}$ also are independent nodes, the probability $\rho_e(v, v_i)$ is specified on the incoming edge to v_i. All other edges have probability one. It can be verified that the transformed left-deep tree will give the same result. The H-TOPK algorithm (Algorithm 9) is designed using the left-deep binary tree, where v_1 is treated as u_0 and v is treated as u_{l-1}.

	$r^h_{v,j}$ for $h = \omega(e_4)$			$r^h_{v,j}$ for $h = \omega(e_5)$		
node (v)	$j = 0$	$j = 1$	$j = 2$	$j = 0$	$j = 1$	$j = 2$
e_1	0	1.0	0	0	1.0	0
e_2	0	1.0	0	0	1.0	0
e_3	0	1.0	0	0	1.0	0
e_4	**0.2**	**0.8**	**0**	**0**	**0.2**	**0.8**
e_5	1.0	0	0	1.0	0	0
e_6	0.1	0.9	0	0.1	0.9	0
e_7	**0.2024**	**0.4952**	**0.3024**	**0.138**	**0.2264**	**0.3332**
mux_1	0.1	0.9	0	0.1	0.9	0
ind_1	**0.2024**	**0.4952**	**0.3024**	**0.138**	**0.2264**	**0.3332**
ind_2	0.2	0.8	0	0.2	0.8	0

Table 5.1: Consecutive Computing $r^h_{v,j}$ for $h = \omega(e_4), \omega(e_5)$

Optimization-I: As indicated in the P-RANK algorithm, it needs to call the H-PROB algorithm for every answer φ_i in $M = \{\varphi_1, \varphi_2, \cdots, \varphi_N\}$, which is sorted in the decreasing order of the scores. The cost of computing using dynamic programming is costly. In fact, the cost can be shared between successive calls of H-PROB, e.g., H-PROB (T_P, φ_i, k) and H-PROB (T_P, φ_{i+1}, k). Consider the same PXML-RANK query $(/\!/E, 3)$ against the PXML tree T'_P (Fig. 5.6(a)). Table 5.1 shows the results of $r^h_{v,j}$ when computing the two consecutive answers, e_4 and e_5. It shows that most values when computing $r^h_{v,j}$ for e_5 remain unchanged, given $r^h_{v,j}$ computed for e_4. The possible change part is highlighted in the dot rectangles in Fig. 5.9, which is along the path from the root to the previous answer. A lemma is given below.

Lemma 5.3.1: *Let* H-PROB *(T_P, φ_i, k) and* H-PROB *(T_P, φ_{i+1}, k) be two consecutive executions, for two answers, φ_i and φ_{i+1} in the sorted answer set M. When $\omega(\varphi_i) \neq \omega(\varphi_{i+1})$, the values $r^{\omega(\varphi_i)}_{v,j}$ and $r^{\omega(\varphi_{i+1})}_{v,j}$ are identical for the nodes that are not on the path from the root of T_P to the node φ_i. When $\omega(\varphi_i) = \omega(\varphi_{i+1})$, for all nodes, $r^{\omega(\varphi_i)}_{v,j} = r^{\omega(\varphi_{i+1})}_{v,j}$.* □

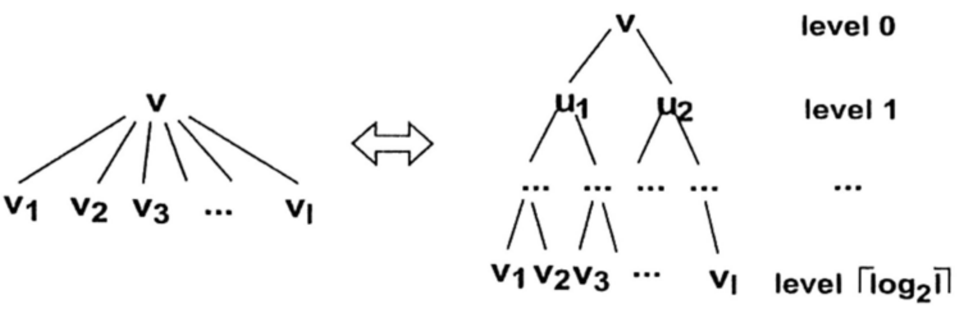


Figure 5.10: Convert a Node with l Children to a Complete Binary Tree

The P-RANK and H-PROB algorithms only need to be slightly changed to adapt the optimization-I.

Optimization-II: As stated in Optimization-I, for two successive H-PROB (T_P, φ_i, k) and H-PROB (T_P, φ_{i+1}, k), only the $r_{v,j}^{\omega(\varphi_i)}$ values for those nodes v on a certain path change. For each such a node v, suppose it has l children $v_1, v_2, \cdots, v_l$, if v is an independent node or an ordinary node, we have to spend $O(k^2 \cdot l)$ time to compute $r_{v,j}^h$ using H-TOPK. When l is large, the cost can be large. In the following, we show that we can reduce it to $O(k^2 \cdot \log(l))$. We construct a complete binary tree $\mathcal{B}$ with $d = \lceil \log_2 l \rceil + 1$ levels, where the root is at level 0 and the p-th level has 2^p nodes for $0 \leq p < d$. The root of $\mathcal{B}$ is v, the non-leaf nodes are independent nodes marked u_i, and the leaf nodes are $v_1, v_2, \cdots, v_l$ under nodes in level $d-1$ as shown in Fig. 5.10. The probability associated with the edge between v and v_i is reserved for the edges incoming v_i in $\mathcal{B}$. Other newly added edges have probability 1. It can be proved that $r_{v,j}^{\omega(\varphi_i)}$ on the new tree is equal to those on the original tree. Utilizing the complete binary tree, we only need $O(k^2 \cdot \log(l))$ to compute v without affecting other nodes because the depth of the new tree is $O(\log(l))$ and in the path from v to v_i, the degree of each node is at most 2.

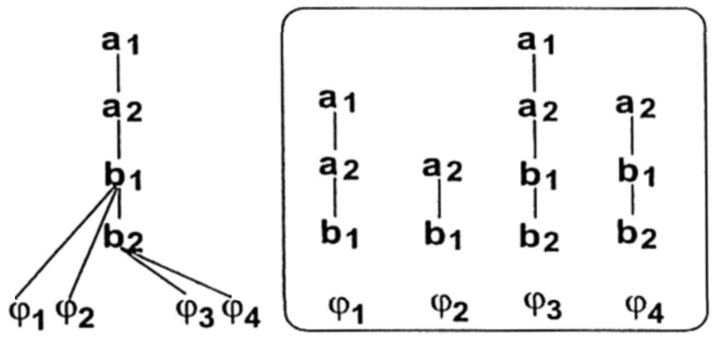


Figure 5.11: Handling Multiple Path Results in a Single Node

5.4. Path Query and Tree Query

In this section, we discuss other twig queries, namely, path queries, $/\!/A/\!/B$, and tree queries $/\!/A[./\!/C]/\!/B$. Our techniques can efficiently process any path queries. Consider a PXML-RANK query $(/\!/A/\!/B, k)$, against the PXML tree T_P (Fig. 5.1(b)). The answers are a set of paths, $M = \{a_1 \to b_1, a_1 \to b_1 \to b_2, a_2 \to b_3, a_2 \to b_3 \to b_4\}$. In our problem setting, the existence probability of a node is equal to the probability of the path from the root to the node, because the existence of a node depends on the existence of its ancestors. Therefore, we can compute the top-k probability for an answer of a path query, as to compute the last node of the answer. Taken $a_1 \to b_1 \to b_2$ as an example, we can compute its top-k probability as to compute the top-k probability for b_2. Our techniques can be used for processing any PXML-RANK queries (Q, k), where Q is a path query.

We further explain how to process when several answers that have the same lowest node, using an example. Suppose that there are four answers to be ranked, $M = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, where an answer φ_i is shown in the rectangle in Fig. 5.11 over the data path in PXML tree on the left side in Fig. 5.11. Here, the two answers, φ_1 and φ_2, share the same lowest node b_1, and the other two answers, φ_3 and φ_4, share the same lowest node b_2. When processing top-k probabilities for the four (path) answers, we virtually add four additional nodes to represent the four (path) answers as indicated

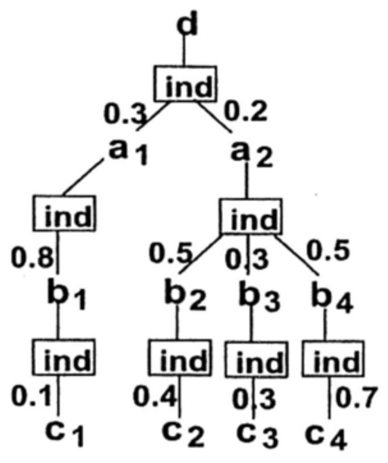


Figure 5.12: A PXML Tree

along the path on the left side in Fig. 5.11. Here, b_1 has two additional virtual children indicated φ_1 and φ_2, and b_2 has two additional virtual children φ_3 and φ_4. With the additional virtual nodes, we can process top-k probabilities using the same techniques we discussed for processing top-k probabilities for node queries.

5.4.1. Discussions on Tree Query

However, it is difficult to efficiently compute any PXML-RANK queries (Q, k), where Q is an arbitrary tree query, even for $k = 1$. We explain it using an example. Consider the PXML tree in Fig. 5.12, and a PXML-RANK query, (Q, k) where $Q = /\!/A[./\!/B]/\!/C$, and $k = 1$. There are 10 answers. For simplicity, we use a 3-tuple to indicate a resulting subtree for the PXML-RANK query. One resulting subtree is $r_1 = (a_1, b_1, c_1)$. There are other 9 resulting subtrees rooted at a_2 with any one of the three b_i, for $2 \leq i \leq 4$, and any one of the three c_j, for $2 \leq j \leq 4$. As one example, consider $r_2 = (a_2, b_2, c_3)$ where a_2 has three children $\{b_2, b_3, b_4\}$, b_2 is in the subtree rooted at b_2 and c_3 is in the subtree rooted at b_3. The fact states that our dynamic programming techniques cannot be used to efficiently compute top-k probabilities even for r_1 over the 10 subtrees. It is because that we cannot compute $r^h_{a_2,j}$ based on the values of $r^h_{b_i,j}$, for $2 \leq i \leq 4$, that are obtained for the children. Any combinations are possible, and we need to enumerate all possible worlds.

We give conditions. A general PXML-RANK tree query, (Q, k), can be computed in polynomial time if one of the conditions are satisfied. The conditions are imposed on the set of answers, $M = \{\varphi_1, \varphi_2, \cdots, \varphi_N\}$, to be ranked, which is generated by twigQuery(Q, T_P) in Algorithm 6. The conditions can be checked when processing twig queries without high overhead. Below, for a given tree query Q, we call a path in Q a primary path and denote it as P_Q. For example, $/\!/A/\!/B$ is a primary path of $/\!/A[./\!/C]/\!/B$. Note that an answer φ_i can be a subtree.

1. The edges of results do not overlap with each other. $\varphi_i \cap \varphi_j = \emptyset$ for $i \leq j$.

2. Every edge, e', of every answer $\varphi_i \in M$, which is not on the primary path P_Q, must be associated with $\rho_e(e') = 1$. (An ordinary edge, e', is considered as an edge with $\rho_e(e') = 1$).

3. Let φ_i and φ_j be two different answers. Suppose β_i and β_j are the paths in φ_i and φ_j that match P_Q, respectively. There exist two subtrees $\gamma_i = \varphi_i - \beta_i$ and $\gamma_j = \varphi_j - \beta_j$ where $\varphi_i = \beta_i \cup \gamma_i$ and $\varphi_j = \beta_j \cup \gamma_j$. If there exist a node v_i on the path β_i that is a descendant of a node v_j on the path β_j, then γ_i and γ_j must be identical.

As an example, consider the PXML tree, T'_P in Fig. 5.1(b), and the twig query $Q = /\!/A[./\!/C]/\!/B$ (Fig. 5.1(c)). The answer set $M = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$ where $\varphi_1 = \{(a_1, b_1), (a_1, c_1)\}$, $\varphi_2 = \{(a_1, b_1), (b_1, b_2), (a_1, c_1)\}$, $\varphi_3 = \{(a_2, b_3), (a_2, c_2)\}$, and $\varphi_4 = \{(a_2, b_3), (b_3, b_4), (a_2, c_2)\}$. Let $P_Q = /\!/A/\!/B$. The four answers, φ_i, for $1 \leq i \leq 4$, do not satisfy the first and the second condition. But they satisfy the third condition. For example, consider φ_1 and φ_2, $\beta_1 = (a_1, b_1)$ and $\beta_2 = (a_1, b_1)(b_1, b_2)$; and $\gamma_1 = (a_1, c_1)$ and $\gamma_2 = (a_1, c_1)$. There exist b_1 on β_1 that is a descendant of a node a_1 on β_2. $\gamma_1 = \gamma_2$.

