The Hong Kong Polytechnic University

Department of Computing

CLEANING AND QUERYING LARGE UNCERTAIN DATABASES

by

JINCHUAN CHEN

A thesis submitted in partial fulfillment

of the requirements

for the Degree of Doctor of Philosophy

December 2008

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signed)

CHEN JINCHUAN (Name of student)

Abstract

The management of uncertain databases has recently attracted tremendous interest from both industry and academy communities. In particular, there is a need to handle uncertain data in many emerging applications, such as the wireless sensor network, biometric and biological databases, location-based services, and data stream applications. To obtain meaningful results over these uncertain data, probabilistic queries are proposed, which augment query results with confidence. Although probabilistic queries are useful, evaluating them is costly, in terms of both I/O and computation. Moreover, the calculation of answer probabilities involves expensive numerical integrations. Therefore the efficient evaluation of probabilistic queries is a challenge for uncertain database management. In this thesis, we report our works for speeding up the evaluation performance of three kinds of important probabilistic queries – nearest-neighbor queries, k-nearest-neighbor queries, and imprecise location-dependent queries. New approaches are proposed to improve the efficiency in both I/O and computation, and they are evaluated by extensive simulations over real and synthetic data sets.

Another important issue that we consider in this thesis is the cleaning of uncertain data with the goal of achieving higher quality. Since the applications handling imprecise data have resource limitation, the cleaning process must optimize the use of resources. We study theoretically and experimentally on how the result quality could be maximized with constrained resources, with the use of entropy-based metrics. We also outline the future directions of our work.

Publications arising from the thesis

1. **J. Chen** and R. Cheng. Efficient evaluation of imprecise location-dependent queries. In ICDE'07: Proceedings of the 23rd International Conference on Data Engineering, pages 586-595, 2007.

2. R. Cheng, **J. Chen**, M. F. Mokbel, and C.-Y. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In ICDE'08: Proceedings of the 24th International Conference on Data Engineering, pages 973-982, 2008.

3. **J. Chen** and R. Cheng. Quality-aware probing of uncertain data with resource constraints. In SSDBM'08:Proceedings of the 20th International Conference on Scientific and Statistical Database Management, pages 491-508, 2008.

4. R. Cheng, **J. Chen**, and X. Xie. Cleaning uncertain data with quality guarantees. In VLDB'08: Proceedings of the 34th International Conference on Very Large Data Bases, pages 722-735. VLDB Endowment, 2008.

5. R. Cheng, L. Chen, **J. Chen**, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In EDBT'09: the 12nd International Conference on Extending Database Technology, pages 672-683, 2009.

6. R. Cheng and **J. Chen**. Probabilistic Spatial Queries. In the Encyclopedia of Database Systems, L. Liu and T. Ozsu (eds.), ISBN 978-0387355443, Springer-Verlag, 2009.

7. **J.Chen**, R. Cheng, Y. Zhang and J. Jian. A Probabilistic Filter Pro-

tocol for Continuous Queries. To appear in QuaCon'09: First International Workshop on Quality of Context, 2009.

8. **J.Chen**, R.Cheng, M. F. Mokbel, and C.-Y. Chow. Scalable Processing of Snapshot and Continuous Nearest-Neighbor Queries over One-Dimensional Uncertain Data. Submitted to VLDBJ for consideration.

Acknowledgements

The work presented here is not possible without the help of many people, to whom I would like to express my deep appreciation.

I would like to give my sincere respect and thanks to my supervisor, Dr. Reynold C.K. Cheng. Dr. Cheng provides me the rare chance of admission. He has been more than available throughout my research, implementation, and writing of research papers. His input and direction was invaluable in the creation of this thesis, and the research in contains. Without his help, this thesis would not exist.

I would like to thank Prof. Jiannong Cao, who gives me countless help in my research. His guidance and constructive criticism benefited me a lot. His care, patience and strict working style impress me very much. Appreciation also goes to Dr. Mohamed F. Mokbel and Dr. Lei Chen, for their guidance and suggestions.

Words are never enough to express my deep love and thanks to my parents who give me life and encourage me to face difficulties on my way of growing up. Gratefulness and love also go to my dear wife, Zhao Na, whom I owe so much to, for her unwavering supporting, understanding and encouragement.

The financial support given to my Ph.D. study by the Hong Kong Polytechnic University is also gratefully acknowledged. Last but not least, I would like to thank my friends in Hong Kong, Xike Xie, Yirong Chen, and Fei Dong etc., who create a harmonious atmosphere around me and share a happy time with me.

Contents

List of Tables

List of Figures

1 Introduction

Data uncertainty is an intrinsic property for many emerging applications. This presents a significant challenge for traditional database management systems (DBMS) where data values are usually assumed to be precise and exact. The topics of effective and efficient management of uncertain data attract attentions from both academic and industry communities in recent years. In this thesis, we focus on query processing and quality-aware data cleaning aspects of uncertain databases. Section 1.1 first briefly introduces the uncertain data model we used. We then summarize the concept of probabilistic queries over uncertain databases in Section 1.2. In Section 1.3 we present the problem of cleaning uncertain databases. Section 1.4 illustrates the organization of this thesis.

1.1 Uncertain Data Models

Traditional database systems assume data values to be precise and correct. However, in many arising applications, the data is intrinsically uncertain. Consider a monitoring system which employs a wireless sensor network to obtain readings from external environments. Due to the imperfection of physical devices, as well as limited battery power and network delay, it is impossible to obtain an accurate data reading at every point in time. As a result, the data readings acquired can be contaminated with errors [23, 33]. As another example, a location-based service (LBS) answers queries over moving objects whose locations are collected by the Global Positioning System (GPS). These location data are inherently imprecise due to measurement error, sampling error and network latency [87, 24]. Moreover, some people may want to blur their location data in order to protect their privacy [8].

These factors of "uncertainty" need to be considered carefully, or else the query results can be incorrect.

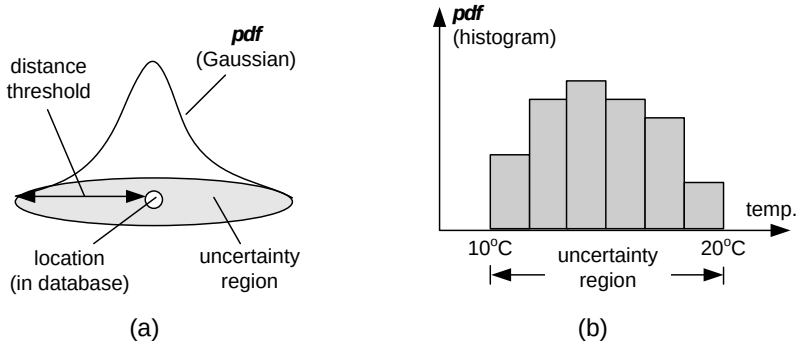


Figure 1.1: Uncertainty of (a) location and (b) sensor value.

A well-studied uncertainty model, called *attribute-uncertainty*, assumes that the actual data value is located within a closed region, called the *uncertainty region*. In this region, a non-zero probability density function (pdf) of the value is defined, such that the integration of pdf inside the region is equal to 1. That is, the value of the pdf outside the uncertainty region is zero. In an LBS where the dead-reckoning approach is used, a normalized Gaussian distribution is used to model the measurement error of a location stored in a database [70, 87] (Figure 1.1(a)). The distribution parameters could be derived based on the application scenario like network reliability and transmission delay [87]. Gaussian distributions are also used to model values of a feature vector in biometric databases [12]. Figure 1.1(b) shows another example, which illustrates the histogram of temperature values in a geographical area observed in a week. The pdf, represented as a histogram, is an arbitrary distribution between 10^oC and 20^oC, which could be estimated by historical records [33]. Unless stated otherwise, in this thesis we will assume the attribute-uncertainty model. We will review other uncertainty models such as *tuple-uncertainty* and *incomplete information databases* in

Section 2.

1.2 Querying Uncertain Data

Since data are uncertain, querying on them may generate imprecise results. To represent this imprecision, the *probabilistic queries* extend query answers with probabilistic confidences. Nowadays, many kinds of probabilistic queries are proposed for different applications, including range query [87], nearest-neighbor [24, 55], skyline [69, 58], and Min/Max [23] etc. Particularly, in this thesis we focus on three important kinds of probabilistic queries as listed in the following.

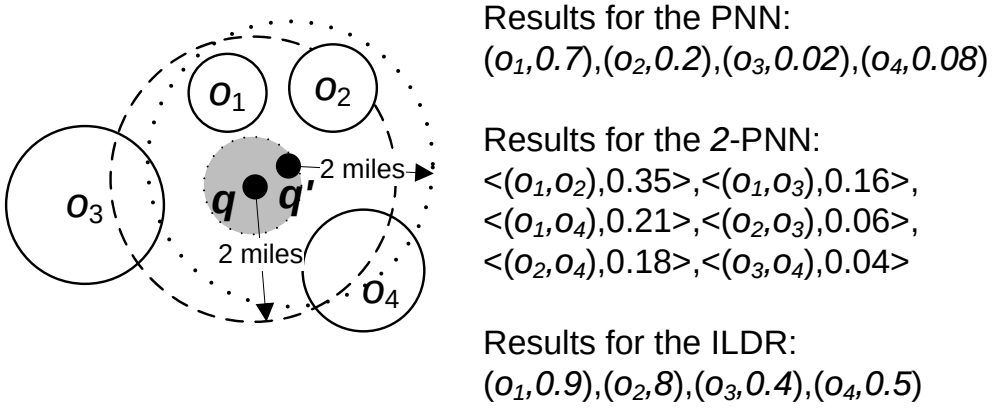


Figure 1.2: Examples of Probabilistic Queries.

- The *Probabilistic Nearest-Neighbor* (PNN) query returns the non-zero probability of each object for being the nearest-neighbor of a given query point.
- The *Probabilistic k-Nearest-Neighbor* (k-PNN) query returns the non-zero probability of each set of k objects for being the nearest-neighbor of a given query point.

- The *Imprecise Location-Dependent Range* (ILDR) query retrieves a set of objects and their probabilities for locating inside a fixed-size range centered at the query issuer's position.

Figure 1.2 illustrates the examples of a PNN, a 2-PNN and an ILDR. The solid-line circles denote the uncertainty regions of the objects $o_1 \ldots o_4$. The query point for the PNN and the 2-PNN is q. The query issuer of the ILDR asks for the objects within 2 miles of his/her position. Notice that the location of the query issuer is also imprecise and bounded by the shaded circle. As an example, Figure 1.2 only shows two possible locations, q and q', of the query issuer and the corresponding query ranges, i.e. the dashed and the dotted circles. In fact the query issuer's location could be each point inside the shaded circle. The results for these queries are listed on the side of Figure 1.2. The confidence values indicate the chances that the object(s) satisfy the queries, namely *qualification probabilities*. For instance, the answer $(o_1, 0.7)$ for the PNN query means that the probability that o_1 is the nearest-neighbor of q is 0.7.

Evaluating these probabilistic queries is not an easy task. Firstly, expensive numerical integration is required for computing qualification probabilities. Furthermore, the processing of ranking-based queries such as PNN and k-PNN needs to calculate integrals over complex functions, since the ordering relationships between the uncertain data objects will be incorporated into the probability calculation. For k-PNN queries, the size of their query results may be exponentially large, since each set of cardinality k is a possible answer, e.g. there are $C_2^4 = 6$ of result sets for the 2-PNN query in Figure 1.2. Finally, the evaluation of the imprecise location-dependent range query requires processing a range query for each possible location of the query issuer, greatly exacerbating the burden of the DBMS.

To address these challenges, in this thesis we discuss how to improve the evaluation efficiency for probabilistic queries. We observe that many query issuers may be not interested in the whole answers, and they may only need the answers with high confidences. In such case, we only need to return the answers whose qualification probabilities larger than a given threshold to the query issuer. With this observation, we thus propose efficient solutions for answering probabilistic queries with thresholds.

1.3 Cleaning Uncertain Data

As mentioned before, the queries over uncertain database may produce imprecise answers with probabilistic guarantees. The query issuers may require better answers, i.e. less ambiguity in the answers. Therefore, another important issue about the management of the uncertain data is to improve query quality, through the means of reducing the ambiguity of the database. The data ambiguity could be alleviated in many ways. As an example, in wireless sensor network, the sensor readings could be imprecise due to sampling error and/or network latency. The remote sensors could be "probed" in order to obtain their latest readings [87, 61, 33]. The cleaning actions need to cost some kind of resources, e.g. network bandwidth. It is therefore necessary to selectively clean some uncertain data in order to optimally utilize the resources. For this purpose, we need to know how good the query answer is, and how much the answer quality could be improved if some data are cleaned. We address these issues and propose optimal algorithms to obtain the best query answer by cleaning attribute-uncertainty data withing a given resource constraint. We also extend the techniques for data with *tuple-uncertainty*.

1.4 Organization of This Thesis

The rest of this thesis is organized as follows. We discuss the related works in Section 2. We then report our works for efficiently processing of the *probabilistic nearest-neighbor queries*, the *probabilistic k-nearest-neighbor queries*, and the *imprecise location-dependent queries* in Section 3, 4 and 5 respectively. In Section 6, we propose a set of optimal algorithms and heuristics for cleaning *attribute-uncertainty* data in order to maximize query result quality. We then extend these techniques for cleaning of uncertain data under *tuple-uncertainty* in Section 7. Section 8 concludes this thesis and outlines some possible future works.

2 Literature Review

We present a brief survey on the related works in this chapter. We discuss different models of the attribute-uncertainty in Section 2.1. Section 2.2 summarizes the works about the query evaluation over the uncertain data. We then discuss some other uncertainty models in Section 2.3. Finally, Section 2.4 introduces the works related to the cleaning of uncertain data.

2.1 Attribute-Uncertainty Models

Wolfson et al. utilize a dead-reckoning policy for bounding the imprecise locations of the moving objects [87]. A moving object periodically updates its location with the help of GPS, and it is aware of its location in the database. Whenever the distance between the actual location of a moving object and its database location exceeds a given threshold, a location update is reported to the database. An object may move on a predefined route, e.g. in a road. In this case this object's actual location is modeled as a route segment with the threshold as its half-width and the database position as its middle point. Rather than on routes, an object may also move freely in space, and its location is then bounded by a circle around the database location with the radius of the threshold. Besides the uncertainty region, a density function is generated to indicate the distribution of the moving object's location, based on the database location and the uncertainty. This uncertainty model is further extended to a $3D$ cylindrical body for representing the uncertain trajectories of the moving objects in [83]. At each point of time, the actual location of an object cannot deviate from its database location by a given threshold. Hence the possible trajectory of this object is bounded by a trajectory volume around its expected trajectory.

Pfoser et al. indicate two kinds of error sources, i.e. the measurement error and the sampling error, which bring uncertainty into the moving objects' locations [70]. The measurement error is introduced by the inaccurate measurements. A probability function such as Gaussian can be used to describe the measurement error. The sampling error comes from the fact that the database cannot constantly sample the locations of a moving object, introducing the uncertainty about the object's location in-between the sampling measurements. Based on the maximum speed of the object, its locations in-between two consecutive position samples could be bounded in an ellipse, called *error ellipse*, with the two sample points as its foci. They also compare the two kind of error sources and find that the measurement error is small compared to the sampling error.

Another important source of generating uncertain data is the wireless sensor network, where the sensor readings cached at the database are usually imprecise due to sampling error, measurement error and network delay. Olston et al. model the value of a sensor attribute as an interval, which is around its last record and has a width smaller than a given precision threshold [67]. An update will be sent to the database whenever the actual reading exceeds this interval. In order to achieve a good trade-off between precision and resource consumption, a dynamic scheme for shrinking and expanding the intervals is proposed. Later on, a probabilistic model is proposed for describing the imprecise sensor values in [33]. In this work, the actual value is regarded as a random variable. Historical records are utilized to estimate its distributions. Moreover, the correlations among different attributes are taken into account. For example, the sensor's temperature is probably high when its voltage is large. They build a multi-dimension Gaussian as the joint distribution for the imprecise values of multiple attributes of an object.

Based on this model, an optimal algorithm is proposed for selectively pulling the remote sensors to satisfy the query precision requirement with minimum resource cost.

Data uncertainty also appears in other applications like biometric [12, 62], and video clips [11]. Böhm et al. aim to solve the identification problems over the imprecise data which are represented by *probabilistic feature vectors*, where the conventional feature values are replaced by Gaussian probability distribution functions [12]. In [62], the authors notice some special distributions, e.g. mixture Gaussians, in the biomedical fields.

Although the uncertain data may be generated from different applications, and be captured with different mechanisms, their representation models generally contain two parts, i.e. an uncertainty region which bounds the imprecision, and a probability density function which provides the distribution information of the actual value, as we have discussed in Section 1.1. Unless otherwise announced, we assume in this thesis the database has the information about the imprecise data in the form of this attribute-uncertainty which could be generated with some existing techniques in former works.

In some applications, the distribution of an imprecise data is better represented as discrete samples. Kriegel et al. model an uncertain spatial object as a sample set [55]. Pei et al. regard the unstable performance of a basketball player as a group of game-by-game performance data [69]. In these two works, all samples are assumed to have equal chances to be the actual entity. Compared to the aforementioned uncertainty model, the models in [55, 69] adopt the discrete distribution instead of the continuous one.

2.2 Queries over Uncertain Database

The notion of probabilistic queries first appeared in [87, 70], where probabilistic confidence guarantees are appended to query answers. Cheng et al. propose a set of probabilistic queries, and develop algorithms for evaluating them [23]. There are also other attempts for the efficient evaluation of different kinds of probabilistic queries.