The conditions allow us to compute a PXML-RANK tree query using our dynamic programming techniques. In brief, when either the second or the third condition is satisfied, we can compute top-k probabilities for a subtree φ_i as to compute top-k

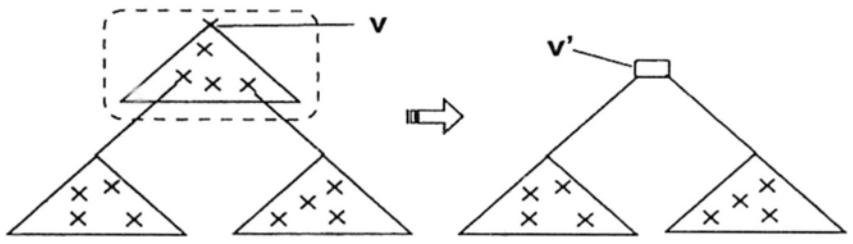


Figure 5.13: Compute top-k probabilities for a Subtree Answer

probabilities for the lowest node of φ_i on the primary path P_Q in the same way of processing path queries. Processing a tree query, when the first condition is satisfied, is complicated, because some nodes of a subtree answer φ_i can be ancestors of another subtree answer φ_j. We discuss our main ideas below, due to the space limit.

Let $M = \{\varphi_1, \varphi_2, \cdots\}$ be a set of the subtree answers to be ranked, and let φ be the current subtree. We call a subtree answer φ_i a h-answer if it has a larger score than the current's. For each φ_i, we use $root(\varphi_i)$ to denote the root of φ_i, P_{φ_i} to denote the path from the root node in the PXML tree to $root(\varphi_i)$, and $Pr_{\varphi_i} = \Pi_{e \in \varphi_i} \rho_e(e)$ if there are no mutually exclusive nodes in φ_i, otherwise $Pr_{\varphi_i} = 0$. We use $r_{v,j}$ instead of $r_{v,j}^{\omega(\varphi)}$ for short below. First, for computing the probability $\Pr(j$ h-answers appear $\mid \varphi$ appears), we set all $\rho_e() = 1$ for the edges in $P_\varphi \cup \varphi$, and remove the branches which are mutually exclusive with any nodes in $P_\varphi \cup \varphi$. We cannot simply condense P_φ to a node, because P_φ may contain some part of other h-answers. The process of setting $\rho_e() = 1$ serves the same purpose of condensing a path into a node. Second, we compute $r_{v,j}$ where a subtree answer φ_i is rooted on v. Note that v is an ordinary node. We condense φ_i to a virtual node v' and remove the branches that are mutually exclusive with any nodes in φ_i (see Fig. 5.13). The true $r_{v,j}$ is computed as follows: $r_{v,j} = r_{v,j} - Pr_{\varphi_i} \times r_{v',j} + Pr_{\varphi_i} \times r_{v',j-1}$, where the $r_{v,j}$ appears on the right side is computed on the left tree in Fig. 5.13 where every node/edge in φ_i is considered separately. The existence of the entire φ_i is ensured on the right tree in Fig. 5.13. It can be easily handled when multiple subtree answers are rooted at the same node v.

5.5. Performance Studies

We conduct extensive experiments to test the performance of our algorithms. We have implemented our PXML-RANK algorithm. The main algorithm to be tested is P-RANK. We have implemented P-RANK without Optimization-I and Optimization-II, the P-RANK algorithm using Optimization-I, and the P-RANK algorithm using both Optimization-I and Optimization-II. We denote them as pRank, pRank-I and pRank-II, respectively. All algorithms were implemented in C++. We conducted all the experiments on a 2.8GHz CUP and 2GB memory PC running XP.

We use two real datasets, DBLP (`http://dblp.uni-trier.de/xml/`) and Mondial (`http://www.informatik.uni-freiburg.de/~may/lopix/lopix-mondial.html`), and the synthetic XML benchmark dataset X-Mark (`http://monetdb.cwi.nl/xml/`) for testing. For XMark, we also generate many datasets with different sizes. For each XML dataset used, we generate the corresponding PXML tree, using the same method as used in [55]. We visit the nodes in the original XML tree from top to bottom. For each node v visited, we randomly choose some distribution nodes with random types and probability distributions to be the children of v, then for the original children of v, we choose some of them to be the children of the new generated distribution nodes. We control the percentage of the distribution nodes to generate different PXML trees for each dataset.

In answering a PXML-RANK query, we first compute all answers using a modified twig pattern matching algorithm based on [75]. The algorithm we use can process the entire PXML tree in a streaming manner, and therefore does not need to keep the entire PXML tree in memory. Then we prune nodes, v, on the PXML tree if the subtree rooted at v dose not effect the ranking, and get another projected PXML tree. We run our three algorithms pRank, pRank-I and pRank-II, to compute top-k probabilities for all results over the projected PXML tree if they satisfy one of our conditions. For each test, we record the time and space consumption of all algorithms. The time consumption

ID	Query	Result
D1	dblp//book[.//author]/key	$1,684$
D2	dblp//article[.//title//sub]/key	$2,928$
D3	dblp//proceedings[key]//series[href]	$5,909$
D4	dblp//incollection[key]//author	$8,842$
D5	dblp/article[.//cite[label]]//key	$13,785$
M1	mondial//river[.//located[country]/province][id]/name	237
M2	mondial//city[country][.//population/year][latitude]//province	705
M3	//country[.//province[name][population]/city[id]][capital]/total_area	$2,595$
M4	mondial/organization[established][headq]/members[type]	$5,226$
M5	mondial//organization[.//members[type]/country][name]/abbrev	$7,505$
X1	site//category[.//text/bold]//id	712
X2	site//description//keyword/emph	1824
X3	//namerica//item[.//parlist//listitem//listitem]/id	$3,043$
X4	//closed_auctions/closed_auction//itemref[item]	$5,850$
X5	//open_auctions/open_auction[id]//author[person]	$7,200$

Table 5.2: Queries Used for All Datasets

consists the query processing time to generate all the results, the projection time and the time for computing top-k probabilities for all results. The main space consumption is caused by maintaining $r^h_{v,j}$ values. For pRank, $r^{\omega(\varphi_i)}_{v_i,j}$ values for a node v_i can be released when $r^{\omega(\varphi_i)}_{v,j}$ has been computed where v is the parent of v_i. For pRank-I and pRank-II, in order to share the computational cost, $r^{\omega(\varphi_i)}_{v,j}$ values need to be kept for computing $r^{\omega(\varphi_{i+1})}_{v,j}$. pRank-II consumes more memory because Optimization-II needs to maintain complete binary trees.

For the DBLP dataset, the original XML tree has $13,318,516$ nodes with 41 different tags and the maximum depth of 6. We range the percentage of distribution node from 10% to 50% and generate 5 different PXML trees. The queries used for the DBLP dataset are listed in Tab. 5.2 from D1 to D5, with combination of both // and

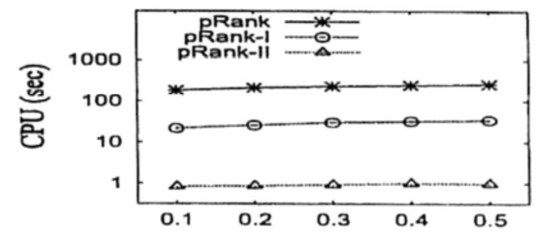
Parameter	Range	Default
DistNode(All)	10%, 20%, 30%, 40%, 50%	30%
Top-k (All)	10, 20, 30, 40, 50	30
Query(DBLP)	D1, D2, D3, D4, D5	D3
Query(Mondial)	M1, M2, M3, M4, M5	M3
Query(XMark)	X1, X2, X3, X4, X5	X3
Node Number(XMark)	0.5, 1, 1.5, 2, 2.5 ($\times 10^6$)	1.5

Table 5.3: Parameters Used for Testing

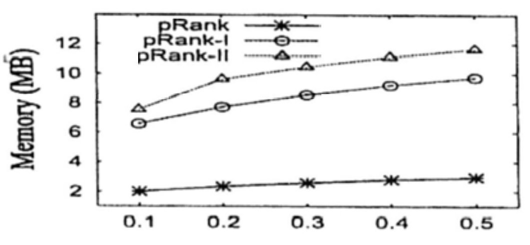
/ operators. We list them in increasing order of result size. The parameters used for testing DBLP dataset are listed in Tab. 5.3, where DistNode means the percentage of distribution node. For the Mondial dataset, the original XML tree has 70,459 nodes with 51 different tags and the maximum depth of 5. The queries used and parameters with default values are listed in Tab. 5.2 from M1 to M5 and Tab. 5.3 respectively. For the XMark datasets, we generate 5 different datasets with different sizes for testing. There are 77 different tags with the maximum depth of 12 for each of the generated XML trees. We set the percentage of distribution node to be 30% and convert them to the corresponding PXML tree. The number of nodes for each PXML tree is shown in the last row of Tab. 5.3. The queries used and parameters with default values are listed in Tab. 5.2 from X1 to X5 and Tab. 5.3 respectively.

5.5.1. Test-DBLP

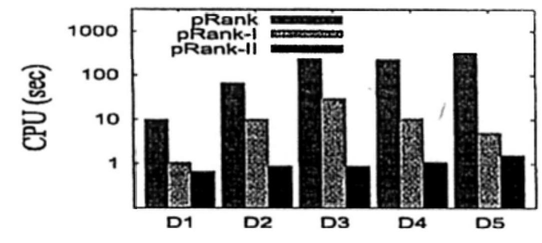
Fig. 5.14 shows the testing results over DBLP datasets. From Fig. 5.14(a) and 5.14(b), we know that when the percentage of distribution nodes increases, both the time and memory consumption for the three algorithms marginally increase. pRank-II is more than 300 times faster than pRank although cost about three times more memory than pRank. pRank-I is more than 10 times faster than pRank although cost about 2 times more memory than pRank. Fig. 5.14(c) and Fig. 5.14(d) show that when the number of results increases, the time and memory used for the three algorithms do not nec-



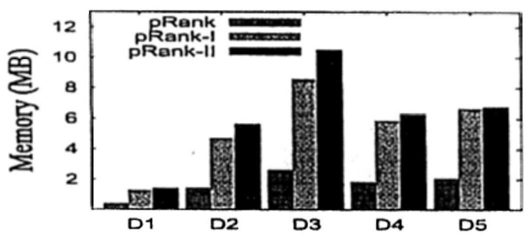
(a) Vary DistNode



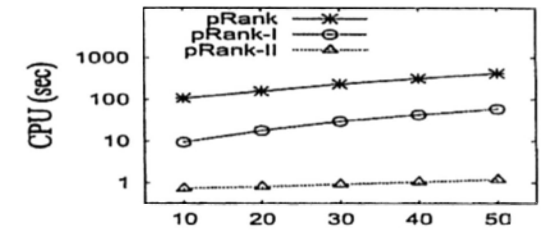
(b) Vary DistNode



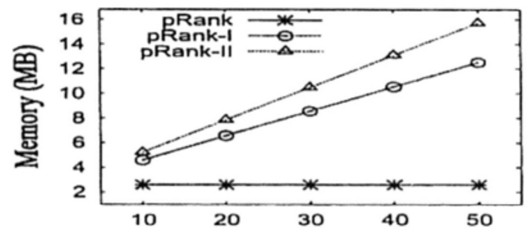
(c) Vary Query



(d) Vary Query



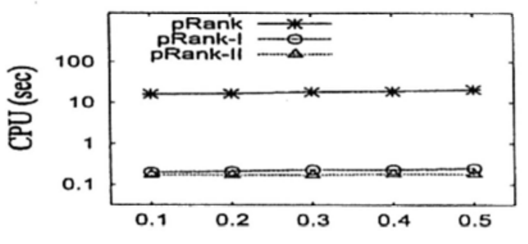
(e) Vary Top-k



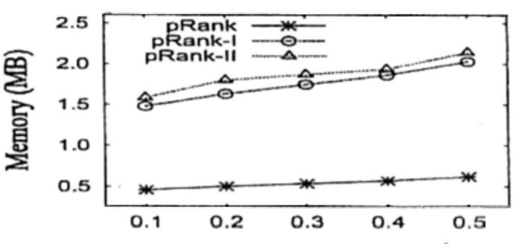
(f) Vary Top-k

Figure 5.14: Testing DBLP Dataset

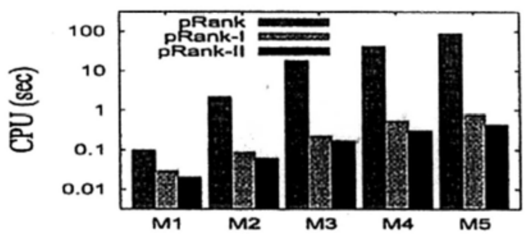
essarily increase. It is because the increasing of the number of results dose not mean that the size of the projected PXML tree to be tested also increases. The time for all the three algorithm is influenced by both the number of results and the size of the projected PXML tree. The memory consumption for all the three algorithms reflects the size of the projected tree. In Fig. 5.14(e) and Fig. 5.14(f), we can see that, when k increases, the time for all the three algorithms will increase. The memory consumption for pRank-I and pRank-II will increase linearly with k, while the memory consumption for pRank is not influenced by k. pRank-II is also much more faster (about 100 times faster) than pRank although cost some more memory (not larger than 8 times more). The performance of pRank-I is between the other two algorithms.



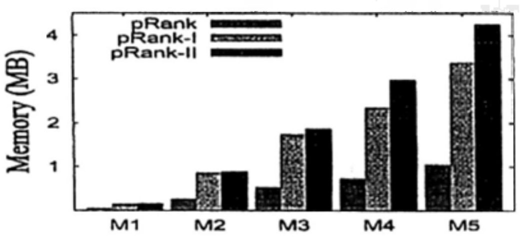
(a) Vary DistNode



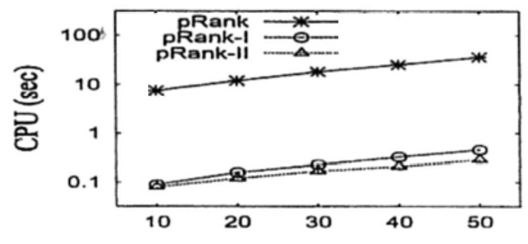
(b) Vary DistNode



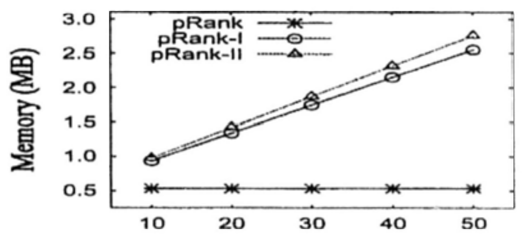
(c) Vary Query



(d) Vary Query



(e) Vary Top-k



(f) Vary Top-k

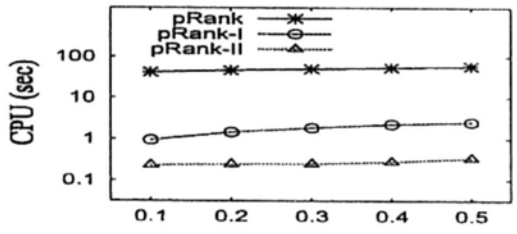
Figure 5.15: Testing Mondial Dataset

5.5.2. Test-Mondial

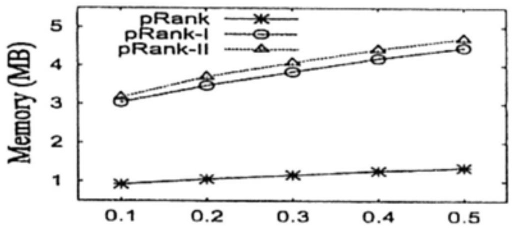
Fig. 5.15 shows the performance of the three algorithms over the Mondial dataset. Fig. 5.15(a) and Fig. 5.15(b) show that when the percentage of distribution nodes increases, the time and memory consumption for all the three algorithms will increase. pRank-II is more than 100 times faster than pRank, and is 2 times faster than pRank-I. The memory consumption of pRank-II and pRank-I are almost the same and are 3 times more than pRank. In Fig. 5.15(c) and Fig. 5.15(d), when the number of results increases in the Mondial dataset, the size of the projected tree will also increase (which is reflected by the increasing of memory consumption), so the time for all the three algorithms will increase. Fig. 5.15(e) and Fig. 5.15(f) show that when k increases, the time for all the three algorithms will increase. The memory consumption for pRank-I and pRank-II will increase linearly while the the memory consumption for pRank remains the same. Comparing to the DBLP dataset, in the Mondial dataset, the time for pRank-I is more similar to pRank-II, because in the DBLP dataset, we can always find nodes with very large number of children in the projected PXML tree, for example the root node tagged "DBLP", which will decrease the efficiency of pRank-I, whereas, in the Mondial dataset, the degree of each node is not large which makes the advantages of optimization-II less obvious.

5.5.3. Test-XMark

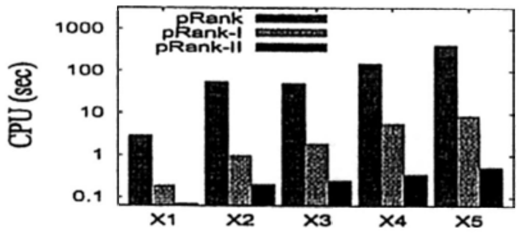
Fig. 5.16 shows the performance of the algorithms on the XMark datasets. Fig. 5.16(a) and Fig. 5.16(b) show that when the number of distribution nodes increases, the time and memory consumption for all the three algorithms will also increase. Fig. 5.16(c) and Fig. 5.16(d) show that when the number of result increases, the time for all the three algorithms will increase if the size of the projected PXML tree (memory consumption) increases. Otherwise, the time is influenced by the size of the projected PXML tree. Fig. 5.16(e) and Fig. 5.16(f) show the performance of the three algorithms when increasing k, which are similar to that in the DBLP dataset. In Fig. 5.16(g) and Fig. 5.16(h), when the size of the data (Node Number) increases, the time and memory consumption for all the three algorithms will increase, because when the size of the data increases, both the number of result and the size of the projected PXML tree will increase. The time used for pRank-II is about 100 times faster than pRank, and the time used for pRank-I is about 30 times faster than pRank for all experiments on the XMark datasets. The memory consumption for pRank-I is similar to the memory cost for pRank-II, and is at most 5 times the memory consumption for pRank for all experiments on the XMark datasets.



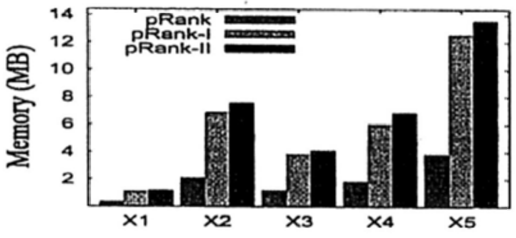
(a) Vary DistNode



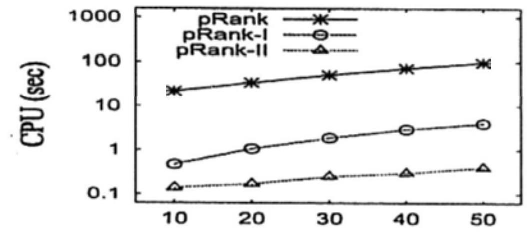
(b) Vary DistNode



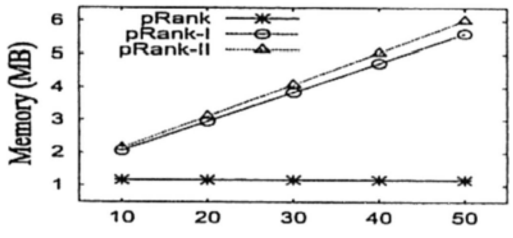
(c) Vary Query



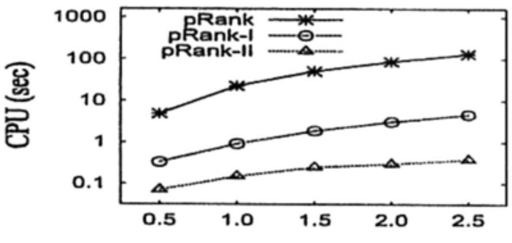
(d) Vary Query



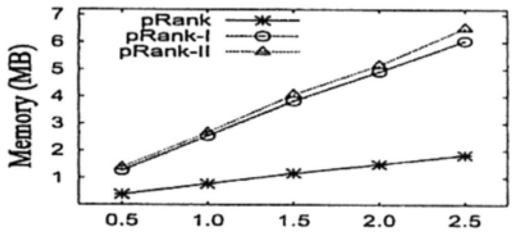
(e) Vary Top-k



(f) Vary Top-k



(g) Vary Node ($\times 10^6$)



(h) Vary Node ($\times 10^6$)

Figure 5.16: Testing XMark Datasets

CHAPTER 6

CONTEXT-SENSITIVE DOCUMENT RANKING

6.1. Introduction

Keyword search is an important issue in the information retrieval area [9] and has been widely and extensively studied over decades. Most of the existing keyword search approaches return a set of documents with ranking that is most important and relevant to a set of user-specified keywords. However, in many real applications, the documents may not be the only data source, and there are other forms of interrelated data for the same application domain. In this chapter, we study a new ranking problem that is to rank documents, in particular, when users cannot find any documents that contain all the given keywords from the set of documents itself, or when users can possibly find more relevant documents by exploring the documents as well as the interrelated data. In other words, we consider how to effectively use the additional relevant information, that may be maintained in a database, to rank documents. We explain our motivation using an example.

Fig. 6.1 shows a collection of reviews (documents) on the left side about movies. There are three movie reviews with review titles, namely, "Ocean's Twelve", "Seven", and "Twelve Monkeys". In the movie review entitled "Ocean's Twelve", it wrote "... those things make the film truly great ...". These movie reviews are about movies

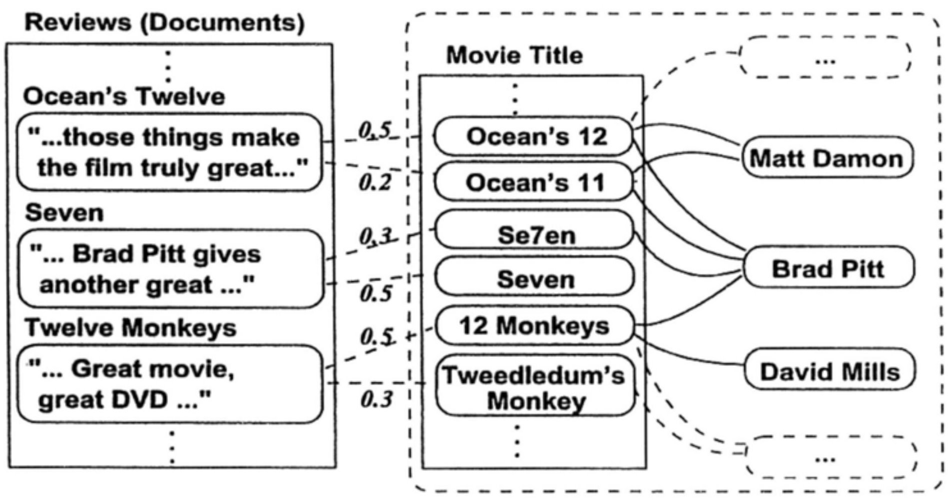


Figure 6.1: Movie Reviews & Movie Records

which can be maintained as tuples in a multi-attribute table, consisting of the movie title and other possible attribute values as illustrated in Fig. 6.1 on the right side (refer to IMDB `http://www.imdb.com/interfaces`). The names of directors or actors or actresses may not appear in the movie reviews or even in the movie title, but they appear in some tuples in some tables in the IMDB movie database that are linked to the movie titles via relationships such as movies directed by directors, and movies played by actors/actresses, based on the primary keys and foreign key references. We call such interrelated information maintained in multi-attribute tables a context to which the movie review is directly/indirectly related. The relationship between a movie review (document) and a movie title (or simply movie) maintained in a multi-attribute table is with uncertainty. Such uncertainty may be caused by missing titles, or mismatching of the discussions in the body of movie review, or different movies with the same title. For example, the movie review entitled "Ocean's Twelve" may not exactly match the movie title "Ocean's 12" maintained in the multi-attribute table. In Fig. 6.1, there is a link between a movie review and a movie in a table, where such uncertainty is indicated as a probability.

Suppose a user wants to know something "great" about the actor "Brad Pitt", by issuing a 2-keyword query, "great" and "Brad Pitt", against the collection of movie reviews. The movie review entitled "Seven" will have the highest score, followed by movie review on "Twelve Monkeys" and "Ocean's Twelve". Suppose that the user issues a keyword query with two additional keywords, "Matt Damon" and "David Mills", the movie reviews and their rank will remain the same, since none of the reviews contain the additional two keywords. As observed from Fig. 6.1, there is information about "Matt Damon" and/or "David Mills" indirectly linked to the movies and the movie reviews, as a part of structural information, which are not effectively used to answer users' keyword queries. Furthermore, assume a 2-keyword query is "great" and "David Mills". Without the context, there are no documents containing the two given keywords. With the context as shown in Fig. 6.1, it is highly possible to find movie reviews on movies the two keywords are most related.

The main issue we study is how to rank documents using a set of keywords, given a context that is associated with the documents. The main contributions of this chapter are summarized below. First, we study a new ranking problem to rank documents for a given set of keywords. The uniqueness of the problem is that the documents to be ranked are associated with sets of interrelated multi-attribute tuples called context, which contains additional information that assist users to rank the relevant documents with some uncertainty. Second, we model the problem using a graph $\mathcal{G}$ with two different kinds of nodes (document nodes and multi-attribute nodes), and discuss its score function, cost function, and ranking with uncertainty. Third, we propose new algorithms to rank documents that are most related to the user-given keywords by integrating the context information. We also conducted extensive experimental studies to show the effectiveness of our approaches using a real dataset.

The remainder of this chapter is organized as follows. Section 6.2 gives the problem statement, followed by discussions on related work in Section 6.3. We discuss score functions and ranking in Section 6.4 and Section 6.5, respectively. In Section 6.6, we give an overall system architecture, and discuss how to rank with uncertainty. Experimental studies are given in Section 6.7.