2.2.1 Probabilistic Nearest-Neighbor Query

Given a query point, the *Probabilistic Nearest-Neighbor Query* returns objects which are possible to be the closest one to the query point as well as their probabilities [20]. An R-tree-based solution for probabilistic nearest-neighbor queries has been presented in [24, 25]. The main idea is that objects with zero probabilities can be filtered, based on the fact that these objects' uncertainty regions must not overlap with an object's uncertainty region whose maximum distance from the query point is the smallest among all the objects. They also discuss the calculation of qualification probability values for objects that cannot be pruned by the R-tree. The main approach is to convert each uncertainty region to two functions: pdf and cdf (cumulative density function) of the distance of the object from the query point. They show how this conversion can be done for 1D space (intervals) and 2D space (circle and line uncertainty). The qualification probabilities are then derived by evaluating an integral of a complicated function of distance pdfs and cdfs. The cost of these solutions is quite high, since they require numerical integration over some aggregate functions of arbitrary pdfs. Our solution also employs the distance pdfs and cdfs. However, we avoid integration for most of the objects involved, thus saving significant amount of computation time.

Another method to evaluate the exact values of qualification probabili-

ties for probabilistic nearest neighbor queries is proposed in [55], where each uncertain object is represented as a set of sample points which are grouped in several clusters. The qualification probability of an object is obtained by summing up the chances that each of its member samples is the nearest-neighbor. The main idea is to avoid the unnecessary pair-comparisons among the sample points based on the fact that a cluster is not needed to be expanded if it must be further to the query point than a sample point of other objects. Compared with that work, our solution is tailored for a constrained version of the probabilistic nearest-neighbor queries, where threshold and tolerance conditions are used to avoid computation of exact probabilities. Also, we do not need the additional work of converting the pdf of each object to points first. Moreover, this conversion could introduce sampling error if there are not enough samples.

There are also some other related works concerning different variants of the probabilistic nearest-neighbor queries, such as the probabilistic group nearest neighbor query [59], the top-k probable nearest neighbors [10], and the probabilistic reverse nearest neighbor queries [17]. Recently, an indexing method is proposed for processing probabilistic nearest-neighbor threshold queries [72].

2.2.2 Probabilistic k-Nearest-Neighbor Query

The k-PNN [22] can be considered as a version of the k-nearest neighbor query (k-NN) evaluated on uncertain data. There are lots of attempts for answering k-NN queries [50, 39, 37, 54]. These works about k-NN queries assume that the data being queried is precise. On the contrast, our work addresses the k-PNN query over uncertain data.

To our best knowledge, few work has addressed k-NN queries over un-

certain data. Beskales et al. [10] proposed a *top-k* query that ranks the probability that each object is the nearest neighbor of q, and returns the k objects with the highest probabilities. Notice that the ranking criterion is based solely on each object's probability of being the nearest-neighbor of q. This is not the probability that all objects returned in the query answer are the k nearest neighbors of q (by considering all the possible cases). On the other hand, the query studied in this thesis is a "true" k-nearest-neighbor query, where we consider the probability that a set of objects are the k nearest neighbors of q. It expands the aggregated results returned by *top-k* queries and provides query issuer more information about the actual result. For example, suppose a user issues a 2-NN query over the uncertain objects in Figure 1.2. Following the semantic in [10], the result will be $\{o_1, o_2\}$ since these two objects have the highest qualification probabilities to be nearest-neighbor of q. However, according to Figure 1.2, $\{o_1, o_4\}$ also has high chances to be the real answer for this 2-NN query, and is missed in the result. Furthermore, although *top-k* query places more emphasis on the relative importance of the results, it misses the absolute probabilistic confidence levels. For instance, a *top-k* query may receive k objects each of which has a very low qualification probability, if the query point is located in a very dense area. The qualification probabilities of these k objects may be too low to be interesting for the query issuer, and computing them is a waste of resources. This problem could be easily solved in k-PNN, where a user specified probability threshold will avoid the cost of computing results with low qualification probabilities.

In [62], Ljosa et al. proposed an efficient index structure, called APLA-tree, for evaluating k-NN queries. They used the expected distance (under L1-norm) of an object's uncertainty pdf from q as a ranking criterion. Thus,

their k-NN query is based on the expected distance, and does not have probabilities in their answers. Different to these works, our goal is to investigate efficient methods for evaluating k-NN queries for uncertain databases. Compared to 1-PNN, i.e. PNN [20], the evaluation of k-PNN faces the additional problem of examining a large number of k-subsets. To handle this problem, we develop new methods to significantly reduce the number of candidate k-subsets.

2.2.3 Imprecise Location-Dependent Query

An important piece of work in this thesis is to efficiently answer the *imprecise location-dependent queries* [18]. A location-based service (LBS) is an information service which makes use of the geographical information to serve a mobile user. A typical example of LBS is the the E-911 system mandated by the U.S., which requires cell phone companies to provide an accurate location of a cell phone user who calls for emergency help [86]. The *Location-Dependent Queries* have been a subject of research interest in these few years. This kind of queries decide the content of query results depending on the location of the query issuer [57, 40]. In [57], a survey of research problems related to location-dependent information services has been presented. The problem of managing location queries in a distributed manner has been studied in [40], where the MobiEyes system is developed to answer queries efficiently. In [47], the authors define location-dependent query operators so that more complex queries can be constructed. They also study how mobile agents can be used to support distributed query processing.

As mentioned in Section 1.1, the location data is intrinsically imprecise. For *imprecise location-dependent queries*, the location of the query issuer can also be uncertain, which may affect the validity of the query answer. Song

et al. [78] study the evaluation of a a continuous nearest-neighbor query for retrieving the nearest neighbors for all points on a line segment. Their algorithm is further improved in [80]. In [46], the range nearest-neighbor query is proposed, which retrieves the nearest neighbor for every point in a multi-dimensional rectangular range. In these works, although the query issuer's location is imprecise, the data being queried has no uncertainty. Furthermore, they do not consider the problem of computing qualification probabilities.

To the best of our knowledge, none of the previous work address the issue of providing probability guarantees for imprecise location-dependent queries. In [27], the authors study the trade-off of location privacy, service quality, and the uncertainty of location-dependent range queries. A service quality metric based on the objects' qualification probabilities is proposed. In this thesis we address the efficiency issues of evaluating this type of queries.

Particularly, we focus on processing probabilistic range queries depending on imprecise location of the query issuer, i.e. imprecise location-dependent range queries. The *probabilistic range queries* return objects whose values fall into a predefined query window and their qualification probabilities [87, 23, 70]. An indexing method, called *PTI*, computes and stores histograms of data objects in the index in order to estimate the upper bounds of the result qualification probabilities with respect to a probabilistic range query [26]. Once the upper bounds are smaller than a predefined threshold, the corresponding data objects can be pruned without being loaded from hard disks. In consequence, many I/O and computation cost could be saved. This indexing technique is extended for handling the objects in multi-dimensions, with a compact structure to improve the storage efficiency [79, 81]. In our work, *PTI* is utilized to improve the performance of evaluating the imprecise

location-dependent range queries.

2.3 Other Uncertainty Models

Besides the attribute-uncertainty, there is another popular model for representing uncertain data, i.e. the *tuple-uncertainty*, which is also called *probabilistic database*. Cavallo et al. propose the notion of probabilistic database [14], where each tuple is attached with a probability to indicate the chances it is in the relation, and all such tuple probabilities in a relation should sum up to one. This model requires all tuples in a relation to be disjoint events. Barbará et al. extend this model by replacing the general constraint in a relation with local constraints, i.e. only the tuples belonging to the same physical entity should be disjoint events [6]. This model is followed by many works [38, 3, 30, 21]. Recently, efficient query algorithms for uncertainty of categorical data are studied [75]. In [7], the ULDB model is presented, which combines the properties of probabilistic and lineage databases. In Section 7, we will discuss how to clean tuple-uncertainty data in order to achieve best result quality.

The *incomplete information database* attracts many attentions in the last two decades, which comes from the needs for handling the *null* values. This kind of values provide no information except the attribute domain. There are many attempts to return semantically correct response to the queries over tables containing null values [44, 15, 28, 85, 88, 48, 43, 49]. There are two representing systems for incomplete information database. The first one is proposed by Codd [28], where all null values are marked by a single symbol, and each null value is a variable within its domain. The *V-table* concerns the correlations among multiple tuples, which requires the null values be denoted by multiple symbols, and the null values sharing the same symbol stand for

the same variable [15]. The *V-table* is also extended to the *conditioned table* which includes an additional column to further prescribe the conditions the variables should follow [48, 2].

Different from the probabilistic queries, the queries over incomplete information database are returned with precise results. This is obtained by extending the relational operators. In [28], a three-valued logic is utilized to process the operators, e.g. the null value is not equal to any non-null value. Imieliński and Lipski propose a condition which should be satisfied so that the results for a query over incomplete information system can be semantically correct [48]. The condition is to ensure that the answer for the query should contain enough information to decide all tuples that must be inside the results of evaluating the same query over all possible database instances. A recent work proposes an extended language, the I-SQL, on incomplete information, to support the "what if" queries [5].

There is also another representation model for the uncertain data, i.e. the *fuzzy databases*. It is based on the fuzzy set theory. Instead of the probabilities, it collects and stores the *possibility distribution* of an uncertain data [13, 71, 89]. Note that the values of a possibility distribution do not have to sum up to one.

2.4 Cleaning Uncertain Databases

In applications like sensor network monitoring, it is important for a system to generate a probing plan that only requests relevant sources to report their data values, in order to optimize the use of resources. In [68], efficient algorithms are derived to fetch remote data items in order to generate a satisfactory result quickly. Liu et al. [61] propose an optimal algorithm to find the exact result for minimum and maximum queries by probing the smallest set

of data sources. The uncertainty model of a data item considered in these two work is simply a one-dimensional interval. Since the pdf of the value within the interval is not considered, the query results are "qualitative", i.e. they are not be augmented with probabilistic guarantees. Our work, on the other hand, defines quality metrics for probabilistic query answers, and use the measures to devise probing sets. Although Madden et al. [33] consider the pdf of data values in their uncertainty models, their methods do not consider the strict resource constraints imposed on the system (e.g., the maximum amount of resources that can be spent on a query). The quality metric they consider is based on a simple probability threshold (e.g., the probability of the object should be higher than 95%). Our work aims at achieving the highest quality under limited resource constraints. Our solutions can be applied to multi-dimensional uncertain data with artibtrary pdfs.

Previously, a number of quality measures have been studied. In [33, 26], if the qualification probability of a result is higher than a user-defined threshold, then the query result is considered to be satisfactory. In [74], the quality of a top-k query is given by the fraction of the true top-k values contained in the query results. In [23], different metrics are defined for range queries, nearest-neighbor queries, `AVG` and `SUM` queries. In these works, quality metrics are designed for specific query types. Our metrics, on the other hand, measure the amount of information gain (i.e. entropy) contained in the query result, which provide a fair comparison of quality among the answers from different queries. We also propose efficient solutions to compute the metrics. Another quality metric, called the query reliability, is defined in [32, 42]. However, this metric was not studied in the context of probabilistic databases. Moreover, it is not clear how they can be applied to the problem of data cleaning.

3 Probabilistic Nearest-Neighbor Queries

As mentioned before, data uncertainty prevails in many applications. The evaluation of queries over uncertain database encounters many challenges. We now present our works about efficient evaluation of an important query class for uncertain data, i.e. the Probabilistic Nearest-Neighbor Query (PNN). Evaluating this query is computationally expensive, since it involves integration operations over complex functions. By observing that a user is interested only in answers with high probability values and can tolerate a controlled degree of error, we propose a variant of PNN known as the Constrained Probabilistic Nearest-Neighbor Query (C-PNN). To answer C-PNN accurately and efficiently, we develop a series of algorithms known as *probabilistic verifiers*. These verifiers can support a variety of pdfs. We discuss the data structure and run time issues for these verifiers. Experiments on both synthetic and real-time data sets show that these verifiers can significantly reduce the cost of evaluating a C-PNN.

3.1 Introduction

An important query for uncertain objects is the *Probabilistic Nearest Neighbor* Query (PNN in short) [23]. This query returns the non-zero qualification probability of each object for being the nearest neighbor of a given point q. The qualification probability augmented with each object allows us to place confidence onto the answers. Figure 3.1 illustrates an example of PNN on four uncertain objects (A, B, C and D). Also shown in the figure is the query point q and the qualification probability of each object. A PNN could be used in a scientific application, where sensors are deployed to collect the temperature values in a natural habitat. For data analysis and clustering purposes,

one may execute a PNN on the sensor data to find out which district(s) possess temperature(s) that are the closest to a given centroid. Another example is to find the IDs of sensor(s) that yield the minimum (maximum) wind-speed from a given set of sensors [23, 33]. It is worth notice that a minimum (maximum) query can be characterized as a PNN by setting q to a value of $-\infty$ (respectively ∞).

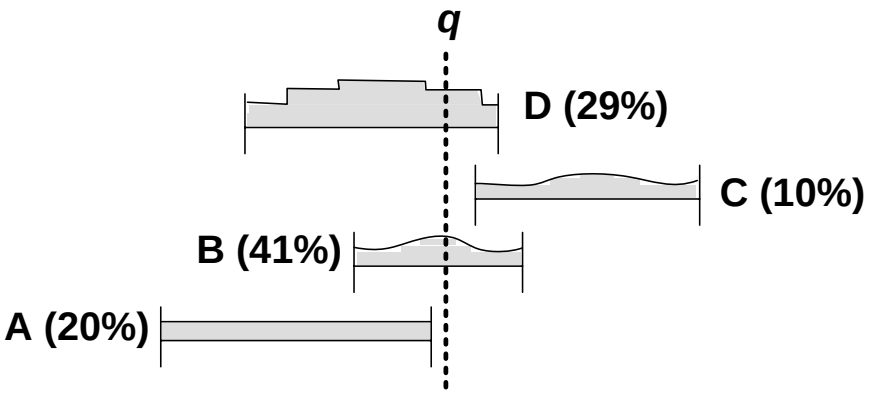


Figure 3.1: Probabilistic NN Query (PNN).

Although PNN is useful, evaluating it is not an easy task. In particular, since the exact value of a data item is not known, one needs to consider the item's possible values in its uncertainty region. Furthermore, an object's qualification probability depends not just on its own value, but also the relative values of other objects. If the uncertainty regions of the objects overlap, then their pdfs must be considered in order to derive their corresponding probabilities. In Figure 3.1, for instance, evaluating A's qualification probability (20%) requires us to consider the pdfs of the other three objects, since each of them has some chance of overtaking A as the nearest neighbor of q.

To our best knowledge, currently there are two major techniques for computing the exact values of qualification probabilities. The first method is to derive the pdf and cdf of the distance of each object from q. The qualification

probability of an object is then computed by integrating a function of multiple objects' distance pdf's and cdf's over the uncertainty region [23, 24, 33]. Another method is to use the Monte-Carlo method, where the pdf of each object is sampled and represented as a set of points. The qualification probability is evaluated by considering the portion of points that could be the nearest neighbor of q [55]. The cost of these solutions is quite high, since they either require numerical integration over some aggregate functions of arbitrary pdfs, or the handling of samples that has to be acquired from each object. Furthermore, the accuracy of the answers depends on the precision of numerical integration or number of samples used. Notice that there exists indexing solutions for pruning objects with zero qualification probabilities for PNN [24, 25]. In some situations, however, the computation time of qualification probabilities is at least the amount of time needed for index retrieval. Hence, there is a genuine need for reducing the computation time.

3.1.1 Solution Overview

As explained, the cost of evaluating the exact value of qualification probability for PNN queries could be quite expensive. However, in some situations query users may not be interested in the precise value of qualification probabilities. For example, a user may only be just interested in answers with confidence higher than a predefined threshold. He/She may also be satisfied with answers within some degree of approximation error. Consider the example in Figure 3.1 again. A user may only be interested with uncertain objects with qualification probabilities higher than a lower bound of 30%. If the user further allows a maximum error of $\pm 2\%$ for calculating probabilities, then both objects B and D would be the answer. In this example, the *threshold* (30%) and *tolerance*($\pm 2\%$) are additional requirements or *con-*

straints imposed on the answers of a PNN. We denote this modified version of the PNN as the **Constrained Probabilisitc Nearest-Neighbor Query** (or **C-PNN** in short). The C-PNN has additional advantage over its PNN counterpart, since it succinctly captures the needs of the query user, and facilitates the user to interpret the probabilistic guarantees of the answers more easily. In our example, the C-PNN returns only two objects (B,D) as the answer, as opposed to the PNN, where the qualification probabilities of all the four objects are returned.

The use of the C-PNN has another implication – its answers can be generated in an efficient manner. By exploiting the threshold and tolerance constraints, we show that it is possible to avoid costly operations like integration and Monte-Carlo methods. Specifically, we have developed a number of testing criteria, known as *probabilistic verifiers*, for making decisions on whether an object should be included into, or excluded from, the final answer. Probabilistic verifiers are efficient because they only derive the lower or upper bound of qualification probabilities with simple algebraic operations. For example, the **U-SR** verifier (to be described in this chapter) uses the objects' uncertainty regions to deduce that the qualification probability of object A in Figure 3.1 cannot be higher than 25%. If the threshold is 30% and tolerance is $\pm 2\%$, we can immediately exclude A from the C-PNN's answer set, without calculating A's exact probability (20%).