6.2. Problem Statement

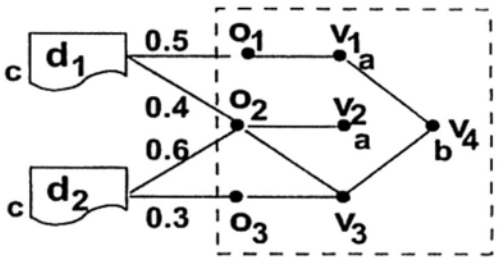
We consider a set of documents, $D_A = \{d_1, d_2, \cdots\}$. In addition, we assume there exists a context, $\mathcal{C}$, which is considered as a multi-attribute graph $G_R(V, E)$ that specifies the knowledge about documents. A multi-attribute graph G_R is capable of representing all the tuples in a relational database, where a node represents a tuple, and an edge between two nodes represents a foreign-key reference between the two corresponding tuples. All nodes (tuples) in the same relation are said to have the same type. There exists a set of specific A-typed nodes in $G_R(V, E)$, V_A $(\subseteq V)$ which are explicitly linked to the documents in D_A. A document may be linked to several A-typed nodes, and an A-typed node may be related to several documents. To integrate the set of documents, D_A, and the multi-attribute graph, $G_R(V, E)$, we consider a weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V} = V \cup D_A$, and $\mathcal{E} = E \cup E_D$ where E_D is a set of pairs, (d_i, v_j), if d_i is related to an A-typed node v_j. We call a node in D_A a document node, and a node in G_R a multi-attribute node. We assign edge-weights to edges in the subgraph $G_R(V, E)$ of $\mathcal{G}$ based on [54, 30]. In detail, for a foreign key reference from u to v, the edge weight for (u, v) is given as Eq. (6.1), and the edge weight for (v, u) is given as Eq. (6.2).

$$w_e((u, v)) = 1 \tag{6.1}$$

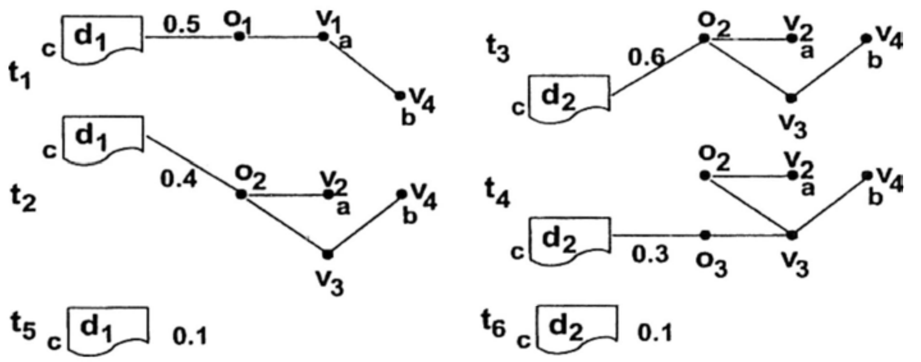
$$w_e((v, u)) = log_2(1 + N_{in}(v)) \tag{6.2}$$

where $N_{in}(v)$ is the number of nodes that refer to v. We assign the edges in E_D with a non-zero weight which is a normalized similarity score computed as given in [6]. We consider the weight assigned to $(d_i, v_j) \in E_D$ as the probability that d_i is related to v_j. A document node, d_i, may be related to several different multi-attribute nodes, v_j and v_k, with probability distribution, prob(d_i, v_j) and prob(d_i, v_k), respectively. However, the probability must satisfy $\sum_{v_j}$ prob$(d_i, v_j) \leq 1$ in order to form a distribution.

Example 1: Fig. 6.2(a) shows a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. There are two documents in $D_O = \{d_1, d_2\}$, and a multi-attribute graph $G_R(V, E)$ illustrated in the dashed rectan-



(a) A Sample Graph



(b) Local Contexts

Figure 6.2: Example

gle. There are two types of nodes in V, namely O and V, where the former consists of $\{o_1, o_2, o_3\}$, and the latter consists of $\{v_1, v_2, v_3, v_4\}$. The edges between two n-odes in G_R show how the multi-attribute nodes are interrelated. The edge (d_1, o_1) and (d_1, o_2) show that d_1 is related to o_1 and o_2 with probability distribution, 0.5 and 0.4, respectively. The edges (d_2, o_2) and (d_2, o_3) show that d_2 is related to o_2 and o_3 with probability distribution, 0.6 and 0.3, respectively. There are three distinct keywords appearing in $\mathcal{G}$: a, b, and c. The keyword a appears in multi-attribute nodes v_1 and v_2, the keyword b appears in the multi-attribute node v_4, and the keyword c appears in document nodes d_1 and d_2. □

Problem Statement: Finding top-k documents for an l-keyword query, $Q = \{w_1, w_2, \cdots, w_l\}$, against graph $\mathcal{G}$.

The ranking is based on the following consideration. First, if a document node itself contains all the l-keywords, it should be ranked higher. If a document node, d_i, does not contain all the l-keywords, it is ranked based on d_i and its associated local context (which will be introduced next) that all together contain all the l-keywords. Also, because a document d_i may be related to several local contexts that contain all the keywords individually, it needs to identify a local context that will make the document d_i ranked high. The selection of such a local context is done by considering all possibilities.

6.3. Related work

Fuzzy object matching: In the data integration and data cleaning, fuzzy object matching is a central problem, which is to reconcile objects from different collections that have used different naming conventions. Fuzzy object matching is also known as record linkage, de-duplication or merge-purge [6, 42, 27]. For text documents, edit distance [89] and Jaccard similarity on q-grams [38] are commonly used as the similarity score function. Efficient algorithms for computing the similarity scores are discussed in [10, 91]. The basic approach of fuzzy object matching is to first compute a similarity score between pairs of objects, and then classify objects into matches and non-matches using a threshold [6]. As the popularity of probabilistic databases, in [76], Ré et al. propose to convert these similarity scores into probabilities, which represent the confidence of matching between these two objects, and store them directly in a probabilistic database.

Keyword Search in RDBs: Keyword search in relational databases has been studied extensively recently. It provide users an easy way to get insights of the underling RDB. Most of the works concentrate on finding minimal connected tuple trees from an RDB [14, 43, 65, 30]. There are two approaches, based on the representation of data. One accesses data directly using SQL, where the data is stored in an RDB [43, 65]. The other materializes an RDB as a graph, then all the operations are on the graph [14, 30]. As an exception, there is work on finding communities, which is a multi-center induced subgraph, on a materialized graph based on a keyword query [74].

All the works above only consider one single database. Li et al. [60] propose keyword search over heterogeneous data, e.g. unstructured data (e.g. web pages, linked together through hyper links), semi-structured data (e.g. XML), structured data (e.g. Database). But in their work, there is no connection between different datasets, they are different connected components in a whole graph. Sayyadian et al. in [78] propose a system Kite to answer a keyword query over multiple databases. Kite combines schema matching and structure discovery techniques to find approximate foreign-key joins across heterogeneous databases. In their work, the confidence of foreign-key joins in a result contributes linearly to the final score.

6.4. The Score Function

Given an l-keyword query $Q = \{w_1, w_2, \cdots, w_l\}$, we first discuss a concept, called a local context, for a document node d_i in $\mathcal{G}$ to be possibly ranked. A local context is a connected tree in $\mathcal{G}$, denoted as $T(V, E)$. The local context contains one document node, d_i, that can also link to a connected rooted tree, $\triangle(V, E)$, as a subgraph in G_R. Note that $G_R \subset \mathcal{G}$. We say that the local context T supports the document d_i. A connected tree $\triangle(V, E)$ specifies an interrelated tuple structure among tuples that is related to the document node d_i. The root of $\triangle_j(V, E)$ is an A-typed node, say v_j, and the probability for d_i to link to v_j is $\text{prob}(d_i, v_j)$. A keyword w is contained in $T(V, E)$, if it appears in the document node d_i or in any multi-attribute node in the corresponding $\triangle(V, E)$. A local context, $T(V, E)$, contains l-keywords based on an "or" semantics. In other words, a local context for a specific document node d_i may not contain all the required l-keywords. For example, a document node d_i may not link to any A-typed node in $\mathcal{G}$, or a document node d_i may link to an A-typed node, v_j, but there does not exist a connected tree $\triangle(V, E)$, rooted at v_j, containing the required keywords that do not appear in d_i. In such cases, the corresponding $T(V, E)$ does not contain all the required keywords. On the other hand, for a given document node d_i, there may exist more than one connected trees, for example, two trees $\triangle_j$ and $\triangle_k$

rooted at v_j and v_k. Each of the trees contains all or some of the required keywords that do not appear in d_i. We treat it as two local contexts. In order to select one of them as the local context for d_i, we define a score function, denoted $score(T, Q)$.

In the following we use T and t_i as local contexts interchangeably. Here, T means a general local context, whereas t_i means a local context for a specific document node.

Example 2: Reconsider Fig. 6.2(a). Given a 3-keyword query, $Q = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, we show the six local contexts, t_i, for $1 \leq i \leq 6$, in Fig. 6.2(b). Among them, the local contexts, t_1, t_2, and t_5, support document d_1, and the local contexts, t_3, t_4, and t_6, support document d_2. Assume $score(t_i, Q)$, for $1 \leq i \leq 6$, are the scores, which we will discuss in detail. For document node d_1, with probability 0.5 the score is $score(t_1, Q)$, with probability 0.4 the score is $score(t_2, Q)$, and with probability 0.1 the score is $score(t_5, Q)$. □

As shown in Example 2, there are local contexts with only a document node d_i if $1 - \sum_{v_j} \mathsf{prob}(d_i, v_j) > 0$. For example, for document d_1, the local context t_5 is with probability $1 - 0.5 - 0.4 = 0.1$; for document d_2, the local context t_6 is with probability $1 - 0.6 - 0.3 = 0.1$. In the following, we denote the probability of a local context, t' for a document d_i as $\mathsf{prob}(t')$. It is equal to $\mathsf{prob}(d_i, v_j)$ if the edge (d_i, v_j) appears in the local context t', and it is $\mathsf{prob}(d_i) = 1 - \sum_{v_j} \mathsf{prob}(d_i, v_j)$ otherwise.

Score Function for Local Context: We discuss score functions to score a local context for a document below, and will discuss ranking documents in local contexts, in the next section, which is based on the scores and probabilities of having such scores.

Recall that a local context t' consists of a document node d_i which may link to a connected tree $\triangle_j$ rooted at a multi-attribute node v_j. There are two main components in our score function to score a local context t'. One is related to the keywords it contains. The other is how to evaluate the cost of using a connected tree $\triangle_j$. The connected tree $\triangle_j$ is needed if it contains some keywords that are missing in the document node d_i. However, there is a cost to use such a connected tree $\triangle_j$ to include more keywords. It is because that there may be a $\triangle_j$ which is very relevant to d_i, and there may be a $\triangle_j$ which is not very relevant to d_i.

First, for the first component, we consider a local context T as a virtual document [63, 65], and treat all local contexts as a supporting-set, where each local context can be scored using an IR-style relevance score, with respect to the l-keyword query $Q = \{w_1, w_2, \cdots, w_l\}$. It is defined below [65].

$$score_{IR}(T, Q) = \sum_{w \in T \cap Q} \frac{1 + \ln(1 + \ln(tf_w(T)))}{(1-s) + s \cdot \frac{dl_T}{avdl}} \cdot \ln(idf_w) \tag{6.3}$$

Here, $w \in T \cap Q$ indicates the subset of keywords of Q appearing in T. $tf_w(T)$ is the term frequency, e.g. the number of occurrences of the keyword w in the local context T; $idf_w = \frac{N+1}{df_w(Doc)}$, where $df_w(Doc)$ is the document frequency, e.g. the number of virtual documents that contain the keyword w and N is the total number of the virtual documents; $(1-s) + s \cdot \frac{dl_T}{avdl}$ is the length normalization, where dl_T is the document length of the local context T, $avdl$ is the average document length among the whole virtual document collection, and s is a parameter. We also consider a completeness coefficient for $score_{IR}(T, Q)$, denoted as $complete(T, Q)$, which specifies how important for a local context to contain a keyword. The more keywords contained the better.

$$complete(T, Q) = \frac{\sqrt[p]{\sum_{w_i \in Q} (I(w_i, T) \cdot ln(idf_{w_i}))^p}}{\sqrt[p]{\sum_{w_i \in Q} (ln(idf_{w_i}))^p}} \tag{6.4}$$

Here, $I(w_i, T)$ is an indicator function, it equals to 1 if and only if local context T contains keyword w_i and 0 otherwise. The value of $complete(T, Q)$ is in the range $[0, 1]$. $complete(T, Q)$ is equal to 1 if all the keywords appear in T. The intuition behind the completeness function is that, if more (distinct) keywords appear in a local context, then it should be ranked higher; if the number of (distinct) keywords appeared is the same, then the more discriminative (larger $ln(idf_w)$) the keyword it contains the higher it ranks.

Second, for the second component, we define it to be the total distance between the document node d_i and the nodes in $\triangle_j$ that contain the keywords in a local context T [41].

$$cost_{st}(T, Q) = \sum_{w \in V(\triangle_j) \cap Q} dis(d_i, node(w)) \tag{6.5}$$

where $V(\triangle_j)$ denotes the set of nodes in T that appear in $\triangle_j$, $node(w)$ denotes the node in $\triangle_j$ that contains the keyword w, and $dis(a,b)$ is the distance between two nodes in the local context based on the edge weights.

Combining the two components, the score of a local context, T, with respect to a query $Q = \{w_1, w_2, \cdots, w_l\}$ is given as follows.

$$\begin{aligned} score(T,Q) &= \alpha \cdot complete(T,Q) \cdot score_{IR}(T,Q) \\ &\quad -(1-\alpha) \cdot cost_{st}(T,Q) \end{aligned} \tag{6.6}$$

where α is a parameter in the range of $[0,1]$, that specifies the relative importance of the two factors. A local context, T, is ranked higher if its score $score(T,Q)$ is larger, which implies that either its textual component score $score_{IR}(T,Q)$ is larger or its structural component cost $cost_{st}(T,Q)$ is smaller. In our formulation, a local context does not necessarily include all the l-keywords, even though more keywords is demanded to increase the completeness factor $complete(T,Q)$. As indicated in Eq. (6.6), if we include one more node in a local context that contains a keyword, it increases the IR-styled score (Eq. (6.3)) and completeness factor (Eq. (6.4)), but it also increases the cost (Eq. (6.5)). Increasing the IR-styled score or completeness factor implies that the local context is more related to the query Q, which makes the final score (Eq. (6.6)) larger (higher rank). Increasing the cost implies that the local context (connected tree) becomes larger to include more multi-attribute nodes (or keywords), which makes the final score (Eq. (6.6)) smaller (lower rank).

Example 3: Consider the graph $\mathcal{G}$ as show in Fig. 6.2(a), $\{a, b, c\}$ are keywords. Document d_1 discusses o_1 with probability 0.5. If this possible matching pair (d_1, o_1) is a true match, we should find the best local-context that describes (supports) d_1. Context t_1 in Fig. 6.2(b) is a context with matching pair (d_1, o_1) that supports d_1. There also exists other contexts support d_1 with matching pair (d_1, o_1) besides t_1, for example, the subgraph induced by nodes $\{d_1, o_1\}$ or $\{d_1, o_1, v_1\}$. For simplicity, we use node set to denote the subgraph induced by it. It is easy to see that context (d_1, o_1) has the smallest structure cost $cost_{st}$, as it includes no multi-attribute graph edges. But it also has the smallest IR score $score_{IR}$ and completeness score $complete$, as it only includes one keyword "c". If we include one more node v_1, the context $\{d_1, o_1, v_1\}$ will have a larger score, because it contains two keywords "a" and "c", the increase of IR score and completeness factor compared to $\{d_1, o_1\}$ is larger than the increase of structure cost of including one more edge (o_1, v_1). The context t_1 has the largest score (Eq. 6.6) among all the contexts that support d_1 with matching pair (d_1, o_1). So if the matching pair (d_1, o_1) is true, we should say that the score of d_1 is $score(t_1)$. Then the score of d_1 is $score(t_1)$ with probability $\text{prob}(t_1) = 0.5$. □

6.5. Ranking with Uncertainty

In the previous section, we discuss our score function to score a local context. In this section, we concentrate on ranking. We adopt ranking with uncertainty probability as discussed in [86, 93, 45], where x-Relation model is used. In the x-Relation model [3, 93], there is a set of independent x-tuples (called generation rules in [86, 45]), where an x-tuple consists of a set of mutually exclusive tuples (called alternatives), represents a discrete probability distribution of the possible tuples it may take in a randomly instantiated data. Each alternative t has a score and a probability, where $\Pr(t)$ represents its existence probability over possible instances.

Example 4: Fig. 6.3(a) is an x-Relation, consists of two x-tuples, $\tau_1 = \{t_1(0.5), t_2(0.4), t_5(0.1)\}$, and $\tau_2 = \{t_3(0.6), t_4(0.3),\ t_6(0.1)\}$. The x-tuple τ_1 denote a probability distribution over t_1, t_2 and t_5, with probability 0.5 it takes t_1, with probability 0.4 takes t_2, and with probability 0.1 takes t_5. □

In our problem setting, the local contexts confirm to the x-Relation model. The set of local contexts that support the same document forms an x-tuple, and an x-tuple specifies that the local contexts belong to it are mutually exclusive. Intuitively, an x-tuple supports one document with mutually exclusive evidences. And the local contexts that support different documents are independent.

d_i	o_j	prob(d_i, o_j)	t_x
d_1	o_1	0.5	t_1
	o_2	0.4	t_2
	–	0.1	t_5
d_2	o_2	0.6	t_3
	o_3	0.3	t_4
	–	0.1	t_6

(a) x-Relation

Possible world (I)	$\Pr(I)$
$\{t_1, t_3\}$	0.30
$\{t_1, t_4\}$	0.15
$\{t_1, t_6\}$	0.05
$\{t_2, t_3\}$	0.24
$\{t_2, t_4\}$	0.12
$\{t_2, t_6\}$	0.04
$\{t_3, t_5\}$	0.06
$\{t_4, t_5\}$	0.03
$\{t_5, t_6\}$	0.01

(b) Possible Worlds

Figure 6.3: x-Relation Model

Example 5: For the graph in Fig. 6.2(a), there are six local contexts $\{t_1, t_2, t_3, t_4, t_5, t_6\}$, as shown in Fig. 6.2(b). The score distribution for the document d_1 is $\{(score(t_1), 0.5), (score(t_2), 0.4), (score(t_5), 0.1)\}$, because $\{t_1, t_2, t_5\}$ is the set of local contexts that support the document d_1. The score distribution for the document d_2 is $\{(score(t_3), 0.6), (score(t_4), 0.3), (score(t_6), 0.1)\}$, because $\{t_3, t_4, t_6\}$ is the set of local contexts that support the document d_2. Here, for example, t_1 and t_2 are mutually exclusive because they are about the same document d_1, and t_1 and t_3 are independent because one is about the document d_1 and the other is about the document d_2. Fig. 6.3(a) shows the six possible local contexts. Here, prob(d_i, o_j) is the probability of the local context, or in other words, the probability that d_i discusses about o_j. The 9 possible worlds, with non-zero probability, are shown in Fig. 6.3(b), together with their probabilities. The possible world $\{t_5, t_6\}$ means that, with probability 0.01, the two document nodes, d_1 and d_2 are not related to any of the three multi-attribute nodes, o_1, o_2, and o_3. The possible world $\{t_1, t_3\}$ means that, with probability 0.3, d_1 is related to o_1, and d_2 is related to o_2. Note that the sum of the probabilities of all the possible worlds is equal to 1. □

In our problem setting, an x-tuple specifies a probability distribution on the score for a document. In each possible world, it maintains exactly one local context from each x-tuple, then the size of a possible world will be exactly the size of D_A, i.e. $|I| = |D_A|$ for $\forall I \in pwd(\mathcal{X})$. And each possible world corresponds to a possible linkage between a document node in D_A and an A-typed multi-attributed node in G_R.

Expected Score: Several top-k definitions in the x-Relation model have been proposed recently based on the possible worlds semantics [86, 93, 45, 53, 28]. Before we discuss the top-k probability and the expected rank, we first study a straightforward strategy of getting score for documents deterministically which is not based on the possible world semantics. As we have shown, an x-tuple specifies a probability distribution on the score for documents. It is natural to define the score of a document as the expected score of the local contexts that support it. The expected score of a document is defined as follows,

$$EScore(d_i) = \sum_{T:doc(T)=d_i} score(T, Q) \cdot \mathsf{prob}(T) \tag{6.7}$$

where $score(T, Q)$ is the score of local context T against query Q (refer to Eq. (6.6)), and $\mathsf{prob}(T)$ is the probability of local context T. Recall that, $\mathsf{prob}(T) = \mathsf{prob}(d_i, v_j)$ if (d_i, v_j) appears as an edge in T, and $\mathsf{prob}(T) = 1 - \sum_{v_j} \mathsf{prob}(d_i, v_j)$ if T consists of a single node d_i. It is natural to use the expected score to rank documents. However, one drawback of the expected score is that the expected score is sensitive to the score values, high score value with low probability can result in high expected score. The possible worlds based uncertain ranking is insensitive to the exact score values, and it only depends on the relative order among the local contexts.

Example 6: For the graph $\mathcal{G}$ in Fig. 6.2(a) and a query $Q = \{a, b, c\}$, there are six local contexts as shown in Fig. 6.3(a) which supports the two documents. The expected score of d_1 and d_2 are, $EScore(d_1) = 0.5 \cdot score(t_1, Q) + 0.4 \cdot score(t_2, Q) + 0.1 \cdot score(t_5, Q)$, $EScore(d_2) = 0.6 \cdot score(t_3, Q) + 0.3 \cdot score(t_4, Q) + 0.1 \cdot score(t_6, Q)$. □

Top-k Probability: The top-k probability of a local context, T, denoted as $tkp(T)$, is the marginal probability that it ranks top-k in the possible worlds, given as below.

$$tkp(T) = \sum_{I \in pwd(\mathcal{X}), T \in topk(I)} \Pr(I)$$

where $T \in topk(I)$ means that the local context is ranked as one of the top-k local contexts in the instance I. And the top-k probability of a document is defined as below.

$$tkp(d_i) = \sum_{T:doc(T)=d_i} tkp(T) \tag{6.8}$$

The top-k probability of a documents is the summation of the top-k probability of the local contexts that support the document. Note that $\sum_{T:doc(T)=d_i} \text{prob}(T) = 1$ and the $\text{prob}(T)$ has been considered in $tkp(T)$.

Example 7: Consider the six local contexts in Fig. 6.3(a), assume the local contexts in decreasing score order are $t_1, t_3, t_2, t_4, t_6, t_5$. That is document d_2 in the local context t_6 has a larger score than d_1 in the local context t_5, $\{t_1, t_2, t_3, t_4\}$ have a larger score than $\{t_5, t_6\}$. Among $\{t_1, t_2, t_3, t_4\}$, t_1 is the most compact, then it has the largest score.

Now, we consider the top-1 probability. $tkp(t_1) = 0.30 + 0.15 + 0.05 = 0.5$, which is the summation of the probability of the first three possible worlds in Fig. 6.3(b). $tkp(t_2) = 0.12 + 0.04 = 0.16$. Note that the local context t_2 ranks the second place in the possible world $\{t_2, t_3\}$. $tkp(t_5) = 0$, because it ranks the first place in no possible world. Therefore, the top-1 probability of the document d_1 is $tkp(d_1) = tkp(t_1) + tkp(t_2) + tkp(t_5) = 0.66$. Similarly, we can get $tkp(d_2) = 0.34$. Comparing $tkp(d_1)$ and $tkp(d_2)$, we can see that document d_1 is more likely to be ranked higher than d_2. It is interesting to note that, without the multi-attribute graph G_R, d_2 is ranked higher than d_1, because we assume that the local context t_6 ranks higher than t_5. Also, consider the expected score (Eq. (6.7)), then both ranking orders are possible, because it is sensitive to the actually score values, and different score values with the same relative order will result in different ranking orders. □

Expected Rank The rank of a local context T in a possible world I is defined to be the number of local contexts whose score is lager than $score(T, Q)$ (so the top local context has rank 0), i.e., $rank_I(T) = |\{T_i \in I \mid score(T_i, Q) > score(T, Q)\}|$. The expected rank of a local context is defined as follow.

$$ERank(T) = \sum_{I \in pwd(\mathcal{X}), T \in I} rank_I(T) \cdot \Pr(I)$$

The expected rank of a local context is only defined on the possible worlds that contain it. The expected rank of a document is defined as below.

$$ERank(d_i) = \sum_{T: doc(T) = d_i} ERank(T) \tag{6.9}$$

The expected rank of a document is the summation of the expected rank values of the local contexts that support the document. It means the expected rank position for the document d_i in a randomly generated possible world. Note that the smaller the value $ERank(d_i)$ the higher it ranks.