Figure 3.2 shows the role of probabilistic verifiers played in our solution. There are three major stages. The first step is to prune or *filter* objects that have no chance of being the nearest neighbor of q, using an R-tree based solution [25]. The objects with non-zero qualification probabilities (shaded) are then passed to the *verification* stage, where probabilistic verifiers are used to determine whether an object satisfies or fails the C-PNN. In the

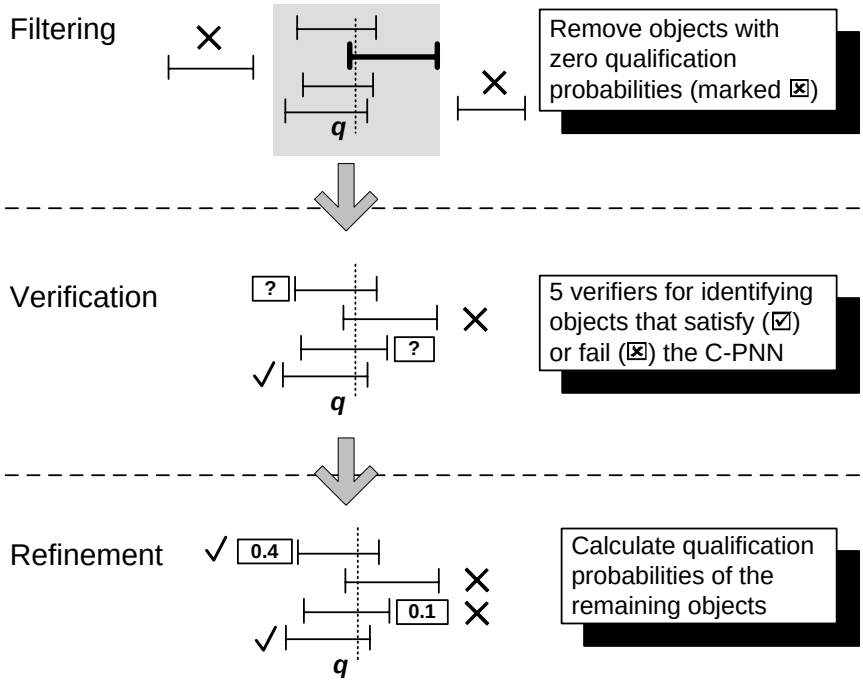


Figure 3.2: Solution Framework of C-PNN.

figure shown, two objects have been determined (one satisfies and the other fails the C-PNN). Objects that cannot be determined by the verifiers are then passed to the *refinement* stage, where the numerical integration will be utilized for making final decision. In this example, since the threshold is 0.3, the object with probability 0.4 will be included, while the one with probability 0.1 is excluded.

In this chapter, we focus on the verification and refinement stages. Particularly, we present the main ideas of five probabilistic verifiers. These verifiers can be classified into two groups. The first type of verifiers, known as *result-based verifiers*, exploits the relationship between the qualification probabilities of objects. The second type, called *subregion-based verifiers*, requires more information than the first group, but is also more powerful. It utilizes the information of uncertainty regions and pdfs of the objects in

order to facilitate verification. All these verifiers can handle arbitrary pdfs in the form of histograms. We propose a paradigm that strings these verifiers together in order to provide an efficient solution. We demonstrate that even if the verifiers cannot successfully decide on the object, the information they have computed can still be used to accelerate the refinement process. As shown in our experiments for both real and synthetic datasets, this approach only adds a small run-time and space overhead. The price paid for deploying the verifiers is justified by the fact the cost of refinement is largely reduced. This results in significant improvement in response time – the query execution time ranges from two to forty times better than if no verifiers are used.

The rest of this chapter is organized as follows. In Section 3.2, we present the solution framework for C-PNN. We present the probabilistic verifiers in Section 3.3. Experimental results are presented in Section 3.4. We conclude the chapter in Section 3.5.

3.2 A Solution Framework for C-PNN

In this section we describe the data uncertainty model (Section 3.2.1), and present the formal semantics of C-PNN (Section 3.2.2). We then outline our solution in Section 3.2.3.

3.2.1 Attribute-Uncertainty Model

We assume an uncertain object, namely o_i, is associated with an imprecise attribute, e.g. temperature, humidity, and location etc. For notational convenience, we also use o_i to denote its imprecise attribute. In Section 1.1, we have mentioned the *attribute-uncertainty* model to capture data uncertainty. Formally, this uncertainty model consists of two components:

Definition 3.1 *The* **uncertainty region** *of o_i, denoted by u_i, is a closed region which bounds the imprecise value of o_i.*

Definition 3.2 *The* **uncertainty probability density function (pdf)** *of object o_i, denoted by $f_i(\bullet)$, is a pdf of o_i, that has a value of 0 outside u_i.*

The formula for the uncertain pdf is application-specific. Wolfson et al. propose that the object location follows the Gaussian distribution inside the uncertainty region [76]. An important case of uncertainty pdf is a uniform distribution [70], that is, $f_i(\bullet) = \frac{1}{|u_i|}$; essentially, this implies a "worst-case" scenario where we have no knowledge of which point in the uncertainty region possesses a higher probability.

Next, we will present the formal semantics of C-PNN.

3.2.2 Definition of C-PNN

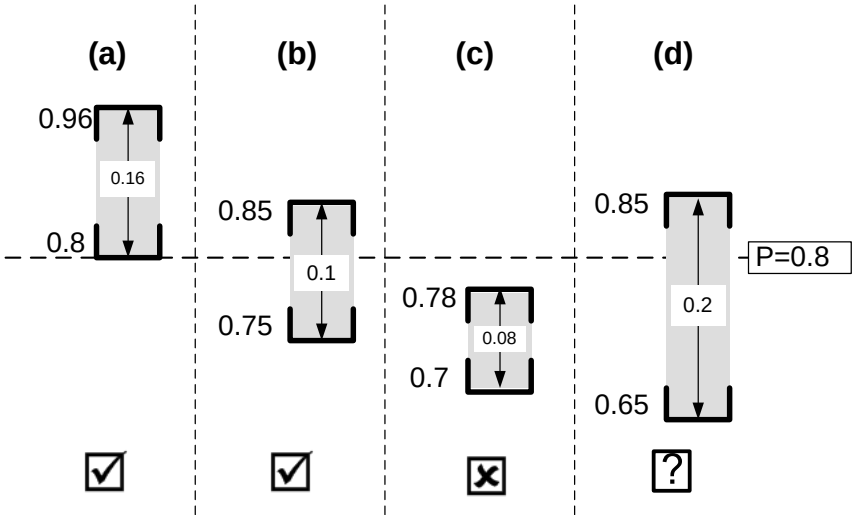


Figure 3.3: A C-PNN with $T = 0.8$ and $\Delta = 0.15$.

Let D be a set of uncertain objects in 1D space (i.e., an arbitrary pdf defined inside a closed interval), and let o_i be the i^{th} item of D (where

$i = 1, 2, \ldots, |D|$). Let $q \in \Re$ be the query point of PNN, and $p_i \in [0, 1]$, be the qualification probability (i.e., the probability that o_i is the nearest neighbor of q). We suppose $p_i.l, p_i.u \in [0, 1]$, such that $p_i.l \leq p_i.u$, and $p_i \in [p_i.l, p_i.u]$. Intuitively, $[p_i.l, p_i.u]$ is the range of the qualification probability p_i, and $p_i.u - p_i.l$ is the error in estimating the actual value of p_i. For convenience, we call this range *probability bound*. The C-PNN is then defined as follows.

Definition 3.3 *A* **Constrained Probabilistic Nearest Neighbor Query (C-PNN)** *returns a set* $\{o_i | i = 1, 2, \ldots, |D|\}$ *such that* p_i *satisfies both of the following conditions:*

- $p_i.u \geq T$
- $p_i.l \geq T$, *or* $p_i.u - p_i.l \leq \Delta$

where $T \in (0, 1]$ *and* $\Delta \in [0, 1]$.

Here, T is called the *threshold* of a C-PNN. It allows an uncertain object to be returned as an answer, only if its qualification probability is not less than T. We also define the parameter *tolerance* (Δ in short), which limits the amount of error allowed in the qualification probability (p_i). Figure 3.3 illustrates the semantics of C-PNN. It shows the probability bound $[p_j.l, p_j.u]$ of some object o_j (shaded) in four different scenarios. Let us assume the C-PNN has a threshold $T = 0.8$ and tolerance $\Delta = 0.15$. Case (a) shows that the actual qualification probability p_j of some uncertain object o_j (i.e., p_j) is within a closed bound of $[p_j.l, p_j.u]$=$[0.8, 0.96]$. Since p_j must be not smaller than T, according to Definition 3.3, o_j is an answer to this C-PNN. In scenario (b), o_j is also a valid answer since the upper bound of p_j (i.e., $p_j.u$), equals to 0.85 and is larger than T. Moreover, the error of estimating p_j (i.e., 0.85-0.75), is 0.1, which is less than Δ (0.15). Thus the two conditions of Definition 3.3 are satisfied. For (c), o_j could not the answer, since

the upper bound of p_j (i.e., 0.78) is less than T, and so the first condition of Definition 3.3 is violated. In (d), although object o_j satisfies the first requirement ($p_j.u = 0.85 \geq T$), the second condition is not met. According to Definition 3.3, it is not an answer to the C-PNN. However, if the probability bounds could later be "shrinked" (e.g., by probabilistic verifiers), then the conditions can be checked again. For instance, if $p_j.l$ is later updated to 0.81, then o_j will be the answer. Table 3.1 summarizes the symbols used in the C-PNN problem.

The use of Δ is to represent a query issuer's tolerance to the error contained in the result probabilities, which cannot be achieved by simply reducing the probability threshold. To see this, let us use a "lower probability bound l" instead of the probability tolerance Δ, by setting $l = T - \Delta$. Then, we use the pair $[l, T]$ to test the answer probability bounds. In Figure 3.3, for example, if $T = 0.8$ and $\Delta = 0.15$, then $l = 0.8 - 0.15 = 0.65$. With some slight adjustment in the testing procedure, we can achieve the same effect of using Δ. In case (c), for instance, we need to test that the upper probability bound is less than T, whereas in (d), the undetermined case happens when the width of the probability bound is higher than $T - l$ (i.e., Δ). Note that in both of these cases, it is not sufficient to use l alone - we also have to use T in doing the testing. In other words, we still need to use two values ((l, T) instead of (T, Δ)) to perform the testing. Since Δ, which represents the amount of tolerance in the probability error, may be easier to understand, we suggest to use Δ instead of l.

3.2.3 The Verification Framework

Recall that uncertain objects that cannot be pruned in the filtering stage (shaded in Figure 3.2) are passed to the verification and refinement phases.

Symbol	Meaning
o_i	An uncertain object ($i = 1, \ldots, \|D\|$)
u_i	Uncertainty region of o_i
$f_i(\bullet)$	Uncertainty pdf of o_i
q	Query point
p_i	Prob. that o_i is the NN of q (qualification prob.)
$[p_i.l, p_i.u]$	Lower & upper bounds of p_i
T	Threshold
Δ	Tolerance

Table 3.1: Symbols for C-PNN.

We use the term *candidate set* (C in short) to denote this group of objects. Let us now describe how objects in a candidate set are verified and refined.

As discussed before, *probabilistic verifiers* (or "verifiers" in short) are used to decide whether an uncertain object satisfies a given C-PNN. More specifically, a verifier is an algorithm that produces a list of probability bounds of the objects in the candidate set. This list is passed to a *classifier*, which assigns a label to an uncertain object by checking its probability bounds against the two conditions in Definition 3.3. The object is marked *satisfy* if it qualifies as the answer to the C-PNN (Figures 3.3(a), (b)). It is labeled *fail* if it surely does not satisfy the C-PNN (Figure 3.3(c)). Otherwise, the object is marked *unknown* (Figure 3.3(d)). Note that this labeling task can be done easily by checking an object's probability bounds against the two conditions in Definition 3.3.

Figure 3.4 shows the five verifiers (in shaded boxes) and the classifier. Notice the L-PB verifier is used twice. During initialization, all the objects in the candidate set are labeled *unknown*, and their probability bounds are set to $[0, 1]$. Other essential information such as distance pdf and cdf are also precomputed for the candidate set objects. The candidate set is then passed to the first verifier (RS) for processing. The RS produces the newly computed

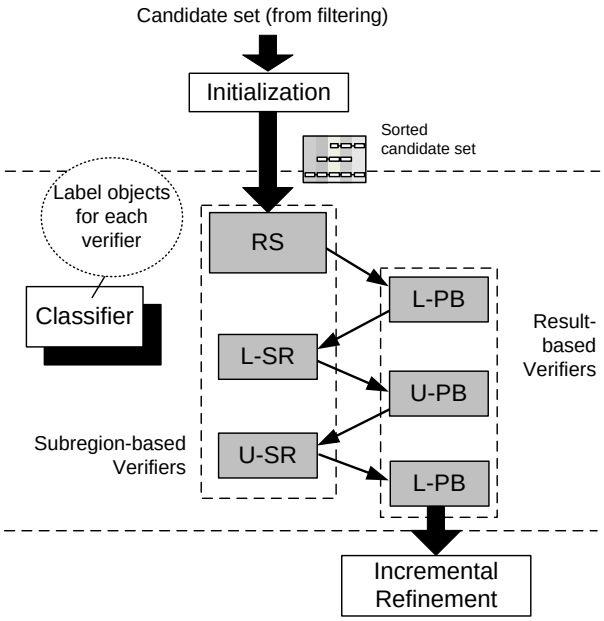


Figure 3.4: The Verification Framework.

probability bounds for the objects in the candidate sets, and passes this list to the classifier to label the objects. Any objects that are labeled *unknown* are passed to the next verifier (L-PB), and so on, until all the objects are either labeled *satisfy* or *fail*. When this happens, all the objects that are marked *satisfy* are returned to the user, and thus the query is finished. Thus, it is not always necessary for all the verifiers to be executed.

Notice that a verifier only adjusts the probability bound of an *unknown* object if this new bound is smaller than the one computed previously. Also, the verifiers are arranged in the order of their running times, so that if a low-cost verifier (e.g., the RS verifier) can successfully determine all the objects, there is no need to trigger the execution of a more costly verifier (e.g., the L-SR verifier). In the end of verification, any objects that are still labeled *unknown* are passed to the refinement stage for performing computing their exact probabilities. We discuss a faster technique to improve this process in

Section 3.3.5. Now let us examine the details of the verifiers.

3.3 Verification and Refinement

From Figure 3.4, we can see that the verifiers are classified into *subregion-based* and *result-based*. The main difference between these two classes of verifiers lies in the way they derive probability bounds. Subregion-based verifiers uses the information of *subregions* to compute the probability bounds. A subregion is essentially a partition of the space derived from the uncertainty regions of the candidate set objects. This information is used by the RS, L-SR and U-SR verifiers. On the other hand, the result-based verifiers (i.e., U-PB and L-PB) uses results derived by the subregion-based verifiers to infer the bounds. For example, the U-PB verifier uses the lower-bound information derived by the L-SR verifier. As we will explain, a subregion-based verifier is more powerful than a result-based verifier, but are also more costly to use.

The rest of this section is organized as follows. Section 3.3.1 discusses how subregions are produced. We then present the RS-verifier in Section 3.3.2, followed by the L-SR and U-SR verifiers in Section 3.3.3. In Section 3.3.4 we examine the result-based verifiers. We then describe the "incremental refinement" method, which uses the verifiers' information to improve the refinement process, in Section 3.3.5.

3.3.1 Computing Subregion Probabilities

The initialization phase in Figure 3.4 performs two tasks: (1) computes distance pdf and cdf for each object in the candidate set, and (2) derives *subregion probabilities*.

We start with the derivation of distance pdf and cdf. Let $R_i \in \Re$ be the

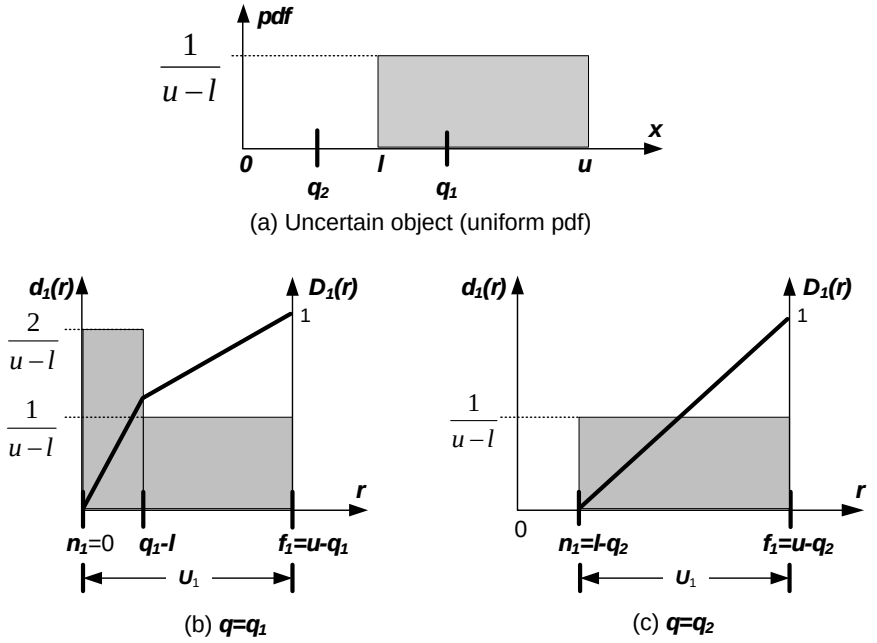


Figure 3.5: Distance pdf and cdf

absolute distance of an uncertain object o_i from q. That is, $R_i = |o_i - q|$. We assume that R_i takes on a value $r \in \Re$. Then, the distance pdf and cdf of o_i are defined as follows [23, 24]:

Definition 3.4 *Given an uncertain object o_i, its* **distance pdf**, *denoted by $d_i(r)$, is a pdf of R_i; its* **distance cdf**, *denoted by $D_i(r)$, is a cdf of R_i.*

Figure 3.5(a) illustrates an uncertain object o_1, which has a uniform pdf with a value of $\frac{1}{u-l}$ in an uncertainty region $[l, u]$. Two query points (q_1 and q_2) are also shown. Figure 3.5(b) shows the corresponding distance pdf (shaded) of $R_1 = |o_1 - q_1|$, with q_1 as the query point. Essentially, we derive the pdf of o_1's distance from q_1, which ranges from 0 to $u - q_1$. In $[0, q_1 - l]$, the distance pdf is obtained by summing up the pdf on both sides of q_1, which equals to $\frac{2}{u-l}$. The distance pdf in the range $[q_1 - l, u - q_1]$ is simply $\frac{1}{u-l}$.