Example 8: Following Example 7, consider the six local contexts in Fig. 6.3(a), assume the local contexts in decreasing score order are $t_1, t_3, t_2, t_4, t_6, t_5$. We consider the expected rank. $ERank(t_1) = 0$, because it ranks at the first place in the first three possible worlds in Fig. 6.3(b). $ERank(t_2) = 0.24$, because it ranks at the second place in the possible world $\{t_2, t_3\}$ and ranks at the first place in other possible worlds. $ERank(t_5) = 0.06 + 0.03 + 0.01 = 0.1$, because it ranks the second place in the last three possible worlds. So the expected rank value of document d_1 is $ERank(d_1) = ERank(t_1) + ERank(t_2) + ERank(t_5) = 0.34$. Similarly, we can get $ERank(d_2) = 0.66$. Comparing $ERank(d_1)$ and $ERank(d_2)$, we can see that the document d_1 ranks higher than d_2 based on the expected rank semantics. Note that the smaller expected rank value the higher it ranks. The ranking is the same of that based on top-1 probability in Example 7. □

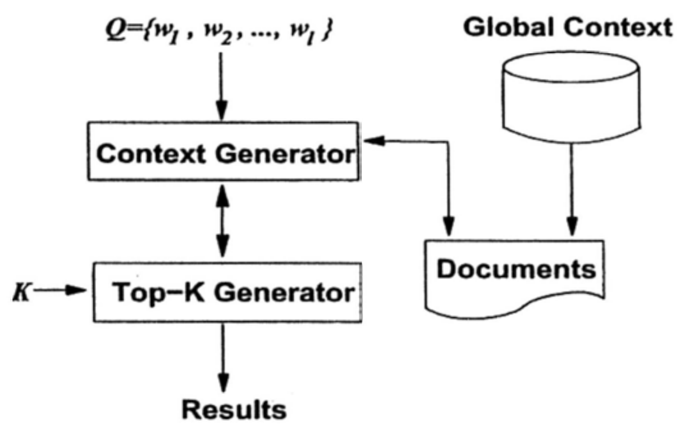


Figure 6.4: The System Architecture

6.6. Query Processing

Given an l-keyword query, Q, against a graph $\mathcal{G}$, we first generate the set of local contexts, $\mathcal{T} = \{t_1, t_2, \cdots\}$, where a local context t_x, scored $score(t_x, Q)$, contains a unique document node, d_i, and has a probability prob(T), which means that the local context t_x supports that document d_i has a score $socre(t_x, Q)$ with probability prob(T). If the document node d_i is an isolated node in $\mathcal{G}$, then there is a local context in $\mathcal{T}$ that contains only one node d_i, with probability one. Each local context supports one document, given the set of local contexts, the score of documents is a probability distribution. After getting the local contexts, documents are ranked based on the probability score distribution. We integrate the information to rank top-k documents.

An overview of the system architecture is shown in Fig. 6.4. A user submits an l-keyword query $Q = \{w_1, w_2, \cdots, w_l\}$ and k, and rank the top-k documents under a global context G_R. As a preprocess step, it constructs the graph $\mathcal{G}$ by linking each document $d_i \in D_A$ to several high similar A-typed multi-attribute nodes in G_R. There are two main components in the system, namely, *Local Context Generator* (ConGen) and *Top-k Generator* (TopKGen). Here, ConGen generates the local contexts incrementally in the decreasing order of the score function (Eq. (6.6)) for a user-given l-keyword query $Q = \{w_1, w_2, \cdots, w_l\}$. It is important to emphasize that ConGen generates the local contexts one-by-one in the decreasing order of the score function, which means that the local context ranked higher will be generated earlier. TopKGen will process the local contexts in the decreasing score order by iteratively calling ConGen to obtain a generated local context in one iteration.

Example 9: Consider Fig. 6.2(a), the subgraph in the dotted box is a multi-attribute graph, d_1 and d_2 are two documents, a user submit a keyword query $Q = \{a, b, c\}$. The system first build probabilistic linkages between D_A and G_R to form a graph $\mathcal{G}$ as in Fig. 6.2(a). Then the local context generator will generate six local contexts, $t_1, \cdots, t_4$ as shown in Fig. 6.2(b) and two other contexts with document node only, d_1 and d_2. After getting the set of contexts, we get a distribution of the scores for each document, e.g. the score distribution for document d_1 is $\{(score(t_1), 0.5), (score(t_2), 0.4), (score(d_1), 0.1)\}$, because $\{t_1, t_2, d_1\}$ is the set of contexts that support document d_1. Finally, we can rank the documents based on their score distributions. □

6.6.1. Local Context Generation

A naive way to compute a local context is to start from a pair of a document node and the corresponding A-typed multi-attribute node, (d_i, v_j), with prob$(d_i, v_j) > 0$, extend the local context by finding the shortest distance from the node v_j to a node containing a keyword in $\mathcal{G}$. A node containing a keyword will be included as a part of the local context if it increases the score (Eq. (6.6)) or in other words makes the rank of the local context higher. All the local contexts are then sorted in the decreasing order and returned to TopKGen one-by-one in every iteration on demand.

We propose an incremental algorithm, which generates the local contexts incrementally one by one in the decreasing order of the score, and we do not need to compute all the local contexts. There are two main issues. First, the number of possible pairs, (d_i, v_j), in graph $\mathcal{G}$, can be very large. Second, graph $\mathcal{G}$ itself can be very large.

For the first issue, we do not need to compute all the possible local contexts. A local context containing (d_i, v_j) is upper bounded by the following upper bounded score,

$$\alpha \cdot \sum_{w \in Q} \frac{1 + \ln(1 + \ln(tf_w(d_i) + 1))}{(1 - s) + s \cdot \frac{dl_T}{avdl}} \cdot \ln(idf_w). \tag{6.10}$$

The upper bound is given based on the following reason. When a keyword is newly included in a local context, it will increase the term frequency by one, and also increase the structural cost non-negative. In the ideal situation, the multi-attribute node v_j contains all the keywords in the l-keyword Q, the local context score of (d_i, v_j) will be exactly as the Eq. (6.10). If the upper bound score (Eq. (6.10)) is smaller than the best score of the currently computed local contexts (refer to Eq. (6.6)), then the next computed local context with the highest score will be taken from the currently maintained local contexts. For the second issue, when computing a local context containing (d_i, v_j), we find the nearest multi-attribute node that contains a certain keyword, which takes time when graph $\mathcal{G}$ is large in size. It is impractical to compute all shortest distances between a document and a multi-attribute node containing a keyword. Instead, for a keyword, w, we compute the shortest distance to every multi-attribute node, starting from those nodes that contain the keyword, w. The shortest distance is computed to the extent that it increases the score of the local context, or it can also be computed on demand. We only need to scan the graph G_R of graph $\mathcal{G}$, once for a keyword by starting from a virtual node that connects to all nodes containing the keyword.

The incremental algorithm is outlined in Algorithm 10. Here, D is the set of documents, assume that they are ordered in the decreasing upper bound score (Eq. (6.10)) (line 1). And the computed local contexts are maintained in a max-heap H (line 2). D and H will be initialized only on the first execution (line 1-2). When $H = \emptyset$, we assume that the top element in the heap $H.top()$ has a negative infinity score. Every time we get the next local context, if the top local context in H has a score greater than the upper bound score of the top document in D, then the top local context in H will be the next highest score local context. Otherwise, the top document in D needs to be inserted into H, and this step repeats until we ensure that the next highest score

Algorithm 10 Next(Q)

Input: a keyword query $Q = \{w_1, w_2, ..., w_l\}$.

Output: output the next high scored local context.

```
1: let $D$ be the list of documents, accessed in the decreasing of estimated score;
2: let $H$ be a max-heap that stores intermediate local contexts ($H \leftarrow \emptyset$ initially);
3: while not $D.empty()$ and $D.top() > H.top()$ do
4:     $d_i \leftarrow D.next()$;
5:     for each possible $(d_i, v_j)$ pair do
6:         compute the local context containing $(d_i, v_j)$ with the best score;
7:         insert the local context into $H$;
8:     end for
9: end while
10: return $H.next()$.
```

local context is the top result in H (line 4-7). We return the top local context in H as the next highest score local context (line 8).

6.7. Performance Studies

We have implemented six algorithms, based on different ranking semantics, to find the top-k documents with the help of a multi-attribute graph $\mathcal{G}$. The six algorithms are denoted as: *ERank*, *tkp*, *EScore*, *OptProb*, *OptScore*, and *DocOnly*. The first three, *ERank*, *tkp*, and *EScore*, are ranking based on the expected rank, top-k probability, and expected score as discussed in Section 6.5, respectively. *OptProb* and *OptScore* are two heuristics to choose the score of a document as the score of the local context in the corresponding supporting set (x-tuple) with the largest probability and score, respectively. *DocOnly* is to rank the documents based on the IR-score only, without a multi-attribute graph. All the algorithms were written in Visual C++ 2003. We conducted all the experiments on a 2.8GHz CPU and 2GB memory PC running XP.

For testing the algorithms, we use the real datasets, for both documents and multi-attribute graph. For the documents, we use the movie review dataset, and we crawled $330,201$ reviews about different movies from the Amazon website (`http://www.amazon.com`). For the multi-attribute graph, we use an IMDB movie database (`http://www.imdb.com/interfaces`). The multi-attribute graph is created in the same way as used in the literature of keyword search in RDB [74]. That is, we create a node for each tuple in the database, and there is a directed edge from node u to v if and only if there is a foreign key reference from u to v. In our experiments, we use 6 tables from IMDB database using the follow schema: Movie(<u>Mid</u>, Name), Genre(Mid, Genre), Director(<u>Did</u>, Name), Direct(Mid, Did), Actor(<u>Aid</u>, Name), and Play(Aid, Mid, Character), where primary keys are underlined. The numbers of tuples of the 6 tables are: $110K$, $148K$, $62K$, $113K$, $577K$, and $1410K$, respectively. In the resulting graph G_R, there are $2,423,262$ nodes and $6,394,016$ edges. The similarities between the matchings of movies in IMDB and movie reviews are calculated using the methods given in [76].

We evaluate the ranking accuracy of our algorithms using a (discounted) cumulative gain measure [52] based on human judgements. The (discounted) cumulative gain measure is widely used in measuring ranking accuracy of top-k queries [17, 37, 31]. The other measures, like precision, recall, and mean average precision, are inadequate in our problem setting for the following reasons. First, these measures assume that each document is either "definitely relevant" or "definitely irrelevant" to a keyword query, while a document can be relevant to some degree. Second, they need to know or retrieval all the relevant documents, while the number of relevant documents can be very large, and on the other hand, users are only interested in a few highly relevant documents.

In order to calculate the (discounted) cumulative gain, first, for a keyword query Q, a relevance value is assigned to each document. We use the relevance value between 0 and 4 (4 denoting high relevance, 0 denoting no relevance). Let r_i denote the relevance value of the document at the i-th position returned by a top-k query. The

Query	Keywords	Size
q_1	survive, murdered	2
q_2	marine, video	2
q_3	ducks, vista, disney	3
q_4	civil, war, peace	3
q_5	captain, excellent, music, sea	4
q_6	adventure, webster, fantastic, robert	4
q_7	michael, fan, freedom, range, endure	5
q_8	menagerie, star, hunter, west, river	5

Table 6.1: Keyword Queries

cumulative gain at rank position i, CG_i, is computed by summing the relevance value from position 1 to i.

$$CG_i = \begin{cases} r_i, & \text{if } i = 1 \\ CG_{i-1} + r_i, & \text{if } i > 1 \end{cases}$$

Discounted cumulative gain is defined similarly by incorporating rank-based discount factor. In our evaluation, we discount the relevance value of a document by the logarithm of its rank. The discounted cumulative gain at rank position i, DCG_i, is defined as below.

$$DCG_i = \begin{cases} r_i, & \text{if } i = 1 \\ DCG_{i-1} + r_i/\log_2 i, & \text{if } i > 1 \end{cases}$$

For the keywords, we pick up 8 keyword queries q_1 to q_8 with relevant keywords of size varying from 2 to 5. The keywords selected cover a large range of selectivity, as shown in Tab. 6.1. According to the keyword frequency in documents, the query in decreasing average keyword frequency order is q_7, q_5, q_8, q_2, q_4, q_1, q_6, q_3, where q_7 has the keyword frequency 0.04 and q_3 has the keyword frequency 0.007. Consider the multi-attribute graph, the query in decreasing average keyword frequency order is q_7, q_6, q_5, q_4, q_8, q_2, q_3, q_1, where q_7 has the keyword frequency 0.0009 and q_1 has the keyword frequency 0.00003. Specifically, the query q_6 has a low keyword frequency in the documents, but has a high keyword frequency in the multi-attribute graph. The

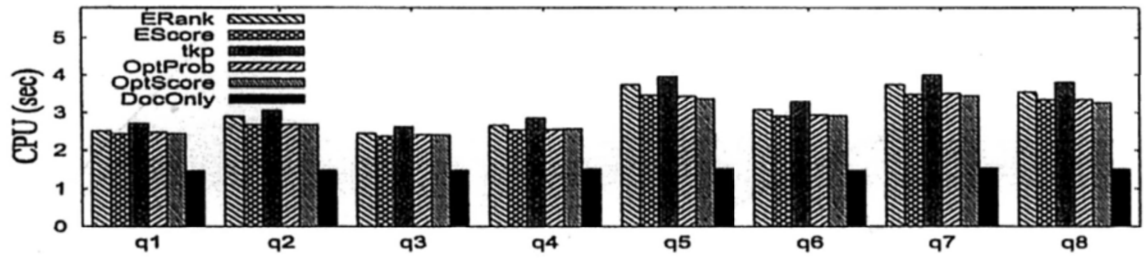


Figure 6.5: Execution Time

query q_7 has a high frequency in both components of the dataset, and the query q_4 has median keyword frequency in both components.

The execution time of top-30 query of keywords from q_1 to q_8 are shown in Fig. 6.5. As Expected, *DocOnly* takes the least amount of time, because it only uses the document data, while other algorithms use the additional multi-attribute graph which is complex than the plain text data. The other algorithms take almost the same amount of time. These algorithms need to compute the local contexts with best score, which is the dominating cost.

6.7.1. Case Study for Query q_5

We evaluate the ranking accuracy of the keyword query $q_5 = \{$"captain", "excellent", "music", "sea"$\}$ in Tab. 6.1. For this query, we expect to find some reviews regarding excellent musical movies which are about sea and a captain. Most of the algorithms find the most relevant movie review in a top-1 result, which is a review about the movie "The Bounty (1995)". We run all the six algorithms for a top-30 query, and obtain 30 documents for each algorithm. Then, we assign relevance values (0-4) to all these documents. Also, top-10 documents are selected in the rank order.

Based on the top-10 documents picked, we get a Precision-Recall style figure for each algorithm as shown in Fig. 6.6. We report the least number of results needed to be returned by the algorithms to contain the top-k documents, for k ranged from 1 to 10. All the algorithms report correctly the top-1 document, except the *OptProb* algorithm.

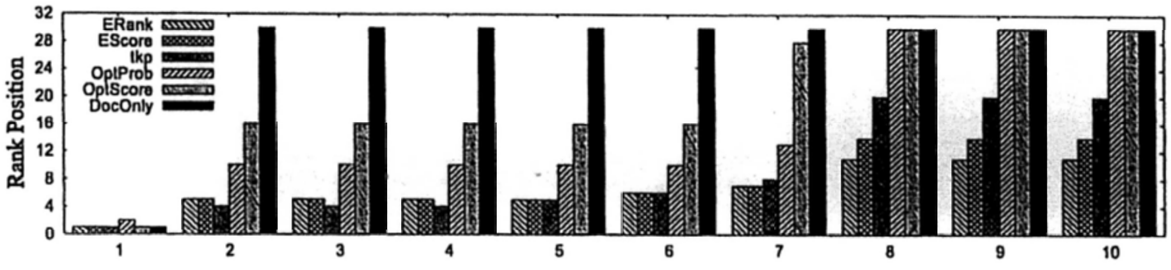
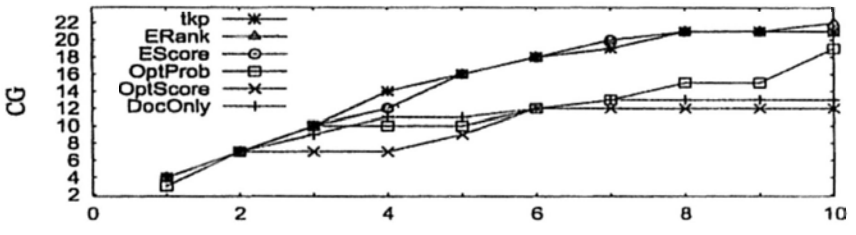
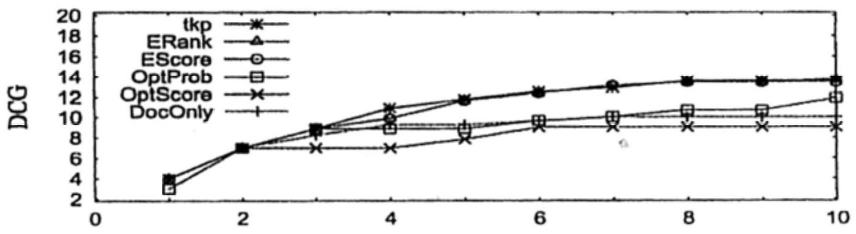


Figure 6.6: Least Rank Position to Include the Top-k (1 to 10) Documents



(a) Cumulative Gain

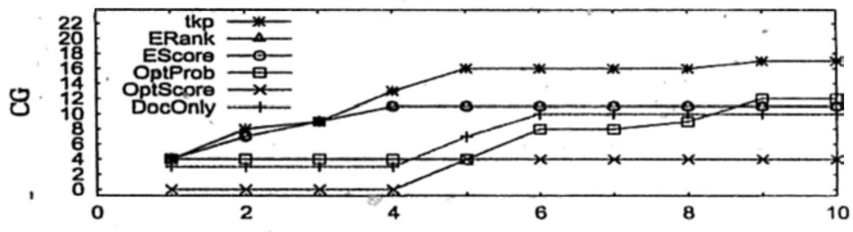


(b) Discounted Cumulative Gain

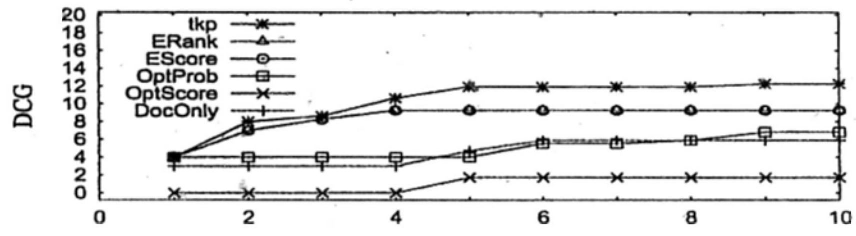
Figure 6.7: (Discounted) Cumulative Gain for q_5

Because the top-1 document includes all the keywords of q_5, and the multi-attribute graph do not provide any informative information of this document for this keyword query. Overall, the uncertainty aware ranking algorithms (*ERank*, *tkp*, *EScore*) perform better than other heuristic algorithms.

Fig. 6.7 shows the cumulative gain and discounted cumulative gain for the top-10 documents returned by these algorithms. Different from Fig. 6.6, the three algorithms: *ERank*, *tkp*, *EScore* have very similar cumulative and discounted cumulative gains, because in the cumulative gain measure different documents with the same relevance



(a) Cumulative Gain



(b) Discounted Cumulative Gain

Figure 6.8: (Discounted) Cumulative Gain for q_6

value are indistinguishable. Although the ranking between documents with the same relevance value are different for the algorithms, they result into the same (discounted) cumulative gain. Consistently, the uncertainty aware algorithms $ERank$, tkp, $EScore$ perform better than the other algorithms.

6.7.2. Case Study for Query q_6

We evaluate the ranking accuracy of the keyword query $q_6 = \{$"adventure", "webster", "fantastic", "robert"$\}$ as shown in Tab. 6.1. This query intents to find the reviews for fantastic movies, either the genre of the movie is adventure or the review categorizes it as an adventure movie, and "webster" and "robert" are either character names or actor/director names. We run all the six algorithms for top-30 query, and obtain the 30 documents for each algorithm. We assign the relevance values (0-4) to all these documents. One different characteristic of q_6 from q_5 is that, while the relevant document for q_5 contains a lot of keywords in q_5, the keywords of q_6 mostly appear in the multi-attribute graph. Only very few long documents contains some (not all) keywords in

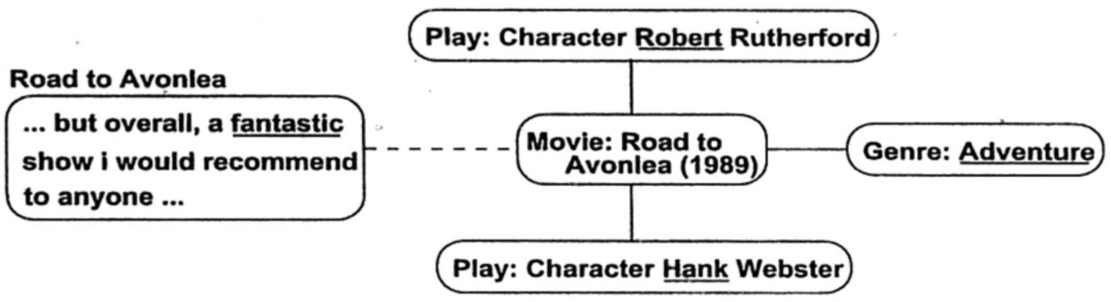
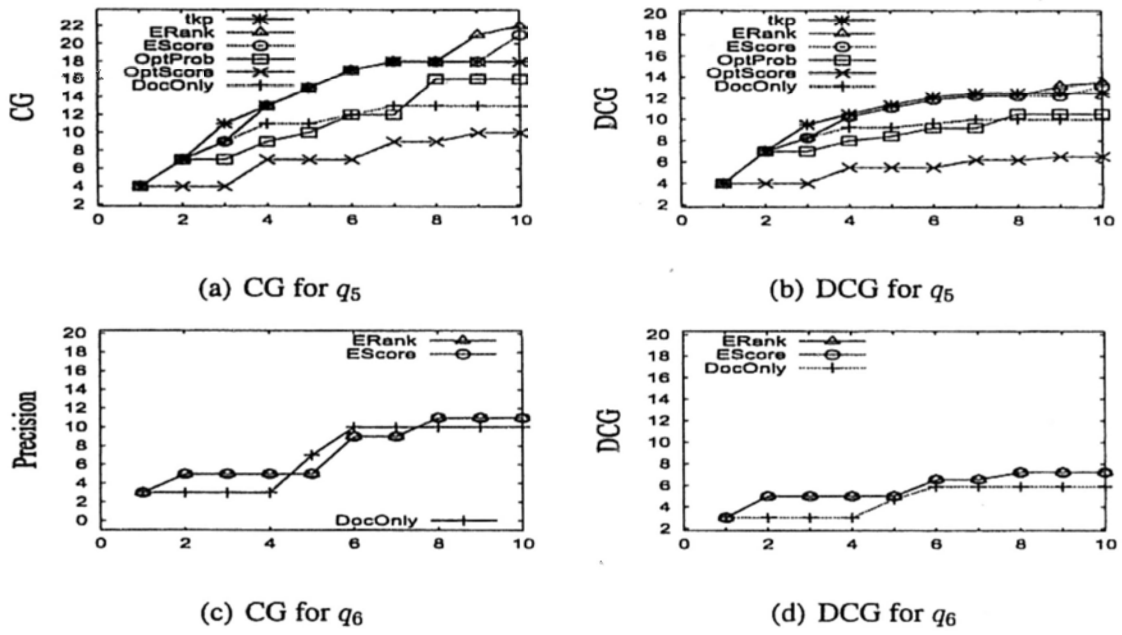

Play: Character Robert Rutherford
Road to Avonlea
... but overall, a fantastic show i would recommend to anyone ...
Movie: Road to Avonlea (1989)
Genre: Adventure
Play: Character Hank Webster

Figure 6.9: Top-1 Document with Context

q_6, and other documents only contain one out of the four keywords.

Fig. 6.8 shows the cumulative gain and discounted cumulative gain for top-10 documents returned by these algorithms. Different from Fig. 6.7, *tkp* performs better than *ERank* and *EScore*, because most part of the score of a local context comes from the multi-attribute graph for q_6. Note that, although the multi-attribute graph will increase the structure cost, it also increases the IR-score and completeness factor. The probability of a local context has more impacts on the documents ranking. The uncertainty aware algorithms *ERank*, *tkp*, and *EScore* perform better than other algorithms. The *OptScore* algorithm works badly in the scenario of q_6 such that only one document is relevant in the top-10 results.

Fig. 6.9 shows the top-1 result returned by *ERank*, *tkp*, and *EScore*. In this case, the document contains only one keyword "fantastic". The multi-attribute graph provides other keywords for the documents, and the relationships between these keywords and the document to be ranked is very tight. This top-1 result can not be found by *DocOnly*, as it contains only one keyword. *OptProb* and *OptScore* can not find this top-1 result either, because there exist other local contexts with a larger score but lower probability.



(a) CG for q_5 (b) DCG for q_5

(c) CG for q_6 (d) DCG for q_6

Figure 6.10: (Discounted) Cumulative Gain for q_5 and q_6

6.7.3. Multi-attribute Graph

We also test the influence of the multi-attribute graph to the rank accuracy of keyword queries. We create another multi-attribute graph, G'_R, by removing the table Genre(Mid, Genre) from IMDB, where the keyword "music" used in the query q_5 and "adventure" used in the query q_6 appear in this table. Fig. 6.10 shows the (discounted) cumulative gains for q_5 and q_6 respectively using the multi-attribute graph G'_R. As observed, with less information of the multi-attribute graph, for the query q_5, CG_8 decreases from 21 to 18, for the query q_6, the performance degrades much fast than that of q_5. For the query q_6, DCG_{10} decreases from 9.2 to 7.2, where it only decreases 0.2 for the query q_5. Another point is that *tkp*, *OptProb*, *OptScore* are more sensitive to the multi-attribute graph, because for the query q_6, those three algorithms return very few relevant answers.

CHAPTER 7

THESIS CONCLUSION

In this thesis, we explored several issues of uncertain data management. First, we proposed a novel linear time algorithm to compute the positional probability. Based on the positional probability, we also proved a tight upper bound of the top-k probability of tuples, which is then used to stop the top-k computation earlier. Then, we studied top-k probabilistic ranking queries with joins when scores and probabilities are stored in different relations. We investigated two probabilistic score functions, and gave upper/lower bounds of such probabilistic score functions in random access and sequential access. We also proposed new I/O efficient algorithms to find top-k objects. Third, we extended the possible worlds semantics to probabilistic XML rank query. We proposed a new dynamic programming algorithm which can compute top-k probabilities for the answers of node queries based on the previously computed results in probabilistic XML data. We also shown techniques to support path queries and tree queries. Then, we studied context sensitive document ranking, which ranks documents using a set of keywords given a context that is associated with the documents. We model the problem using a graph with two different kinds of nodes (document nodes and multi-attribute nodes), where the edges between document nodes and multi-attribute nodes exist with some probability. We discussed several probabilistic top-k definitions, and proposed new algorithms to rank documents that are most related to the user-given keywords by integrating the context information.