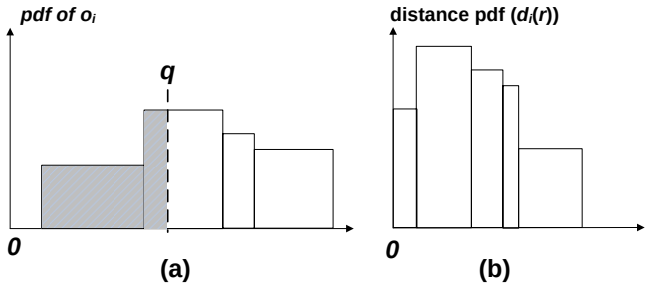


Figure 3.6: Histogram pdf.

Figure 3.5(c) shows the distance pdf for query point q_2. For both queries, we draw the distance cdf in solid lines. Notice that the distance cdf can be found by integrating the corresponding distance pdf. From Figure 3.5, we observe that the distance pdf and cdf for the same uncertain object vary, and depend on the position of the query point.

We represent the distance pdf of each uncertain object as a histogram. Note that this distance pdf/cdf can conceptually be found by first decomposing the histogram into a number of "histogram bars", where the pdf in the range of each bar is the same. We can then compute the distance pdf/cdf of each bar using the methods (for uniform pdf) described in the previous paragraph, and combine the results to yield the distance pdf/cdf for the histogram. Figure 3.6(a) illustrates the histogram pdf of an uncertain object, and its corresponding distance pdf (in (b)). The corresponding distance cdf is a piecewise linear function. In practice, we store a histogram as an array of the histogram bars' end-points and their corresponding pdfs. Then, given a query q, we split the histogram into two halves (as shown in the two shaded parts in Figure 3.6). A merge sort is then performed on these end-points, during which their distance pdfs and cdfs are computed. If there are a total of h end-points in a histogram pdf, the process of generating distance pdfs and cdfs can be done in $O(h)$ times.

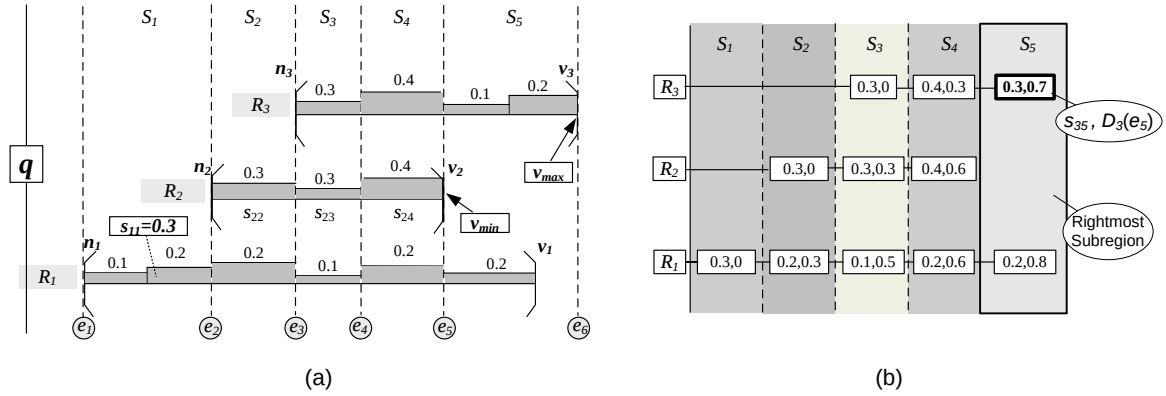


Figure 3.7: Illustrating the distance pdfs and subregion probabilities.

Note that although we focus on 1D uncertainty, our solution only needs distance pdfs and cdfs. Thus, our solution can be extended to 2D space, by computing the distance pdf and cdf from the 2D uncertainty regions, using the formulae discussed in [24].

Now, let us describe the definitions of near and far points of R_i, as defined in [23, 24]:

Definition 3.5 *A* **near point** *of R_i, denoted by n_i, is the minimum value of R_i. A* **far point** *of R_i, denoted by v_i, is the maximum value of R_i.*

We use U_i to denote the interval $[n_i, v_i]$. Figure 3.5(b) shows that when q_1 is the query point, $n_1 = 0$, $v_1 = u - q_1$, and $U_1 = [0, u - q_1]$. When q_2 is the query point, $U_1 = [l - q_2, u - q_2]$ (Figure 3.5(c)). We also let v_{min} and v_{max} be the minimum and maximum values of all the far points defined for the candidate set objects. We assume that the distance pdf of o_i has a non-zero value at any point in $[n_i, v_i]$.

Subregion Probabilities. Upon generating the distance pdfs and cdfs of the candidate set objects, the next step is to generate *subregions*. Let us first sort these objects in the ascending order of their near points. We also

rename the objects as $o_1, o_2, \ldots, o_{|C|}$, where $n_i \leq n_j$ iff $i \leq j$. Figure 3.7(a) illustrates three distance pdfs with respect to a query point q, presented in the ascending order of their near points. The number above each range indicates the probability that an uncertain object has that range of distance from the query point.

In Figure 3.7(a), the circled values are called *end-points*. They include all the near points (e.g., e_1, e_2 and e_3), the minimum and maximum of far points (e.g., e_5 and e_6), and the point at which the distance pdf changes (e.g., e_4). No end points are defined between (e_1, e_2) and (e_5, e_6). We use e_j to denote the j-th end-point, where $j \geq 1$ and $e_j < e_{j+1}$. Moreover, $e_1 = n_1$.

The adjacent pairs of end-points form the boundaries of a *subregion*. We label each subregion as S_j, where S_j is the interval $[e_j, e_{j+1}]$. Figure 3.7(a) shows five subregions, where $S_1 = [e_1, e_2]$, $S_2 = [e_2, e_3]$, and so on. The probability that R_i is located in S_j is called the *subregion probability*, denoted by s_{ij}. Figure 3.7(a) shows that $s_{22} = 0.3$, $s_{11} = 0.1 + 0.2 = 0.3$, and $s_{31} = 0$.

For each subregion S_j of an object o_i, we evaluate the subregion probability s_{ij}, as well as the distance cdf of S_j's lower end-point (i.e., $D_i(e_j)$). Figure 3.7(b) illustrates these pairs of values extracted from the example in (a). For example, for R_3 in S_5, the pairs $s_{35} = 0.3$ and $D_3(e_5) = 0.7$ are shown. These number pairs help the verifiers to develop the probability bounds. Table 3.2 presents the symbols used in our solution. Let us now examine how the verifiers work.

3.3.2 The Rightmost-Subregion Verifier

The **Rightmost-Subregion** (or RS) verifier uses the information in the "rightmost" subregion. In Figure 3.7(b), S_5 is the rightmost subregion. If we let $M \geq 2$ be the number of subregions for a given candidate set, then

Symbol	Description
C	$\{o_i \in D \mid p_i > 0\}$ (*candidate set*)
R_i	$\lvert o_i - q \rvert$
$d_i(r)$	pdf of R_i (*distance pdf*)
$D_i(r)$	cdf of R_i (*distance cdf*)
n_i, v_i	Near and far points of distance pdf
U_i	The interval $[n_i, v_i]$
v_{min}, v_{max}	min. and max. of far points
e_k	The k-th end point
S_j	The j-th subregion, where $S_j = [e_j, e_{j+1}]$
M	Total no. of subregions
c_j	No. of objects with distance pdf in S_j
s_{ij}	$Pr(R_i \in S_j)$
q_{ij}	Qualification prob. of o_i, given $R_i \in S_j$
$[q_{ij}.l, q_{ij}.u]$	Lower & upper bounds of q_{ij}

Table 3.2: Symbols for verifiers.

the following specifies an object's upper probability bound:

Lemma 3.1 *The upper probability bound, $p_i.u$, is at most $1 - s_{iM}$, where s_{iM} is the probability that R_i is in S_M.*

The subregion S_M is the rightmost subregion. In Figure 3.7(b), $M = 5$. The upper bound of the qualification probability of object o_1, according to Lemma 3.1, is at most $1 - s_{15}$, or $1 - 0.2 = 0.8$.

To understand this lemma, notice that any object with distance larger than v_{min} *cannot* be the nearest neighbor of q. This is because v_{min} is the minimum of the far points of the candidate set objects. Thus, there exists an object o_k such that o_k's far point is equal to v_{min}, and that o_k is closer to q than any objects whose distances are larger than v_{min}. If we also know the probability of an object located beyond a distance of v_{min} from q, then its upper probability bound can be deduced. For example, Figure 3.7(a) shows that the distance of o_1 from q (i.e., R_1) has a 0.2 chance of being more than v_{min}. Thus, o_1 is not the nearest neighbor of q with a probability of

at least 0.2. Equivalently, the upper probability bound of o_1, i.e., $p_1.u$, is $1 - 0.2 = 0.8$. Note that 0.2 is exactly the probability that R_1 lies in the rightmost subregion S_5, i.e., s_{15}, and thus $p_1.u$ is equal to $1 - s_{15}$. This result can be generalized for any object in the candidate set, as shown in Lemma 3.1.

Notice that the RS verifier only handles the objects' upper probability bounds. To improve the lower probability bound, we need the L-SR verifier, as described next.

3.3.3 The Lower- and Upper- Subregion Verifiers

The Lower-Subregion (**L-SR**) and Upper-Subregion (**U-SR**) Verifiers uses subregion probabilities to derive the objects' probability bounds. For each subregion the L-SR (U-SR) verifier computes the lower (upper) probability bound of each object.

We define the term *subregion qualification probability* (q_{ij} in short), which is the chance that o_i is the nearest neighbor of q, given that its distance from q, i.e., R_i, is inside subregion S_j. We also denote the lower bound of the subregion qualification probability as $q_{ij}.l$. Our goal is to derive $q_{ij}.l$ for object o_i in subregion S_j. Then, the lower probability bound of o_i, i.e., $p_i.l$, is evaluated. Suppose there are $c_j (c_j \geq 1)$ objects with non-zero subregion probabilities in S_j. For example, $c_3 = 3$ in Figure 3.7(a), where all three objects have non-zero subregion probabilities in S_3. The following lemma is used by the L-SR verifier to compute $q_{ij}.l$.

Lemma 3.2 *Given an object $o_i \in C$, if $e_j \leq R_i \leq e_{j+1}$ ($j = 1, 2, \ldots, M-1$),*

then

$$q_{ij}.l = \frac{1}{c_j} \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i} (1 - D_k(e_j)) + (1 - \frac{1}{c_j}) \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i} (1 - D_k(e_{j+1})) \quad (3.1)$$

Lemma 3.2 calculates $q_{ij}.l$ for object o_i by using the distance cdfs of all objects with non-zero subregion probabilities in S_j. We will prove this lemma soon. To illustrate the lemma, Figure 3.7(a) shows that $q_{11}.l$ (for o_1 in subregion S_1) is equal to 1, since $c_1 = 1$. On the other hand, $q_{23}.l$ (for o_2 in S_3) is $\frac{(1-0.5)(1-0)}{3} + (1 - \frac{1}{3})(1 - 0.3)(1 - 0.6)$, or 0.35.

Next, we define a real constant Y_j, where

$$Y_j = \prod_{U_k \cap S_j \neq \emptyset} (1 - D_k(e_j)) \quad (3.2)$$

Then, Equation 3.1 can be rewritten as:

$$q_{ij}.l = \frac{Y_j}{c_j(1 - D_i(e_j))} + (1 - \frac{1}{c_j}) \frac{Y_{j+1}}{1 - D_i(e_{j+1})} \quad (3.3)$$

By computing Y_j first, the L-SR can use Equation 3.3 to compute $q_{ij}.l$ easily for each object in the same subregion S_j.

After the values of $q_{ij}.l$ have been obtained, the lower probability bound $(p_i.l)$ of object o_i can be evaluated by:

$$p_i.l = \sum_{j=1}^{M-1} s_{ij} \cdot q_{ij}.l \quad (3.4)$$

The product $s_{ij} \cdot q_{ij}.l$ is the minimum qualification probability of o_i in subregion s_{ij}, and Equation 3.4 is the sum of this product over the subregions.

Note that the rightmost subregion (S_M) is not included, since the probability of any object in S_M must be zero.

The U-SR verifier uses Lemma 3.3 to evaluate the upper subregion probability bound ($q_{ij}.u$) of object o_i in subregion S_j:

Lemma 3.3 *Given an object $o_i \in C$, if $e_j \leq R_i \leq e_{j+1}$ ($j = 1, 2, \ldots, M-1$), then*

$$q_{ij}.u = \begin{cases} 1 & \text{if } j = 1 \\ \frac{1}{2} \cdot (\prod_{U_k \cap S_{j+1} \neq \emptyset \wedge k \neq i}(1 - D_k(e_{j+1})) & \\ + \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i}(1 - D_k(e_j))) & \text{if } j > 1 \end{cases} \tag{3.5}$$

Similar to L-SR, Equation 3.5 can be simplified to:

$$q_{ij}.u = \begin{cases} 1 & \text{if } j = 1 \\ \frac{1}{2}(\frac{Y_j}{1-D_i(e_j)} + \frac{Y_{j+1}}{1-D_i(e_{j+1})}) & \text{if } j > 1 \end{cases} \tag{3.6}$$

where Y_j and Y_{j+1} are given by Equation 3.2. Thus, if the Y_j's are known, we can conveniently compute $q_{ij}.u$ with Equation 3.6. The upper probability bound ($p_i.u$) can be computed by replacing $q_{ij}.l$ with $q_{ij}.u$ in Equation 3.4. Next, we present the correctness proofs of the L-SR and U-SR verifiers.

Correctness of L-SR and U-SR.

We first state a claim about subregions: if there exists a set K of objects whose distances from q (i.e., R_i) are certainly inside a subregion S_j, and all other objects ($C - K$)'s distances are in subregions $j + 1$ or above, then the qualification probability of each objects in K is equal to $\frac{1}{|K|}$. This is because all objects in $C - K$ cannot be the nearest neighbor of q, and all objects in K must have the same qualification probability. In Figure 3.7(a), for example,

if the distances R_1 and R_2 are inside $S_2 = [e_2, e_3]$, then $p_1 = p_2 = \frac{1}{2}$, and $p_3 = 0$. The following lemma states this formally.

Lemma 3.4 *Suppose there exists a nonempty set $K (K \subseteq C)$ of objects such that $\forall o_i \in K, e_j \leq R_i \leq e_{j+1}$. If $\forall o_m \in C - K, R_m > e_{j+1}$, then for any $o_i \in K$, $p_i = \frac{1}{|K|}$, where $|K|$ is the number of objects in K.*

Proof : Let us consider an object $o_i \in K$. When $|K| = 1$, $p_i = 1$. This is because the distances of other $C - 1$ objects must be larger than R_i. Therefore, o_i must be the nearest neighbor, and the lemma is proved.

Now consider $|K| > 1$. According to [23], p_i can be obtained by

$$\begin{aligned} p_i &= \int_{n_i}^{v_i} Pr(R_i = r) \prod_{k=1 \wedge k \neq i}^{|C|} Pr(R_k > r) dr \\ &= \int_{n_i}^{v_i} d_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{|C|} (1 - D_k(r))\ dr \end{aligned} \tag{3.7}$$

Since $d_i(r)$ is a uniform distribution, and $R_i \in [e_j, e_{j+1}]$, we have $d_i(r) = \frac{1}{e_{j+1} - e_j}$ for $r \in [e_j, e_{j+1}]$. Moreover, for any $o_k \in K$, $D_k(r) = \frac{r - e_j}{e_{j+1} - e_j}$. The remaining $|C - K|$ objects cannot be the nearest neighbor, since they must be farther than o_i from q. Hence p_i can be calculated by considering only the objects in K. Substituting into Equation 3.7, we then have

$$\begin{aligned} p_i &= \int_{n_i}^{v_i} \frac{1}{e_{j+1} - e_j} \cdot \prod_{o_k \in K \wedge k \neq i} (1 - \frac{r - e_j}{e_{j+1} - e_j})\ dr \\ &= \int_{e_j}^{e_{j+1}} \frac{1}{e_{j+1} - e_j} \cdot (\frac{e_{j+1} - r}{e_{j+1} - e_j})^{|K|-1}\ dr \end{aligned} \tag{3.8}$$

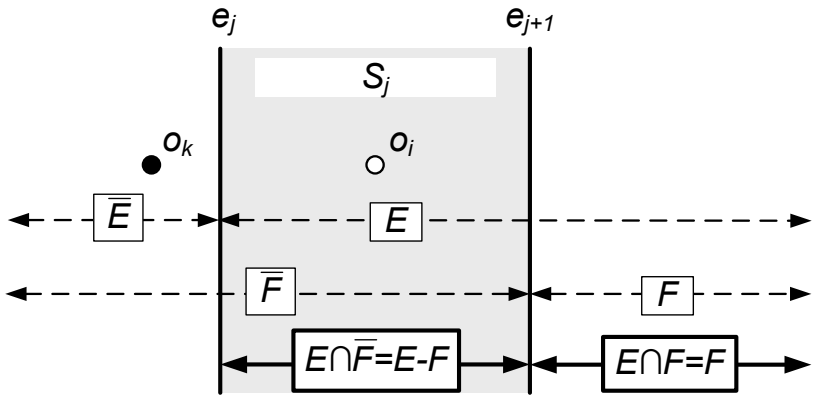


Figure 3.8: Correctness proof for L-SR and U-SR.

Note that we can change the integration bounds from $[n_i, v_i]$ to $[e_j, e_{j+1}]$, since we only consider the objects in K, which are known to be within $[e_j, e_{j+1}]$. Equation 3.8 will lead to the result $p_i = \frac{1}{|K|}$ and prove the lemma. □

We can now prove the correctness of L-SR and U-SR. Let us examine when c_j, the number of objects with non-zero subregion probabilities in subregion S_j, is equal to 1. In fact, this scenario happens to subregion S_1, i.e., $j = 1$, since only this region can accommodate a single distance pdf (e.g., $d_1(r)$ in Figure 3.7). If we also know that distance R_i is in subregion S_j, then o_i must be the nearest neighbor. Thus, both the lower and upper subregion qualification probability bounds ($q_{ij}.l$ and $q_{ij}.u$) are equal to 1, as shown in Equations 3.5.

For the case $c_j > 1$, we derive the subregion qualification probability, q_{ij}. Let E denote the event that "all objects in the candidate set C have their actual distances from q not smaller than e_j". Also, let $\bar{E}$ be the complement of event E, i.e., "there exists an object whose distance from q is less than e_j". We also let F be the event "$\forall o_k \in C$, where $k \neq i$, $R_k \geq e_{j+1}$", and $\bar{F}$

be the event "$\exists o_k \in C$ s.t. $k \neq i \wedge R_k < e_{j+1}$". Figure 3.8 illustrates these four events.

If $Pr(E)$ denotes the probability that event E is true, then $Pr(E) = 1 - Pr(\bar{E})$. We also have $Pr(F) = 1 - Pr(\bar{F})$. Let N be the event "Object o_i is the nearest neighbor of q". Then, using the law of total probability, we have:

$$q_{ij} = Pr(N|E) \cdot Pr(E) + Pr(N|\bar{E}) \cdot Pr(\bar{E}) \tag{3.9}$$

If $\bar{E}$ is true, there is at least one object o_k whose distance R_k is not larger than e_j (Figure 3.8). Since $R_k < R_i$, object o_k must be closer to q than object o_i. Consequently, $Pr(N|\bar{E}) = 0$, and Equation 3.9 becomes:

$$q_{ij} = Pr(N|E) \cdot Pr(E) \tag{3.10}$$

which again, by using the law of total probability, can be rewritten as

$$q_{ij} = Pr(N|E \cap F) \cdot Pr(E \cap F) + Pr(N|E \cap \bar{F}) \cdot Pr(E \cap \bar{F}) \tag{3.11}$$

If $E \cap F$ is true, then all objects except o_i have their distances from q not smaller than e_{j+1}. Since $R_i \leq e_{j+1}$, it must be the nearest neighbor. Thus, $Pr(N|E \cap F) = 1$. We also have $E \cap F = F$ since $F \subset E$. So Equation 3.11 is reduced to:

$$q_{ij} = Pr(F) + Pr(N|E \cap \bar{F}) \cdot Pr(E \cap \bar{F}) \tag{3.12}$$

Next, suppose $E \cap \bar{F}$ is true. Then, in addition to o_i, $m \geq 1$ other object(s) is (are) on the left of e_{j+1}. Since E is also true, the values of R_k for all these m objects must also be in in S_j. Using Lemma 3.4, we can then deduce that $Pr(N|E \cap \bar{F}) = \frac{1}{m+1}$. The minimum value of $Pr(N|E \cap \bar{F})$ is $\frac{1}{c_j}$, which happens when $\forall o_k \in C$, where $k \neq i$ and $s_{kj} > 0, o_k \in S_j$. The

maximum value of $Pr(N|E \cap \bar{F})$ is $\frac{1}{2}$, which happens when $m = 1$. Thus we have

$$\frac{1}{c_j} \leq Pr(N|E \cap \bar{F}) \leq \frac{1}{2} \tag{3.13}$$

Notice that:

$$Pr(E \cap \bar{F}) = Pr(E) - Pr(F) \tag{3.14}$$

Thus, the final steps are to obtain $Pr(E)$ and $Pr(F)$. To obtain $Pr(E)$, note that the probability that an object o_k's distance is e_j or more is simply $1 - D_k(e_j)$. We then multiply all these probabilities, as $\prod_{R_k \geq e_j \wedge k \neq i}(1 - D_k(e_j))$. This can be simplified to:

$$Pr(E) = \prod_{U_k \cap S_j \neq \emptyset \wedge k \neq i} (1 - D_k(e_j)) \tag{3.15}$$

since any object whose subregion probability is zero in S_j must have the distance cdf at e_j, i.e.,$D_k(e_j)$, equal to zero.

Similarly, $Pr(F) = \prod_{U_k \cap S_{j+1} \neq \emptyset \wedge k \neq i}(1 - D_k(e_{j+1}))$.

Combining Equations 3.12, 3.13, 3.15, and 3.14, we can obtain the lower and upper bounds of q_{ij}, i.e., $q_{ij}.l$ and $q_{ij}.u$, as stated in Equations 3.1 and 3.5.

3.3.4 Result-based Verifiers

This class of verifiers makes use of the information deduced by the subregion-based verifiers. The intuition is that if we know the maximum qualification probabilities of any of the $|C| - 1$ objects in C, we can easily derive the lower probability bound of the remaining object in C. Specifically, given an object

o_i, its lower probability bound, $p_i.l$, can be computed as:

$$p_i.l = \max(1 - \sum_{T_k \in C \wedge k \neq i} p_k.u, 0) \tag{3.16}$$

This is because the total qualification probability values of the $|C| - 1$ objects (apart from o_i) is at most $\sum_{o_k \in C \wedge k \neq i} p_k.u$. Since the sum of qualification probabilities of all objects in C is equal to 1, $p_i.l$ can be deduced by Equation 3.16.

By using this principle, the Lower Probability Bound (**L-PB**) verifier refreshes the objects's lower probability bounds. It makes use of the new upper bound information yielded by the U-SR verifier (Figure 3.4). To implement L-PB, note that Equation 3.16 can be written as

$$p_i.l = \max(1 - p_{total} + p_i.u, 0) \tag{3.17}$$

where $p_{total} = \sum_{o_k \in C} p_k.u$.

Thus, we can compute p_{total} first (in $O(|C|)$ times), and then use Equation 3.17 to derive the probability lower bound for every object in C (also in $O(|C|)$ times). The total complexity for L-PB is thus equal to $O(|C|)$.

The other result-based verifier, namely, the Upper Probability Bound (U-PB) verifier, derives an object's upper probability bound by using the lower probability bounds of other objects. It is used after the L-SR verifier, which yields lower bound information (see Figure 3.4). The basic principle is the same as that of L-PB and so their details are not discussed here.

Why is the L-PB verifier not used after the RS verifier, which also yields objects upper probability bounds? The reason is that in our experiments, not many objects' upper bounds can be reduced by the RS. Consequently, most of the lower probability bounds derived by L-PB are equal to zero.

Therefore, we use L-PB only after the execution of U-SR.

3.3.5 Incremental Refinement

As discussed in the beginning of Section 3.3, some objects may still be unclassified after all verifiers have been applied. The exact probabilities of these objects must then be computed or "refined'. This can be expensive, since numerical integration has to be performed over the object's distance pdf [23]. This process can be performed faster by using the information provided by the verifiers. Particularly, the probability bounds of each object in each subregion (i.e., $[q_{ij}.l, q_{ij}.u]$) have already been computed by the verifiers. Therefore, we can decompose the refinement of qualification probability into a series of probability calculations inside subregions. Once we have computed the probability q_{ij} for subregion S_j, we collapse $[q_{ij}.l, q_{ij}.u]$ into a single value q_{ij}, update the probability bound $[p_i.l, p_i.u]$, and test this new bound with classifier. We repeat this process with another subregion until we can classify the object. This "incremental refinement" scheme is usually faster than directly computing qualification probabilities, since checking with a classifier is cheap, and performing numerical integration on a subregion is faster than on the whole uncertainty region, which has a larger integration area than a subregion. The time complexity of incremental refinement has a worst-case complexity of $O(|C|^2M)$, as detailed in [23].

Let us now discuss the implementation issues. We store the subregion probabilities (s_{ij}) and the distance cdf values ($D_i(e_j)$) for all objects in the same subregion as a list. These lists are indexed by a hash table, so that the information of each subregion can be accessed easily. The space complexity of this structure is $O(|C|M)$. It can be extended to a disk-based structure by partitioning the lists into disk pages. The complexities of the verifiers are

Verifier	Probability Bound	Cost
Subregion-based		
RS	Upper	$O(\|C\|)$
L-SR	Lower	$O(\|C\|M)$
U-SR	Upper	$O(\|C\|M)$
Result-based		
L-PB	Lower	$O(\|C\|)$
U-PB	Upper	$O(\|C\|)$

Table 3.3: Complexity of Verifiers.

shown in Table 3.3. The subregion-based verifiers (RS, L-SR and U-SR), as shown in Figure 3.4, are arranged in the ascending order of these running costs. The result-based verifier, L-PB (U-PB) are executed after U-SR (correspondingly U-SR). The complexity of verification (including initialization and sorting of candidate set objects) is $O(|C|(\log|C| + M))$, and is lower than the evaluation of exact probabilities ($O(|C|^2M)$).

3.4 Experimental Results

We have performed experimental evaluation on the effectiveness of our approaches. We first present our simulation model in Section 3.4.1. The detailed results are discussed in Section 3.4.2.

3.4.1 Experimental Setup

All experiments are run on a PC with an Intel T2400 1.83GHz CPU, 1024MB main memory and Windows XP. We implement the filtering phase with an R-tree index proposed in [24]. The R-Tree is developed in the Spatial Index Library version 0.44.2b [63].

We have used a real dataset called the *Long Beach*[1]. This dataset contains 53,144 rectangles, distributed in a $10K \times 10K$ space. Each rectangle

[1]Available at http://www.census.gov/geo/www/tiger/.

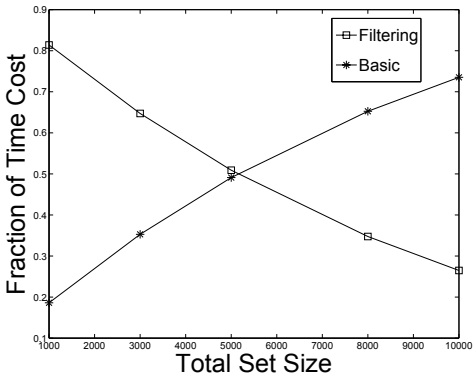


Figure 3.9: Basic vs. Filtering.

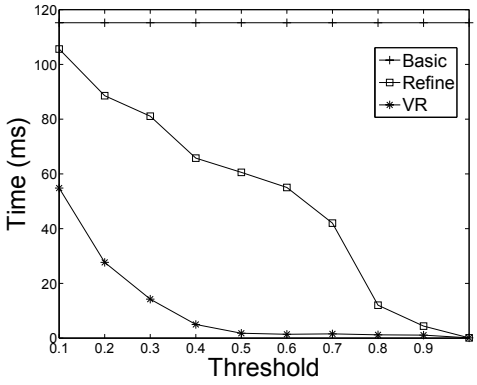


Figure 3.10: Time vs. T.

represents a region in the Long Beach. We treat these rectangles as imprecise data items. The uncertainty pdf is assumed to be uniform. We also consider Gaussian uncertainty pdf in some cases.

Each C-PNN is characterized by two parameters: Threshold (T) and Tolerance (Δ), with default values 0.3 and 0.01 respectively. Moreover, $T > 0.1$, since we assume a C-PNN user is not interested in very small qualification probabilities. The query points are randomly generated inside the data space. Each experimental result is obtained as an average of over 100 queries.

We compare three strategies of evaluating qualification probabilities. The first method, called *Basic*, uses numerical integration directly. The second one, called *VR*, probabilistic verifiers, as well as incremental refinement (to handle tuples that remain unclassified after verification). The third one, called *Refine*, skips verification and performs incremental refinement directly. All these strategies assume that filtering has been used to create the candidate set C for them.

3.4.2 Results

1. Cost of the Basic Method. We first compare the time spent on the

Basic with filtering. Figure 3.9 shows that the fraction of total time spent in these two operations on synthetic data sets with different candidate set sizes. As the size of the total data set $\mathbb{O}$ increases, the time spent on the *Basic* solution increases more than filtering, and so its running time starts to dominate the filtering time data set size is larger than 5000. As we will show next, other methods can alleviate this problem.

2. Effectiveness of Verification. Now we compare the computation time required by the three methods: *Basic*, *Refine*, *VR* with different values of T. As shown in Figure 3.10, both *Refine* and *VR* perform better than *Basic*. When $T = 0.3$, the time cost for *Refine* and *VR* is about 80% and 16% of that of *Basic*. Also, *Basic* is insensitive to the change of T. However, both *Refine* and *VR* benefit from the increase of T. This is because they can exploit the threshold constraint and finish query processing once no 'unknown' tuples remain in C. For large values of T, most tuples can be identified as 'rejected' quickly when their $p_i.u$'s are detected to be smaller than T.

Moreover, *VR* performs better than *Refine*. For example, *VR* is five times faster than *Refine* at $T = 0.3$, and 40 times faster at $T = 0.7$! To understand why, let us look at Figure 3.11, which shows the time breakdown of the three phases in the evaluation process, i.e., filtering, verification, and refinement. The filtering time is fixed regardless of the threshold. The time spent on verification is just 1ms on average, but has a positive effect on refinement. As T increases, more queries are completed during verification, and so the amount of refinement time is reduced. In fact, when $T > 0.3$, no more tuples need to be refined. Thus, the use of verifiers allows *VR* to shorten its refinement time significantly, resulting in a higher performance.

We now examine the effect of using a Gaussian distribution as the uncertainty pdf for each object. Each Gaussian pdf, has a mean at the center of

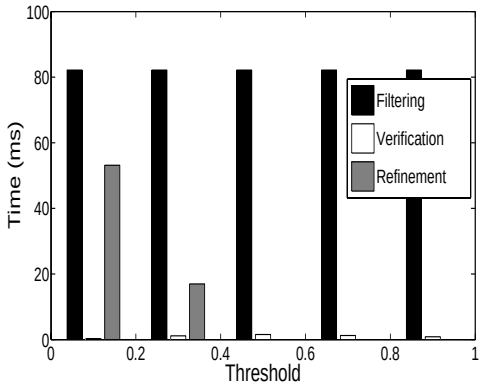


Figure 3.11: Time Breakdown.

its range, and a standard deviation of 1/6 of the width of the uncertainty region. Since we approximate a Gaussian pdf as a histogram, we first conduct a sensitivity test to see how precise a Gaussian pdf should be represented (in terms of the number of histogram bars). We notice that the result qualification probabilities become stable when more than 300 histogram bars are used. Hence we consider this qualification probability as the "true" answer. In Figure 3.12, we can see that as more histogram bars are used, a smaller relative error (compared with the true probability) is attained. However, it also takes a longer time to obtain the answer. When a pdf is represented as 20 histogram bars, the average relative error is less than 5%, and the time required is about 10 seconds. Hence, we adopt this setting for Gaussian pdf.

Figure 3.13 shows the probability computation time for Gaussian pdf. Again, *VR* outperforms the other two methods. The saving is more significant than when uniform pdf is used. This is because the relatively more expensive evaluation of Gaussian pdf can be effectively avoided by the verifiers. Hence, our method also works well with Gaussian pdf.

3. Effectiveness of Probabilistic Verifiers. We now compare the performance of different verifiers presented in Section 3.3.

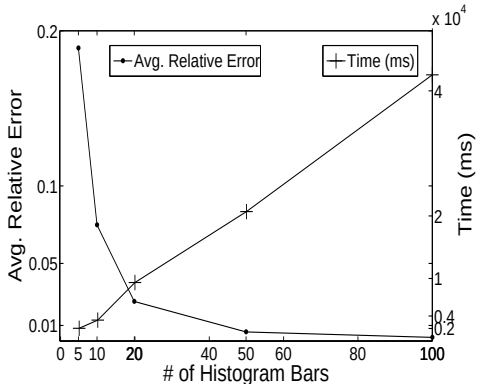


Figure 3.12: Relative error vs. pdf precision.

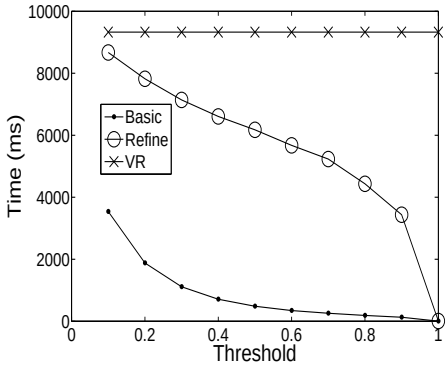


Figure 3.13: Gaussian pdf.

Figure 3.14 shows the fraction of tuples labeled *unknown* that are left in C after the execution of each verifier, according to the order shown in Figure 3.4. This fraction reflects the amount of verification/refinement work that needs to be done. At $T = 0.1$, about 75% *unknown* tuples are in C after RS finishes. Then, 7% more tuples are removed by L-SR. After U-SR is completed, there are only 15% *unknown* tuples. It is worth notice that RS and U-SR work better under large threshold values. This is because RS and U-SR are responsible for reducing the value of $p_i.u$. When T is larger, it is easier for $p_i.u$ to be smaller than T, allowing o_i to be labeled as *fail*. L-SR works better in lower threshold values, as can be seen from the gap between the RS and the L-SR curve. Notice that L-SR tends to increase $p_i.l$. When T is small, $p_i.l$ has a better chance of being higher than it; correspondingly, o_i is easier to be labeled as *satisfy*.

4. Effect of Tolerance. In the final experiment, we study the effect of tolerance on the verifiers. We measure the fraction of queries completed during the verification stage. As shown in Figure 3.15, when Δ increases from 0 to 0.2, more queries will be stopped. When $\Delta = 0.16$, about 10% more queries will be completed than when $\Delta = 0$. Thus, the introduction of

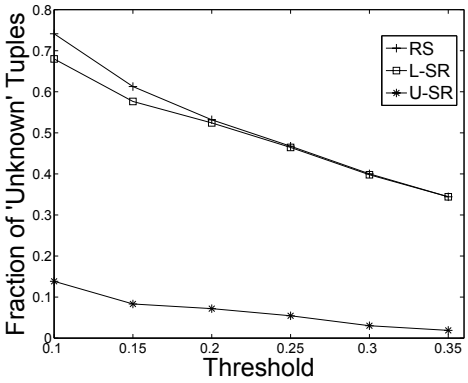


Figure 3.14: Effectiveness of Verifiers.