7.1. Future Directions

While we have addressed some computational questions that arise in uncertain data management, this hardly ends the quest for the holy grail of uncertain data ranking and management. Rather, it opens up some natural directions for future research.

First, in Chapter. 4, we have studied probabilistic ranking over relations. We proposed several lower/upper bounds of the ranking functions, which can be used to terminate the top-k computation very efficiently. We also find that, the hybrid method performs much better than the traditional sorted access or sequential access in terms of IO. Among all the existing main memory algorithms, they access the tuples in decreasing score order, which is similar to our sequential access method. It would be interesting to see the performance of our hybrid algorithm on a main memory resident single relation uncertain data.

Another direction is to study uncertain ranking with selection predicates. All the existing work study snapshot uncertain ranking where all the underlying data are considered. However, usually, users are only interested in subsets of the underlying data by issuing a ranking query with selection predicate. For the future work, we can study how to get the top-k results efficiently when a selection predicate is provided. A naive solution will be that, we first select the relevant tuples to form an uncertain database, and then run any existing algorithm ontop of it. But this will be time consuming, since different users have different selection predicates.

All the existing work only consider static data, they answer queries from scratch without indices. It would be interesting to study continuous top-k query, where insertions and/or deletions are allowed, or updates of the score and/or probability are issued. While answer the query starting from scratch is not efficient, indices or synopses can be build to maintain the ranking structure efficiently.

Besides the ranking issues of probabilistic data, more and more work start to extend their problems to the scenario of uncertain data, e.g. skyline on uncertain data, nearest neighbor and reverse nearest neighbor in uncertain data, finding heavy hitters and quantiles in uncertain data, finding frequent itemsets from uncertain data. It would be interest to study other database queries or data mining tasks in the scenario of uncertain data.

There is an intrinsic connection between privacy preserving data publishing and probabilistic databases. Generally, the data published after privacy preserving encoding represents a set of possible worlds of the original data. One direction of future works would be to represent the anonymized data in probabilistic database and answer queries on the anonymized data using the possible worlds based semantics, or using the existing works in probabilistic database literature to guide the anonymization process.

BIBLIOGRAPHY

[1] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proc. of SIGMOD'08*, 2008.

[2] Serge Abiteboul and Pierre Senellart. Querying and updating probabilistic information in xml. In *Proc. of EDBT'06*, 2006.

[3] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *Proc. of VLDB'06*, 2006.

[4] Parag Agrawal and Jennifer Widom. Confidence-aware join algorithms. In *Proc. of ICDE'09*, 2009.

[5] Sihem Amer-Yahia, Nick Koudas, Amélie Marian, Divesh Srivastava, and David Toman. Structure and content scoring for xml. In *Proc. of VLDB'05*, 2005.

[6] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proc. of VLDB'02*, 2002.

[7] Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data. In *Proc. of ICDE'08*, 2008.

[8] Lyublena Antova, Christoph Koch, and Dan Olteanu. From complete to incomplete information and back. In *Proc. of SIGMOD'07*, 2007.

[9] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1st edition, May 1999.

[10] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proc. of WWW'07*, 2007.

[11] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. of VLDB'06*, 2006.

[12] George Beskales, Mohamed A. Soliman, and Ihab F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *PVLDB*, 1(1), 2008.

[13] George Beskales, Mohamed A. Soliman, Ihab F. Ilyas, and Shai Ben-David. Modeling and querying possible repairs in duplicate detection. *PVLDB*, 2(1):598–609, 2009.

[14] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of ICDE'02*, 2002.

[15] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic twig joins: optimal XML pattern matching. In *Proc. of SIGMOD'02*, pages 310–321, 2002.

[16] Douglas Burdick, Prasad M. Deshpande, T. S. Jayram, Raghu Ramakrishnan, and Shivakumar Vaithyanathan. OLAP over uncertain and imprecise data. *VLDB J.*, 16(1), 2007.

[17] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. Learning to rank using gradient descent. In *Proc. of ICML'05*, 2005.

[18] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. Context-sensitive document ranking. In *Proc. of CIKM'09*, 2009.

[19] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. Fast probabilistic ranking under x-relation model. *CoRR*, abs/0906.4927, 2009.

[20] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. Query ranking in probabilistic xml data. In *Proc. of EDBT'09*, 2009.

[21] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. Context-sensitive document ranking. *J. Comput. Sci. Technol.*, 25(3):444–457, 2010.

[22] Lijun Chang, Jeffrey Xu Yu, Lu Qin, and Xuemin Lin. Probabilistic ranking over relations. In *Proc. of EDBT'10*, 2010.

[23] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of SIGMOD'03*, pages 551–562, 2003.

[24] Reynold Cheng, Sarvjeet Singh, Sunil Prabhakar, Rahul Shah, Jeffrey Scott Vitter, and Yuni Xia. Efficient join processing over uncertain data. In *Proc. of CIKM'06*, 2006.

[25] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Incorporating constraints in probabilistic xml. In *Proc. of PODS'08*, 2008.

[26] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. Xsearch: A semantic search engine for xml. In *Proc. of VLDB'03*, 2003.

[27] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proc. of IIWeb'03*, 2003.

[28] Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *Proc. of ICDE'09*, 2009.

[29] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4), 2007.

[30] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding top-k min-cost connected trees in databases. In *Proc. of ICDE'07*, 2007.

[31] Zhicheng Dou, Ruihua Song, Xiaojie Yuan, and Ji-Rong Wen. Are click-through data adequate for learning web search rankings? In *Proc. of CIKM'08*, pages 73–82, 2008.

[32] Ronald Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.

[33] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.

[34] Norbert Fuhr and Kai Großjohann. Xirql: A query language for information retrieval in xml documents. In *Proc. of SIGIR'01*, 2001.

[35] Tingjian Ge. Join queries on uncertain data: Semantics and efficient processing. In *Proc. of ICDE'11*, pages 697–708, 2011.

[36] Tingjian Ge, Stanley B. Zdonik, and Samuel Madden. Top- queries on uncertain data: on score distribution and typical answers. In *Proc. of SIGMOD'09*, pages 375–388, 2009.

[37] Xiubo Geng, Tie-Yan Liu, Tao Qin, Andrew Arnold, Hang Li, and Heung-Yeung Shum. Query dependent ranking using k-nearest neighbor. In *Proc. of SIGIR'08*, 2008.

[38] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *Proc. of VLDB'01*, 2001.

[39] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Towards efficient multi-feature queries in heterogeneous environments. In *Proc. of ITCC'01*, 2001.

[40] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *Proc. of SIGMOD'03*, 2003.

[41] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. Blinks: ranked keyword searches on graphs. In *Proc. of SIGMOD'07*, 2007.

[42] Mauricio A. Hernandez and Salvatore J. Stolfo. The merge/purge problem for large databases. In Michael J. Carey and Donovan A. Schneider, editors, *Proc. of SIGMOD'95*, 1995.

[43] Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *Proc. of VLDB'02*, 2002.

[44] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *Proc. of ICDE'08*, 2008.

[45] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. of SIGMOD'08*, 2008.

[46] Edward Hung, Lise Getoor, and V. S. Subrahmanian. Probabilistic interval xml. In *Proc. of ICDT'03*, 2003.

[47] Edward Hung, Lise Getoor, and V. S. Subrahmanian. Pxml: A probabilistic semistructured data model and algebra. In *Proc. of ICDE'03*, 2003.

[48] Stratos Idreos, Martin Kersten, and Stefan Manegold. Self-organizing tuple reconstruction in column-stores. In *Proc. of SIGMOD'09*, 2009.

[49] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3), 2004.

[50] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[51] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, and Peter J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *Proc. of SIGMOD'08*, pages 687–700, 2008.

[52] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proc. of SIGIR'00*, 2000.

[53] Cheqing Jin, Ke Yi, Lei Chen, Jeffrey Xu Yu, and Xuemin Lin. Sliding-window top-k queries on unceratin streams. In *Proc. of VLDB'08*, 2008.

[54] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. of VLDB'05*, 2005.

[55] Benny Kimelfeld, Yuri Kosharovsky, and Yehoshua Sagiv. Query efficiency in probabilistic xml models. In *Proc. of SIGMOD'08*, 2008.

[56] Benny Kimelfeld and Yehoshua Sagiv. Twig patterns: From xml trees to graphs. In *Proc. of WebDB'06*, 2006.

[57] Benny Kimelfeld and Yehoshua Sagiv. Matching twigs in probabilistic xml. In *Proc. of VLDB'07*, pages 27–38, 2007.

[58] David C. Lay. *Linear Algebra and Its Applications (3rd Edition)*. Addison Wesley, July 2002.

[59] Feifei Li, Ke Yi, and Jeffrey Jestes. Ranking distributed probabilistic data. In *Proc. of SIGMOD'09*, 2009.

[60] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proc. of SIGMOD'08*, 2008.

[61] Jian Li, Barna Saha, and Amol Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[62] Jianxin Li, Chengfei Liu, Rui Zhou, and Wei Wang. Top-k keyword search over probabilistic xml data. In *Proc. of ICDE'11*, pages 673–684, 2011.

[63] Wen-Syan Li, K. Selçuk Candan, Quoc Vu, and Divyakant Agrawal. Retrieving and organizing web pages by "information unit". In *Proc. of WWW'01*, 2001.

[64] Xiang Lian and Lei Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *Proc. of SIGMOD'08*, 2008.

[65] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: top-k keyword query in relational databases. In *Proc. of SIGMOD'07*, 2007.

[66] Nikos Mamoulis, Man Lung Yiu, Kit Hung Cheng, and David W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Trans. Database Syst.*, 32(3):19, 2007.

[67] Amélie Marian, Nicolas Bruno, and Luis Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2), 2004.

[68] Eirinaios Michelakis, Rajasekar Krishnamurthy, Peter J. Haas, and Shivakumar Vaithyanathan. Uncertainty management in rule-based information extraction systems. In *Proc. of SIGMOD'09*, pages 101–114, 2009.

[69] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *Proc. of VLDB'01*, 2001.

[70] Andrew Nierman and H. V. Jagadish. Protdb: Probabilistic data in xml. In *Proc. of VLDB'02*, 2002.

[71] Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *Proc. of ICDE'09*, pages 640–651, 2009.

[72] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic skylines on uncertain data. In *Proc. of VLDB'07*, 2007.

[73] Hasso Plattner. A common database approach for oltp and olap using an in-memory column database. In *SIGMOD'09*, pages 1–2, 2009.

[74] Lu Qin, Jeffrey Xu Yu, Lijun Chang, and Yufei Tao. Querying communities in relational databases. In *Proc. of ICDE'09*, 2009.

[75] Lu Qin, Jeffrey Xu Yu, and Bolin Ding. *TwigList* : Make twig pattern matching fast. In *Proc. of DASFAA'07*, 2007.

[76] Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. of ICDE'07*, 2007.

[77] Christopher Re and Dan Suciu. Managing probabilistic data with mystiq: The can-do, the could-do, and the can't-do. In *Proc. of SUM'08*, 2008.

[78] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan, and Luis Gravano. Efficient keyword search across heterogeneous relational databases. In *Proc. of ICDE'07*, 2007.

[79] Torsten Schlieder and Holger Meuss. Querying and ranking xml documents. *Proc. of JASIST'02*, 53(6), 2002.

[80] Karl Schnaitter and Neoklis Polyzotis. Evaluating rank joins with optimal cost. In *Proc. of PODS'08*, 2008.

[81] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Prdb: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

[82] Pierre Senellart and Serge Abiteboul. On the complexity of managing probabilistic xml data. In *PODS*, 2007.

[83] Michal Shmueli-Scheuer, Chen Li, Yosi Mass, Haggai Roitman, Ralf Schenkel, and Gerhard Weikum. Best-effort top-k query processing under budgetary constraints. In *Proc. of ICDE'09*, 2009.

[84] Sarvjeet Singh, Chris Mayfield, Sunil Prabhakar, Rahul Shah, and Susanne E. Hambrusch. Indexing uncertain categorical data. In *Proc. of ICDE'07*, pages 616–625, 2007.

[85] Mohamed A. Soliman and Ihab F. Ilyas. Ranking with uncertain scores. In *Proc. of ICDE'09*, pages 317–328, 2009.

[86] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE'07*, 2007.

[87] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Probabilistic top- and ranking-aggregate queries. *ACM Trans. Database Syst.*, 33(3), 2008.

[88] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik. C-store: A column-oriented dbms. In *Proc. of VLDB'05*, pages 553–564, 2005.

[89] Esko Ukkonen. On approximate string matching. In *FCT*, 1983.

[90] Maurice van Keulen, Ander de Keijzer, and Wouter Alink. A probabilistic xml approach to data integration. In *Proc. of ICDE'05*, 2005.

[91] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *Proc. of WWW'08*, 2008.

[92] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. Efficient processing of top-k queries in uncertain databases. In *Proc. of ICDE'08*, 2008.

[93] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. Efficient processing of top-k queries in uncertain databases with x-Relations. *IEEE Trans. Knowl. Data Eng.*, 20(12), 2008.

[94] Xi Zhang and Jan Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *Proc. of DBRank'08*, 2008.