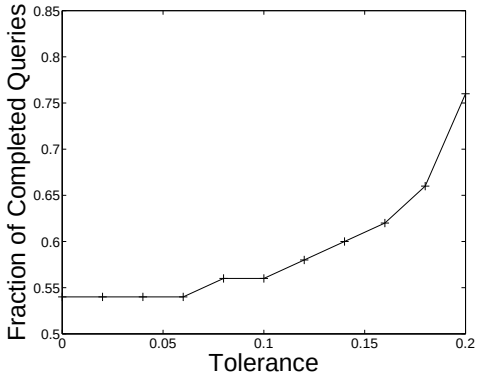


Figure 3.15: Effect of Δ.

tolerance can improve the performance of verification.

3.5 Chapter Summary

Uncertainty management is an emerging and important topic, and has recently attracted a lot of research interest. We identified the problem of high computational complexity for PNN evaluation, and proposed to study the C-PNN, a variant of PNN. We developed probabilistic verifiers to reduce the chance of calculating qualification probabilities, and the incremental refinement algorithm for facilitating probability computation. As shown by our experiments, the overall query performance is significantly improved by the use of these verifiers. Moreover, they only introduce a small time/space overhead, and are scalable with large data set size.

In the future, we will incorporate probabilistic verifiers in a system prototype (e.g.,[1]), and consider to extend our techniques to support other variants of the NN queries. For example, instead of threshold, query issuers may want to know the objects with k highest qualification probabilities, i.e. *top-k* queries [10, 77]. Compared to C-PNN, the *top-k* queries place more emphasis on the relative importance between results, and discard the abso-

lute probabilistic confidence levels. Although these two semantics are quite different, it is still possible to extend our proposed techniques for solving *top-k* queries. Notice that the key point for answering *top-k* queries is to distinguish the two objects whose qualification probabilities are the k^{th} and $k+1^{th}$ highest respectively. With probabilistic verifiers, we can find the lower and upper bounds of all objects' qualification probabilities. Hence possibly we may be able to obtain the *top-k* set based on the probability bounds. If these bounds are not precise enough, we can further refine the qualification probability bounds by incremental refinement until the *top-k* set is found. We will conduct this extension in our follow-up works.

4 Probabilistic k-Nearest-Neighbor Queries

This chapter addresses the *Probabilistic k-Nearest-Neighbor Query (k-PNN)*, which computes the probabilities of sets of k objects for being the closest to a given query point. The evaluation of this query can be both computationally- and I/O- expensive, since there is an exponentially large number of k object-sets, and numerical integration is required. Often a query issuer only needs answers that have sufficiently high confidences. We thus propose the Probabilistic Threshold k-Nearest-Neighbor Query (T-k-PNN), which returns sets of k objects whose probabilities for satisfying the query are higher than some threshold T. Three steps are proposed to handle this query efficiently. In the first stage, objects that cannot constitute an answer are *filtered* with the aid of a spatial index. The second step, called *probabilistic candidate selection*, significantly prunes a number of candidate sets to be examined. The remaining sets are sent for *verification*, which derives the lower and upper bounds of answer probabilities, similarly as the verifiers do in Section 3. By *verification*, a candidate set can be quickly decided on whether it should be included in the answer. We also examine spatially-efficient data structures that support these methods. Our solution can be applied to uncertain data with arbitrary probability density functions. We have also performed extensive experiments to examine the effectiveness of our methods.

4.1 Introduction

As stated in previous sections, uncertainty is inherent in many emerging applications. To deal with the increasing needs of managing data uncertainty and providing high-quality services, researchers have recently proposed the use of probabilistic queries, which produces answers with probabilistic and

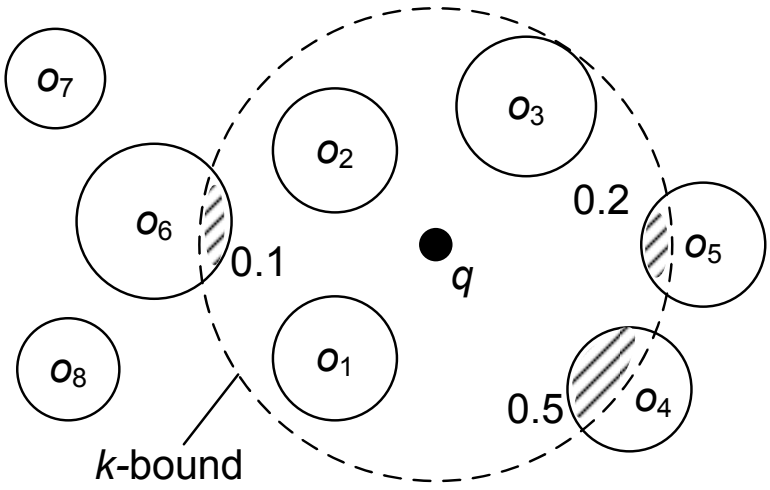


Figure 4.1: Probabilistic k-NN Query (k-PNN) with $k = 3$.

statistical guarantees [6, 23, 30, 3]. In this chapter, we study the *Probabilistic k-Nearest Neighbor Query* (k-PNN) for databases with attribute uncertainty. This query returns the non-zero qualification probability of each set of k objects for being the nearest neighbor of a given point q. Given an uncertain database D of n uncertain objects, where $D = \{o_1, ..., o_n\}$, the k-PNN query can be defined as follows:

Definition 4.1 *A* **Probabilistic k-NN Query (k-PNN)** *returns a list of answers $\{(S, p(S))\}$, where S is a subset of D of cardinality k, and $p(S)$ is the non-zero probability that S consists of the k nearest neighbors of q.*

Figure 4.1 shows an example of k-PNN, evaluated over eight uncertain objects $(o_1, \ldots, o_8)$. If $k = 3$, then the query returns a set of 3-ary tuples, together with their chances for satisfying the query. In this example, $\{o_1, o_2, o_5\}$ and $\{o_1, o_2, o_3\}$ have qualification probabilities of 0.05 and 0.3 respectively. Notice in this definition, the number of k-subsets that satisfy the query may be exponential, and it may be necessary to have additional constraints (e.g.,

return objects whose probabilities are higher than some threshold) in order to limit the size of the answer set.

The k-PNN can be considered as a version of the k-nearest neighbor query (k-NN) evaluated on uncertain data. The k-NN query has been widely used in different applications, including location-based services [50], natural habitat monitoring [33], network traffic analysis [39], knowledge discovery [37], and CAD/CAM systems [54]. For example, in mobile e-commerce, a driver is supplied with the location information of the nearest gas stations. A CAM system uses k-NN queries to discover similar patterns over multi-dimensional data obtained from sensors installed in production lines [54]. A k-NN query can also be used to answer other ranking queries, such as k-min and k-max queries. For one-dimensional data, a k-min (k-max) query can be considered as a k-NN query by setting q to $-\infty$ (respectively $+\infty$). Such queries can be used in scientific monitoring applications to answer questions like: "What are the k bird nests that yield the highest temperature?" [33].

Most works about k-NN queries assume that the data being queried is precise. However, as we can see from the applications mentioned before, data on which k-NN is evaluated (e.g., locations of moving objects and sensor values) are often imprecise.

The methods for processing 1-PNN queries in Section 3 are not readily used by a k-PNN (with $k > 1$) for three reasons. First, the evaluation of k-PNN faces the additional problem of examining a large number of k-subsets. To handle this problem, we develop new methods to significantly reduce the number of candidate k-subsets. Secondly, the probability bound verification is designed for 1-PNN queries only. We develop new lower/upper bound computation methods for k-PNN queries. Thirdly, the solution in Section 3 can only be used to handle distance pdfs of the candidate objects represented

as arbitrary histograms. The techniques in this chapter, on the other hand, are not restricted to histogram pdfs.

Computing a k-PNN is usually more complex than its precise counterpart. Consider, for example, the computation of the probability that $\{o_1, o_2, o_5\}$ are the three closest neighbors to q in Figure 4.1. Since each object's value is not exactly known, we need to consider the values in its uncertainty region. Moreover, the qualification probability of $\{o_1, o_2, o_5\}$ depends not just on the three objects' values, but also on the relative values of other objects (e.g., o_3). If there is a chance that two objects have the same distance from q (e.g., o_2 and o_3), then their pdfs must be considered in order to derive the probabilities. The problem is further aggravated by the large number of combinations of objects. For example, for a 3-PNN evaluated over eight objects in Figure 4.1, we may have to compute the probabilities for $C_3^8 = 56$ possible answers. The number of answers that satisfy the query can also be exponential. Clearly, we need better semantics and methods to handle this query.

4.1.1 Solution Overview

We observe that a query user may not always be interested in getting the precise probability values. He/She may only require answers with confidence that meets some predefined condition. For example, a user may only require answers with confidence higher than some fixed value. In Figure 4.1, for instance, if an answer with at least 20% probability is needed, then the sets $\{o_1, o_2, o_3\}$ and $\{o_1, o_2, o_4\}$ would be the only answers. We term the variant of k-PNN with a probability threshold constraint, T (e.g., 20%), as

the *Probability Threshold k-Nearest-Neighbor Query* (or T-k-**PNN** in short). The threshold constraint allows the user to control the desired confidence required in a query answer. In Figure 4.1, for example, a 0.2-3-PNN returns $\{o_1, o_2, o_3\}$ and $\{o_1, o_2, o_4\}$ as the query answer. Such a query answer also allows a user to extract some useful information (e.g., o_1 and o_2 appear in both 3-subsets in this example). Notice that with a moderate value of T, the number of k-subsets returned is quite small in practice. For instance, in our experiments, at $T = 0.1$, two k-subsets are returned on average.

Moreover, we present three methods to efficiently process a T-k-PNN query. The first method, called *k-bound filtering*, effectively removes all objects that have no chance to be a query answer. Let us consider Figure 4.1 again, which shows the "k-bound" (as a dotted circle centered at q) that completely encloses the three objects o_1, o_2 and o_3. The radius of the 3-bound is defined as the third minimum of the maximal distances of the objects from q (in this example, the maximum distance of o_3 from q). With the k-bound, objects o_7 and o_8 can be pruned immediately, since they have no chance to overtake any of the objects o_1, o_2 or o_3 to become part of the answer to the 3-PNN query. Generally, with the k-bound, a lot of objects can be removed, and as we show in the chapter, its usage can be easily leveraged to a spatial index (e.g., R-tree). For convenience, we call the objects that are not pruned by the k-bound filtering (i.e., those that overlap the k-bound) the *candidate objects*.

After k-bound filtering, we still need to consider the k-subsets of the candidate objects. In Figure 4.1, for instance, $C_3^6 = 20$ of sets of cardinality 3 may need to be considered. To further reduce the search space, we propose the second method, namely *Probabilistic Candidate Selection* (or **PCS**), which can efficiently detect k-subsets (i.e., subsets of database D with

cardinality k) whose qualification probabilities are less than T, also called *unqualified k-subsets*. While k-bound filtering utilizes distance information for pruning, the PCS makes use of the probability information of uncertain data to remove unqualified k-subsets. The rationale behind PCS is that given the probability of a candidate object that lies within the k-bound (called *cutoff probability*), the qualification probability of a k-subset must be lower than the product of the cutoff probabilities of its subsets. In Figure 4.1, for example, the cutoff probabilities of o_4, o_5, and o_6 are 0.5, 0.2 and 0.1 respectively (shown as the shaded area). The qualification probability of the 3-subset $\{o_2, o_4, o_5\}$ must be lower than the product of the cutoff probabilities of $\{o_4, o_5\}$, or $0.5 \times 0.2 = 0.1$. If $T = 0.2$, then $\{o_2, o_4, o_5\}$ can be pruned. Based on this useful fact, the PCS algorithm constructs k-subsets by growing the list of the i-subsets with respect to i (where $i = 1, 2, .., k$). At each iteration i, the product of the cutoff probabilities of each i-subset are checked on whether it is less than T, and the i-subset is pruned if that is true. In addtion, we also design a technique, called "seed-pruning", to further improve the performance of PCS by removing the unqualified k-subsets with the "seeds" – objects that are located within the k-bound. Moreover, an efficient data compression method that suppresses the amount of intermediate storage overhead required by PCS is presented. Our experiments show that PCS reduces a significant portion of k-subsets to be examined.

The third method, called *verification*, is useful for handling k-subsets that are not filtered by the previous two methods. This technique determines whether a k-subset is a query answer, by making use of the uncertainty pdf of objects returned by k-bound filtering. We propose two kinds of verification: *lower-bound* and *upper-bound* verification, which quickly computes the lower and upper bounds of qualification probabilities of k-subsets. These

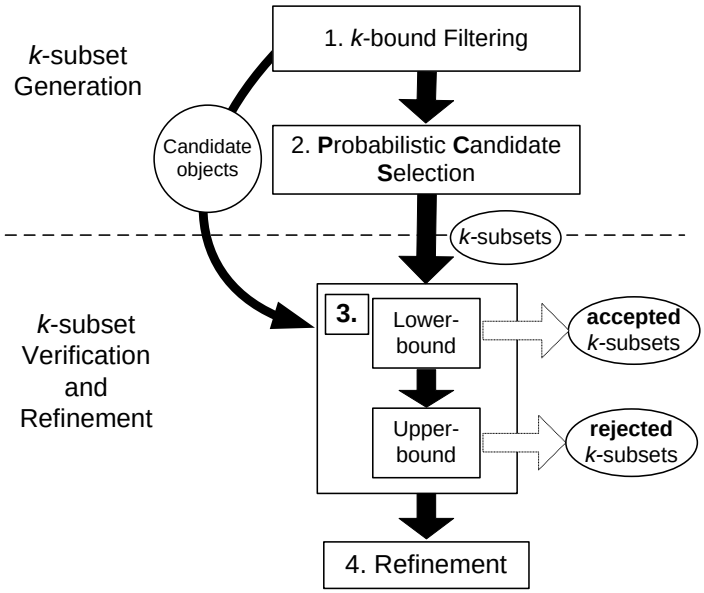


Figure 4.2: Solution Framework of T-k-PNN.

bounds can then be used to determine how the k-subset should be handled. For example, a k-subset can be removed if its upper bound probability is smaller than T; it should be included in the query answer if its lower bound probability is higher than T. We will show the detailed design, complexity analysis, as well as correctness proofs for these methods.

Figure 4.2 depicts the framework of our solution, which consists of four steps. First, the k-bound filtering removes objects that must not be part of the k-nearest neighbor of q. All candidate objects are then passed to the PCS, which derives k-subsets based on the cutoff probability information. Next, the lower (upper) bounds of the qualification probabilities of the k-subsets are used to accept (reject) the k-subsets. Those that still cannot be determined are sent for *refinement*, whose exact probabilities are computed. While refinement is expensive, it can utilize the information generated during verification. Thus this is still faster than computing the qualification

probability of k-subsets directly.

To summarize, we propose a computationally- and I/O- efficient solution for evaluating a T-k-PNN query. Our solution reduces I/O overhead by using a spatial index (e.g., R-tree) to prune away a large number of objects. To alleviate the large computational overhead, we propose PCS for reducing the number of k-subsets to be examined, as well as verification/refinement for avoiding exact probability computation. We propose a framework to connect these techniques in order to provide an efficient solution. We further investigate storage-efficient data structures to support our solution. Our experiments show that our approach can significantly improve the performance of query evaluation. For example, at $T = 0.1$ and $k = 5$ the time required by our method is only 1.6% of the time needed by calculating qualification probabilities directly.

The rest of this chapter is organized as follows. We present the formal semantics of the T-k-PNN, and our solution framework in Section 4.2. Section 4.3 explains the filtering and the probabilistic candidate selection process. The details of verification and refinement are developed in Section 4.4. We present the experimental results in Section 4.5, and summarize this chapter in Section 4.6.

4.2 Preliminaries

We now present the semantics of the T-k-PNN query (Section 4.2.1). Then we explain a simple solution for this query (Section 4.2.2).

4.2.1 Definition of T-k-PNN

Let $p(S) \in [0, 1]$ be the probability that the elements of a k-subset S are the k nearest neighbors of query point q (i.e., qualification probability). Then, a

T-k-PNN can be defined as follows.

Definition 4.2 *A* **Probability Threshold k-NN Query (T-k-PNN)** *returns a set* $\{S|S \subseteq D \wedge |S| = k\}$ *such that* $p(S) \geq T$, *where* $T \in (0, 1]$.

We call T the *threshold* parameter. A k-subset S is allowed to be returned as an answer if its qualification probability is not less than T. Compared with k-PNN (Definition 4.1), this query does not return the actual qualification probability of S to the user. Also, we can further use other constraints (e.g., the maximum number of answers) to limit the number of k-subsets returned to the user.

4.2.2 Basic Evaluation of T-k-PNN

Let us now present a simple solution for answering the T-k-PNN, which forms the basis for further discussions. This method utilizes the probability distribution of each object's distance from q. Formally, let $R_i \in \Re$ be the absolute distance of an uncertain object o_i from q. That is, $R_i = |o_i - q|$. We assume that R_i takes on a value $r \in \Re$.

Based on the distance pdf and cdf of each object (please refer to Definition 3.4 in page 30), the qualification probability of a k-subset S (i.e., $p(S)$) can be computed. The probability, $p(S)$, is then used to compare against T; if $p(S) \geq T$, then S becomes an answer. Now let us take a look at how $p(S)$ is computed:

$$p(S) = \sum_{o_i \in S} \int_0^{+\infty} d_i(r) \prod_{o_j \in S \wedge o_j \neq o_i} D_j(r) \prod_{o_h \in D-S} (1 - D_h(r)) dr \tag{4.1}$$

To understand Equation 4.1, observe that for S to be a query answer, the distance of any object o_h (where $o_h \notin S$) from q must be greater than that

of o_i (where $o_i \in S$). Now, at distance r, the pdf that object $o_i \in S$ has the k-th shortest distance from q is the product of the following factors:

- the pdf that o_i has a distance of r from q, i.e., $d_i(r)$;
- the probability that all objects in S other than o_i have shorter distances than r, i.e., $\prod_{o_j \in S \wedge o_j \neq o_i} D_j(r)$; and
- the probability that objects in $D - S$ have longer distances than r, i.e., $\prod_{o_h \in D-S} (1 - D_h(r))$.

The integration function in Equation 4.1 is essentially the product of the above three factors. By integrating this function over $(0, +\infty)$, we obtain the probability that S contains the k nearest neighbors with o_i as the k-th nearest neighbor. Finally, by summing up this probability value for all objects $o_i \in S$, Equation 4.1 is obtained.

Equation 4.1 is inefficient to evaluate. First, the distance pdf and cdf of each object has to be computed. Secondly, Equation 4.1 involves costly numerical integration, and has to be performed over a large range. Thirdly, the probability of each k-subset S has to be computed, and the number of these k-subsets is exponential. However, we found that T-k-PNN can be handled in a better way. Specifically, later, in Section 4.3, we exploit the extent of the objects' uncertainty regions and probability threshold to significantly prune the number of k-subsets to be examined. Then, in Section 4.4, we derive the lower and upper bounds of k-subsets' qualification probabilities, so that the decision of whether a k-subset should be accepted as a query answer can be made without computing its actual probability.

4.3 Generating k-subsets

In this section, we examine two efficient methods for generating k-subsets that can potentially satisfy the T-k-PNN queries. Section 4.3.1 discusses the design and implementation of k-bound filtering. We then present the Probabilistic Candidate Selection (PCS) process in Section 4.3.2. We investigate a spatially-efficient compression method for supporting PCS in Section 4.3.3.

4.3.1 k-bound Filtering

Given a query point, a naive solution is to enumerate all possible combinations of sets of size k from the uncertain database D and compute their probabilities according to Equation 4.1. If the probability is greater than T, the set will be returned as a result. Clearly, this is an inefficient approach with respect to computation and I/O costs. In fact, given a T-k-PNN, we can first utilize the distance information of the objects to prune those that are not qualified for the answers. Specifically, we propose an efficient filter based on the k-th minimum of maximal distance f_k, called k-bound filter, to remove objects that have zero probability to be the answers of T-k-PNN. The rationale behind the k-bound filter is stated in the following Lemma.

Lemma 4.1 *Given an object $o_i \in D$, a query point q, if $min(R_i) > f_k$, then o_i will not appear in any answer data set, where f_k is the k^{th} minimum of maximal distance (k-bound) among all R_j's ($j = 1, ..., n$).*

Proof : If $min(R_i) > f_k$ holds, o_i cannot belong to the k nearest neighbors of q since there always exist at least k objects with distances smaller than or equal to f_k. Therefore, no answer set S contain such an o_i according to the T-k-PNN definition. In other words, the fact that S is the k nearest neighbors of q implies $min(R_i) \leq f_k(\forall o_i \in S)$. □

Lemma 4.1 offers a filtering method based on the k-bound. Consider again the example in Figure 4.1, with k-bound filtering, objects o_7 and o_8's minimum distances to q are larger than the bound, and so they can be excluded for further consideration. The rest of the objects that overlap with or are contained in the k-bound are used to generate k-subsets. Another advantage of k-bound filtering is, after filtering, probability computation is easier (compared to Equation 4.1), since the integration range $[0, \infty]$ is reduced to $[0, f_k]$ for numerical integration.

```
   input  : R-tree I constructed over D
   input  : q: the query point
   input  : k
   output: the candidate object set C and f_k
 1 initialize min-heap H accepting entries in the form (v, key);
 2 C ← ∅;
 3 f_k ← +∞;
 4 insert (root(I), 0) into heap H;
 5 while H is not empty do
 6     (v, key) ← de-heap H;
 7     if key < f_k then
 8         if v is an uncertain object then
 9             insert v into C;
10         else
11             for each entry v_i contained in v do
12                 compute key ← min(dist(q, v_i));
13                 if key < f_k then
14                     insert(v_i, key) to H;
15                     update f_k according to key;
16 return C, f_k;
```

Algorithm 1: k-bound_Processing

The performance of k-bound filtering can be sped up with the help of a spatial index structure. In this work we index the uncertainty region of

each data object in the R-tree [45], on which the k-bound filtering can be conducted. The reason we choose R-tree is due to its popularity. However other spatial index structures can also be used. The R-tree recursively groups uncertain data objects with *minimum bounding rectangles* (MBRs) until one final node (root) is obtained. The process of filtering over R-tree is detailed in Algorithm 1 (k-bound_Processing). The algorithm maintains a minimum heap $\mathcal{H}$ which contains the entry of form (v, key), where key is the $min(dist(q, v))$. $\mathcal{H}$ is first initialized (line 1). The candidate object set is emptied and f_k is set as infinity (lines 2 and 3). Then, the root node is loaded and stored in $\mathcal{H}$ (line 4). Each time we pop out an entry (v, key) from heap $\mathcal{H}$ (line 6), and check whether key is smaller than f_k (line 7). If the answer is negative, this entry is discarded (Lemma 4.1). Otherwise, we then check whether v is a leaf node (line 8). If the answer is yes, we insert this founded candidate object into C (line 9). If v is an intermediate node, for each entry v_i in v, we compute its minimum distance from q (lines 11 and 12). If this minimum distance is also smaller than f_k, we insert v_i into the heap $\mathcal{H}$ and update f_k if necessary (lines 13 to 15). This process repeats until the queue is empty.

4.3.2 Probabilistic Candidate Selection

After k-bound filtering, assume we obtain $m \in [k, n]$ objects, i.e. $C = \{o_1, ..., o_m\}$, there could still be C_k^m possible k-subset answers. Directly computing these answer sets will result in exponential cost in both memory and computation. In fact, it is not necessary to generate all the k-subsets. In this section, we propose a candidate set generation method based on the probability information, namely *probabilistic candidate selection* (PCS). Specifically, we make use of the probability of an object that lies within the k-bound, called *cutoff probability* (CP) (shown in Figure 4.3(a)) and the fact that the

qualification probability of a k-subset must be less than the product of the cutoff probabilities of its members. The following lemma states this fact.

Lemma 4.2 $p(S) \leq UBProb(S')$, $\forall S' \subseteq S$,
where $UBProb(S') = \prod_{o_i \in S'} Pr(R_i \leq f_k)$

Proof : According to Lemma 4.1, the fact "S contains k nearest neighbors" requires that all member objects of S to have distances from q not larger than f_k. That means $p(S)$ must not be larger than $\prod_{o_i \in S} Pr(R_i \leq f_k)$, i.e. the product of the cutoff probabilities of its member objects. Since the cutoff probabilities are always smaller than or equal to 1, $UBProb(S')$ gives an upper bound of $\prod_{o_i \in S} Pr(R_i \leq f_k)$. Thus Lemma 4.2 is proved. □

```
   input  : C = {o_1, ..., o_m}, q, k, T
   output: C_k: the set of candidate object sets
 1 C_1 ← {{o_1}, ..., {o_m}};
 2 for i ← 1 to k − 1 do
 3     C_{i+1} ← ∅;
 4     for each S ∈ C_i do
 5         z ← max_{o_i ∈ S}(i) ;
 6         for j ← z + 1 to c do
 7             if o_j ∉ S then
 8                 S' ← S ∪ {o_j};
 9                 if S' ∉ C_{i+1} then
10                     if UBProb(S') ≥ T then
11                         C_{i+1}.add(S')
12                     else
13                         break;
14 return C_k
```

Algorithm 2: Prob_Cand_Sel

Based on the cutoff probability of each candidate object within the k-bound, the PCS algorithm constructs $k-$subsets by growing the list of i-

1-subset	CP
$\{o_1\}$	1
$\{o_2\}$	1
$\{o_3\}$	1
$\{o_4\}$	0.5
$\{o_5\}$	0.2
$\{\mathbf{o_6}\}$	**0.1**

(a) Round 1

2-subset	CP
$\{o_1, o_2\}$	1
$\{o_1, o_3\}$	1
$\{o_1, o_4\}$	0.5
$\{o_1, o_5\}$	0.2
$\{o_2, o_3\}$	1
$\{o_2, o_4\}$	0.5
$\{o_2, o_5\}$	0.2
$\{o_3, o_4\}$	0.5
$\{o_3, o_5\}$	0.2
$\{\mathbf{o_4, o_5}\}$	**0.1**

(b) Round 2

3-subset	CP
$\{o_1, o_2, o_3\}$	1
$\{o_1, o_2, o_4\}$	0.5
$\{o_1, o_2, o_5\}$	0.2
$\{o_1, o_3, o_4\}$	0.5
$\{o_1, o_3, o_5\}$	0.2
$\{\mathbf{o_1, o_4, o_5}\}$	**0.1**
$\{o_2, o_3, o_4\}$	0.5
$\{o_2, o_3, o_5\}$	0.2
$\{\mathbf{o_2, o_4, o_5}\}$	**0.1**
$\{\mathbf{o_3, o_4, o_5}\}$	**0.1**

(c) Round 3

Figure 4.3: Step-by-step generating candidate subsets based on CP

subsets with respect to i (where $i = 1$ to $k - 1$). The steps of PCS are listed in Algorithm 2. First, the algorithm generates 1-subsets based on the candidate set C (line 1). Then, the $(i+1)$-subsets are generated by unioning i-subsets and C (lines 2 to 13). The value of $UBProb(S')$ could be obtained by Lemma 4.2 (line 10). All those subsets with $UBProb(S')$ smaller than the threshold will be pruned (line 11). Therefore, many intermediate subsets are pruned, and the number of k-subsets will be greatly reduced. When we extend S to S' by adding o_j, and find that S' should be pruned, then it is no need to check the extensions with $o_{j+1}, ..., o_m$ (line 13), since the data objects are sorted in descending order of their cutoff probabilities.

Figures 4.3(a)-(c) show an example of generating candidate $k-$subsets based on the cutoff probability of each candidate object within the k-bound. Figure 4.3(a) lists the cutoff probability (CP) of each object. We can safely remove candidate object o_6 since its CP is less than the threshold $T = 0.2$. Then, in the second round, as shown in Figure 4.3(b), the subset $\{o_4, o_5\}$ can be removed. Similarly, in the third round (Figure 4.3(c)) the candidate subsets $\{o_1, o4, o_5\}$, $\{o_2, o_4, o_5\}$ and $\{o_3, o_4, o_5\}$ can be safely removed.

In the previous discussions, in each round i, we have used CP and T to determine whether the generated i-subset should be kept for further extension. Here we propose an enhancement that utilizes the i-th minimum maximum distance (i.e., f_i) to further remove the unqualified subsets generated in each round. We can obtain these f_i values by slightly changing the Algorithm k-bound_Processing to return all the f_i values. Next, we suppose all candidate objects have been sorted in ascending order of their maximum distances from q. We put the objects with the k lowest values of maximum distance into an array called *seeds*, and derive the following lemma.

Lemma 4.3 *If the lower bound of R_j of data object o_j is larger than f_i (i-th minimum maximum distance of seeds[i]), any k-subset S containing o_j cannot be the answer to the T-k-PNN if $\exists o_t \in \{seeds[1], ..., seeds[i]\}$ and $o_t \notin S$.*

Proof : The fact that "the lower bound of R_j is larger than the f_i" implies that all objects in $\{seeds[1], ..., seeds[i]\}$ must have shorter distances than R_j. Therefore, if o_j happens to be inside a probabilistic k-nearest neighbors answer set, say S, all objects in $\{seeds[1], ..., seeds[i]\}$ must also be contained in S. □

Lemma 4.3 indicates that a qualified k-subset should contain some specific seeds to become a valid result. This rule can help us prune many unqualified subsets without estimating their qualification probabilities. The detailed steps are listed in Algorithm 3 (Seed_Pruning). In order to use seed pruning method to remove unqualified intermediate subsets generated in each round of PCS, we can invoke Algorithm Seed_Pruning immediately after line 4 of Algorithm Prob_Can_Sel.

```
input  : seeds, S and f_i, ..., f_k.
output: A boolean value indicating whether S is a possible result.
1 for each o_j ∈ S do
2     if min(R_j) < f_1 then
3         γ ← 0;
4     else
5         γ ← the largest i satisfying min(R_j) ≥ f_i;
6     if γ > 0 then
7         if not {seeds[1], ..., seeds[γ]} ⊆ S then
8             return False
9 return True
```

Algorithm 3: Seed_Pruning

4.3.3 A Storage-Efficient Compression Method

The PCS algorithm can be quite expensive in terms of memory consumption, since in each round i, we have to store all the i-subsets whose CPs are greater than T, and the number of such i-subsets could be exponentially large.

To reduce the memory cost we propose a simple but effective compression method. We first present our observations on the i-subsets generated by the PCS algorithm which forms the basis of our discussions. As shown in Figures 4.3(a)-(c), the elements of the generated subsets in each round using Algorithm 2 ($Prob\_Cand\_Set$) are sorted in the descending order of their CPs. In addition, we can also find that many subsets of the same size share a common prefix. For example, in Figure 4.3(b), the first four 2-subsets share the common prefix $\{o_1\}$. Similarly, in Figure 4.3(c), the first three 3-subsets share the common prefix $\{o_1, o_2\}$. Based on these observations, we propose to compress the subsets of the same size that share a common prefix. Specifically, for the subsets of the same size, we store the common prefix of the subsets and the last element of the subset that has the minimum product

of cutoff probability greater than T. We also call this element a *boundary element*. For example, given the first four 2-subsets in Figures 4.3(b), after compression, we only store $\{o_1, o_5\}$ as shown in the first entry of compressed storage in Figure 4.4(b). In this entry $\{o_1\}$ is the common prefix and o_5 is the last element of subset $\{o_1, o_5\}$, whose subset has the minimum product probability among first four 2-subsets in Figure 4.3(b). As shown in Figure 4.4(b), in addition to the compressed item $\{o_1, o_5\}$, we also store the product probability of the common prefix, here it is $\{o_1\}$'s CP, which is 1. Thus, in our compression scheme, for each compressed entry, we store the common prefix, the boundary element, and the product probability of the prefix. As another example, the first three 3-subsets in Figure 4.3(c) are compressed into $\{o_1, o_2, o_5\}$, as shown in Figure 4.4(c). The whole compressed entry is $\{\{o_1, o_2, o_5\}, 1\}$, where 1 is the product probability of prefix $\{o_1, o_2\}$. Figures 4.4(a)-(c) show the whole compressed results of subsets in Figures 4.3(a)-(c). Note that entries in bold fonts of Figures 4.3(a)-(c) are unqualified entries.

Whenever it is necessary to decompress an compressed entry, we can generate the uncompressed subsets by appending all the possible elements starting from the *immediate successor* of the last element in the prefix to the bounding element. Let us use Figure 4.3(b) as an example. Given the compressed entry $\{\{o_1, o_5\}, 1\}$, the prefix is $\{o_1\}$, the bounding element is o_5, the immediate successor element of o_1 is o_2, as all the candidate objects are sorted in the descending order according to their CPs. Then the decompressed set is $\{o_1, o_2\}$, $\{o_1, o_3\}$, $\{o_1, o_4\}$, and $\{o_1, o_5\}$. The corresponding CP of each decomposed subset is the product of the compressed entry's CP (now is 1) and the appended element's CP. Similarly, for the compressed entry $\{\{o_1, o_2, o_5\}, 1\}$ shown in Figure 4.4(c), o_3 is the immediate successor of last

Size-1 Set	CP
$\{o_1\}$	1
$\{o_2\}$	1
$\{o_3\}$	1
$\{o_4\}$	0.5
$\{o_5\}$	0.2

(a) Round 1

Size-2 Set	CP
$\{o_1, o_5\}$	1
$\{o_2, o_5\}$	1
$\{o_3, o_5\}$	1

(b) Round 2

Size-3 Set	CP
$\{o_1, o_2, o_5\}$	1
$\{o_1, o_3, o_5\}$	1
$\{o_2, o_3, o_5\}$	1

(c) Round 3

Figure 4.4: Compressed candidate subsets based on CP

element in prefix $\{o_1, o_2\}$, so we can generate $\{o_1, o_2, o_3\}$, $\{o_1, o_2, o_4\}$, and $\{o_1, o_2, o_5\}$, the corresponding CPs for these decompressed entries are: $1 * 1$, $1 * 0.5$, and $1 * 0.2$ respectively. Our experiments show that this compression scheme reduces the storage required in this phase significantly.

4.4 Verification and Refinement

Based on the k-subsets generated by PCS, we now present efficent techniques for handling the k-subsets. These techniques are based on our solutions for answering C-PNN (Section 3). We summarize the general process and highlight the extensions in this section. For details, please refer to Section 3.3.

Partitions. Like the *subregions* in Section 3, the range between the closest distance of all objects from q and the point f_k, i.e. the *k-th* minimum of maximal distance, is subdivided into non-overlapping fragments called *partitions*. Figure 4.5(a) illustrates three distance pdfs with respect to q, where $k = 2$. As illustrated, the range between e_1 (i.e., the closest distance of all objects from q) and the point f_2 obtained from k-bound filtering (i.e., e_5) is subdivided into non-overlapping fragments called *partitions*. We call each fragment P_j, where P_j is embraced by two end-points, namely, e_j and e_{j+1}, circled in the figure. In this example, there are four partitions, e.g., $P_1 = [e_1, e_2]$, $P_2 = [e_2, e_3]$.

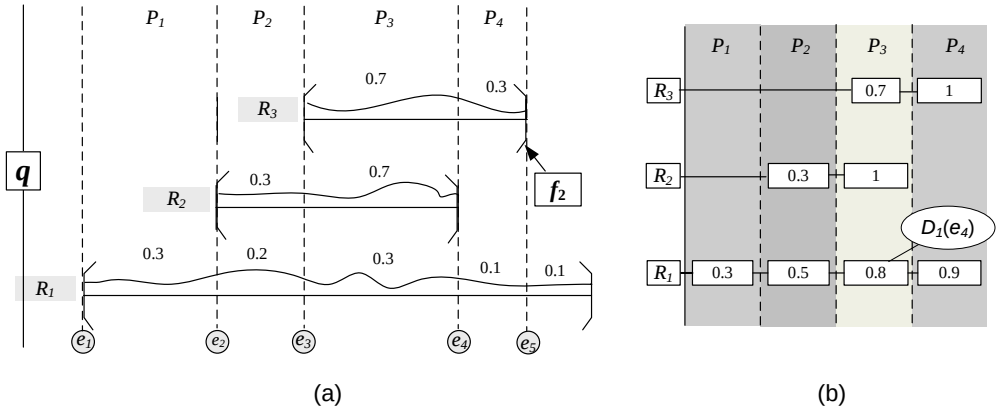


Figure 4.5: Illustrating the distance pdfs and partition probabilities (for $k = 2$).

The *subregions* in Section 3 can only handle distance pdfs of the candidate objects represented as arbitrary histograms, and the ranges of the subregions are depended on the end-points of the histograms. The partitions are not restricted to histogram pdfs, and can have variable sizes. Thus, the system has flexibility in deciding the number of partitions to be used. The more partitions are defined, the more accurate will be the lower and upper verification, with the need of more overhead for storing the partition information and prolonging the verification process. We found that the verification performs well when the boundaries of the objects' distance pdfs are used as end-points.

For each partition P_j of an object o_i, we evaluate the distance cdf of o_i on P_j's upper end-point (i.e., $D_i(e_{j+1})$). In general, the distance cdf of P_j's lower end-point is the same as that of P_{j-1}'s upper end point. For partition P_1, its lower end-point has a distance pdf of zero. We store the values of $D_i(e_{j+1})$ in a two-dimensional array of dimensions $|C| \times M$, where M is the number of partitions, so that they can be accessed in $O(1)$ times. The time for sorting and initializing this array is $O(|C| \log |C| + M|C|)$.

Verification Process. We now demonstrate how partitions can be used to efficiently derive lower and upper bounds of each k-subset's qualification probability (i.e., $p(S)$). Let $[p(S).l, p(S).u]$ be the lower and upper bounds of $p(S)$. Let X be the data structure that stores the partition information, and Q be the set of k-subsets generated from the PCS algorithm. Given these inputs, the verification algorithm (Algorithm 4 judges whether a k-subset S should be considered as a query answer, just like the *classifier* does (Figure 3.4). It produces an answer set A (i.e., k-subsets that satisfy the query) and a refinement set U (i.e., k-subsets that need to be further investigated). The UB (LB) is to find the the upper(lower) bound of S's qualification probability, i.e., $p(S).u$ ($p(S).l$).

input : Partition info. X, set Q of k-subsets
output: set A of answers, set U of k-subsets to be refined

1 $A \leftarrow \emptyset$; $U \leftarrow \emptyset$;
2 **for each** $S \in Q$ **do**
3 **if** $\mathsf{UB}(X, S) \geq T$ **then**
4 **if** $\mathsf{LB}\ (X, S) \geq T$ **then**
5 insert S into A;
6 **else**
7 insert S into U;
8 **return** A, U

Algorithm 4: Verification.

It is worth mention that in Step 3 of Algorithm 4, we put UB *before* LB. This is because in our experiments, a large number of k-subsets can be pruned by UB (in Step 3). By testing the k-subsets with UB first, we avoid applying the LB test to the k-subsets, which have to be pruned anyway. We will revisit this issue in Section 4.5.

We now explain the design of LB and UB, which returns $p(S).l$ and $p(S).u$.

Suppose $m(S)$ is the maximum distance between all objects in S and q, i.e., $\max(\{r_i | o_i \in S\})$. Let $y_j(m(S))$ be the probability that $m(S)$ is within the partition $P_j = [e_j, e_{j+1}]$. Let $p_j(S)$ be the qualification probability of $p(S)$, given that $m(S)$ lies in P_j. Let $[p_j(S).l, p_j(S).u]$ be the lower and upper bounds of $p_j(S)$. If there are M partitions, we have:

$$p(S).l = \sum_{j=1}^{M} p_j(S).l \cdot y_j(m(S)) \tag{4.2}$$

$$p(S).u = \sum_{j=1}^{M} p_j(S).u \cdot y_j(m(S)) \tag{4.3}$$

Moreover,

$$y_j(m(S)) = \prod_{o_i \in S} D_i(e_{j+1}) - \prod_{o_i \in S} D_i(e_j) \tag{4.4}$$

This is because the term $\prod_{o_i \in S} D_i(e_{j+1})$ is the probability that all objects in S have distance from q not larger than the end-point e_{j+1}. By subtracting $\prod_{o_i \in S} D_i(e_j)$ from it, we obtain the probability that at least one object is located inside $P_j = [e_j, e_{j+1}]$. This is also the chance that the maximum distance of all objects in S (i.e., $m(S)$) is within P_j, as shown in Equation 4.4. The following describes the formulas for $p_j(S).l$ and $p_j(S).u$.

Lemma 4.4 *Given that $m(S) \in P_j$, the lower and upper bounds of qualification probabilities of k-subset, S, are:*

$$p_j(S).l \geq \prod_{o_i \in C-S} (1 - D_i(e_{j+1})) \tag{4.5}$$

$$p_j(S).u \leq \prod_{o_i \in C-S} (1 - D_i(e_j)) \tag{4.6}$$

Proof : Equation 4.5: Since the maximum distance of all objects in S from q is less than e_{j+1}, S must be the answer if all other objects in $C-S$ have

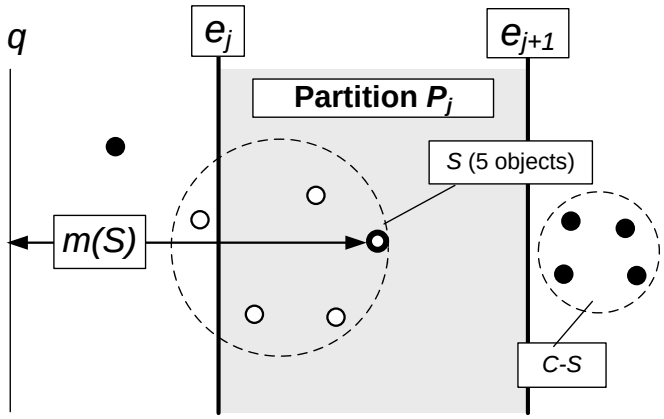


Figure 4.6: Correctness proofs for $p_j(S).l$ and $p_j(S).u$.

distances more than e_{j+1}. For example, Figure 4.6 shows that S, a 5-subset, has a maximum distance ($m(S)$) not more than e_{j+1}. If the remaining objects ($C - S$) are on the right of e_{j+1} (circled), then S must constitute a query answer. The probability that this event happens is $\prod_{o_i \in O-S}(1 - D_i(e_{j+1}))$ (the right side of Equation 4.5), which is also the lower bound of $p_j(S)$.

Equation 4.6: If any object in $C - S$ has distance from q shorter than e_j, then S could not be the set of k closest neighbors of q. In Figure 3.8, for example, since an object (colored black) is on the left of e_j, it is certainly closer to q then at least one object in S. So, S cannot be a query answer for $k = 5$. The event that all objects in $C - S$ have distance from q more than e_j is thus a precondition for S to be the query answer. The probability that this event happens, i.e., $\prod_{o_i \in O-S}(1 - D_i(e_j))$ (the right side of Equation 4.6), is therefore the upper bound of $p_j(S)$. □

With Equation 4.4 and Lemma 4.4, the lower and upper bounds of $p(S)$ (i.e., Equations 4.2 and 4.3) can be estimated. If the partition data structure presented earlier is used, retrieving $D_i(e_j)$ (given i and j) needs $O(1)$ times. Evaluating Equation 4.4 thus needs $O(k)$ times. Computation of

Equations 4.5 and 4.6 both requires $O(|C|)$ times. Thus the LB and UB functions have a complexity of $O(kM|C|)$. The total complexity of the verification algorithm is $O(kM|C||Q|))$.

Incremental Refinement. After verification, objects stored in the set U (Step 3 of Algorithm 4) require further processing, whose exact qualification probabilities need to be computed. This can be expensive, since numerical integration may be needed (see Equation 4.1). With the idea of *incremental refinement* (Section 3.3.5), we can speed up this process with the information obtained during verification. The main idea of incremental refinement is to treat the probability of an object as a sum of qualification probabilities inside partitions. By using the bound information of probabilities in each partition, the answer probabilities can be gradually computed.

Specifically, observe that the probability bounds of each k-subset S in each partition P_j (i.e., $[p_j(S).l, p_j(S).u]$) have been obtained during verification. For each P_j, once we get the value of $p_j(S)$ (by Equation 4.1), we can collapse $[p_j(S).l, p_j(S).u]$ into $p_j(S)$, update the probability bound of $p(S)$ (i.e., $[p(S).l, p(S).u]$), and test this new bound against the threshold T. This process is repeated for the next partition until we can decide whether S should be included in the answer. As shown in our experiments, "incremental refinement" is usually faster than computing probabilities directly, since performing numerical integration on a partition is faster than on $[0, f_k]$, which has a larger area of integration.

4.5 Experimental Results

We have performed extensive experiments on a real data set to examine the effectiveness of our solution. We first describe the experimental setup in Section 4.5.1. Then we present the results in Section 4.5.2.

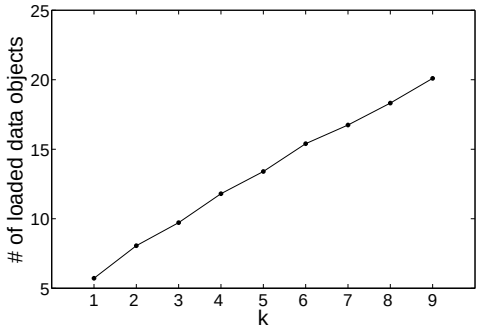


Figure 4.7: # of Loaded Data Objects.

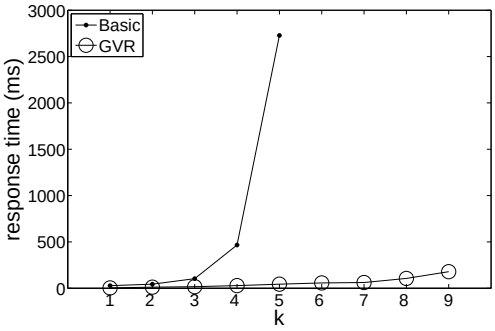


Figure 4.8: Basic vs. GVR.

4.5.1 Experimental Setup

We use the *Long Beach* dataset[2] which includes 53,144 rectangles, distributed in the two-dimension space of $10K \times 10K$ units. Each rectangle is treated as an uncertainty region, with a uniform pdf as the default. We also perform experiments on Gaussian pdf (represnted as a histogram). For each T-k-PNN query, the default values of probability threshold (T) and k are 0.1 and 6 respectively. The query point is randomly chosen from the 2D space. Each data point is an average of results for 50 runs. Under these settings, a T-k-PNN query produces two k-subsets as answers on average.

The experiments, written in Java, are executed on a PC with an Intel 2.66GHz CPU and 2GB of main memory. We have also implemented the k-bound filtering with the R-tree library in the Spatial Index Library [63].

4.5.2 Results

1. k-bound Filtering. In the first experiment, we examine the effectiveness of the k-bound filtering in pruning away unqualified objects. Fig. 4.7 illustrates the number of loaded data objects returned by k-bound filtering with an R-tree. As we can see, when k varies from one to nine, the size of

[2]Available at http://www.census.gov/geo/www/tiger/.

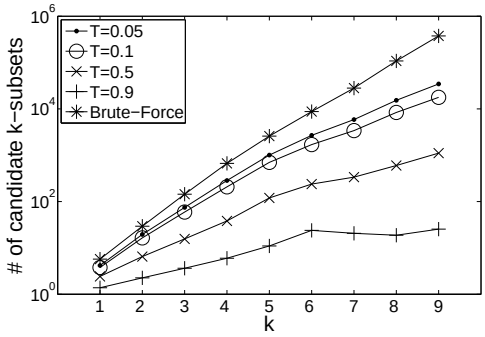


Figure 4.9: Generating k-subsets.

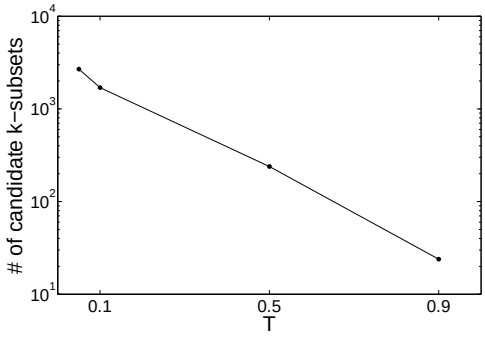


Figure 4.10: Effect of T on PCS ($k = 6$).

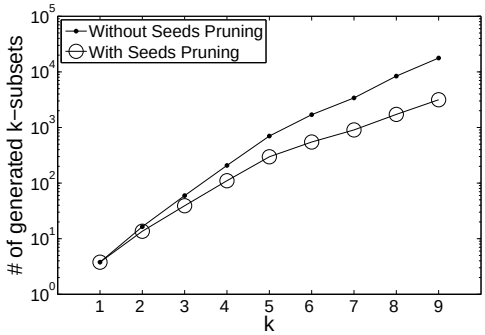


Figure 4.11: Seed Pruning (# k-Subsets).

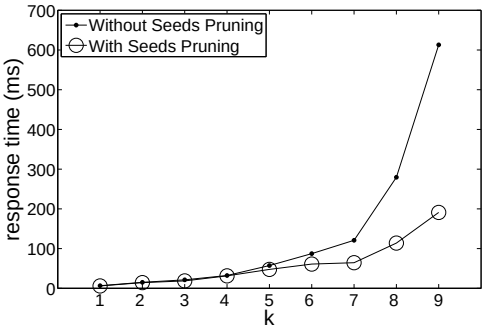


Figure 4.12: Seed Pruning (Response Time).

the candidate object set increases smoothly. This is because the size of the k-bound increases with k. Consequently, the k-bound has overlap with more objects, and so more candidates need to be investigated. Another observation is that the number of candidate objects is small. In fact, the average fraction of the total database size to be examined is less than 0.04%. Thus, the pruning power of k-bound filtering is quite impressive.

On the other hand, although only a small fraction of objects are returned by k-bound filtering, the number of k-subsets generated by the candidate object set can still be very large. At $k = 9$, for instance, 22 objects are left. Out of these objects, a total of C_9^{22} (around 375K) k-subsets need to be examined. This renders a huge computational effort. To alleviate

this problem, we need k-subset **G**eneration (with PCS), **V**erification, and **R**efinement techniques. Let us call these techniques collectively as the **GVR** method, and examine its effectiveness.

2. Performance of GVR. Here we compare the performance of GVR with that of *Basic* evaluation (described in Section 4.2.2). We assume that k-bound filtering has been applied first for both methods. As shown in Figure 4.8, the time required by *Basic* rises sharply with k, since the increase in k makes Equation 4.1 more expensive to compute. On the other hand, the query response time of GVR is an order of magnitude less than *Basic*. For example, when $k = 5$, GVR spends only 1.6% of the time required by *Basic*. We can thus see that GVR is important for improving the query performance. Next, let us investigate individual methods of GVR.

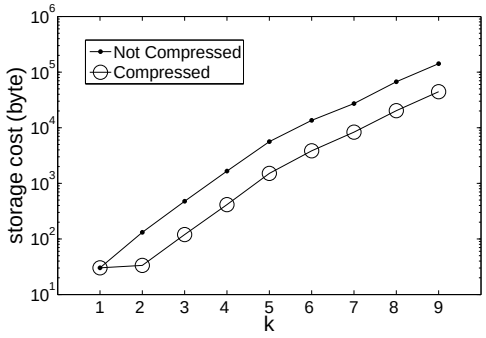


Figure 4.13: Efficient Storage of k-subsets.

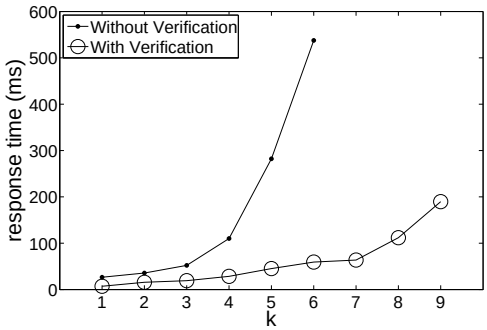


Figure 4.14: Effect of Verification.

3. k-subset Generation. In this experiment, we study the performance of the PCS algorithm in generating k-subsets. Figure 4.9 shows the number of k-subsets produced by different techniques, in log scale. Compared with the "brute-force" method (i.e., enumerating all possible k-subsets from the candidate objects), PCS consistently generates less k-subsets under a wide range of T values. The savings are significant; at $k = 9$, for example, the improvement of PCS over the brute-force method is 90% (for $T = 0.05$) and

99% (for $T = 0.5$). Figure 4.10 shows that when T increases, the number of candidate k-subsets decreases sharply. Thus, the effectiveness of PCS improves with a higher value of T. It also shows that PCS can exploit the probability threshold to provide better performance.

To further enhance PCS, we have proposed **seed pruning** (in Section 4.3.3). As shown in Figure 4.11, this technique reduces the number of k-subsets produced over a wide range of k. For example, at $k = 9$, the improvement is about 80%. Figure 4.12 shows the corresponding effect on query response time, which addresses a saving of 69% at $k = 9$. Thus, seed pruning improves the performance of PCS significantly.

In view of the potentially large number of k-subsets generated during and after the execution of the PCS algorithm, we have designed an effective compression as discussed in Section 4.3.3. Figure 4.13 compares the storage cost with and without using this compression method (in log scale). We observe that under a wide range of values of k, after compression, we only need one-third of the storage that is used to store the raw k-subsets. Therefore, our compression method can greatly reduce the amount of storage required to record k-subsets for further processing.

4. Verification and Refinement. Next, we investigate the advantage of verification and refinement over direct computation of qualification probabilities. Figure 4.14 shows that the use of this technique yields significant improvement over different values of k. For example, at $k = 6$, verification and refinement reduces the query response time by about 90%.

We further examine the effectiveness of lower- and upper-bound verification. The lower (upper) bound verification method attempts to determine whether a k-subset should be accepted (rejected). Figure 4.15 shows that the number of k-subsets classified by UB is much larger than that classified by

LB. The reason is that in the dataset we have tested, many k-subsets have small qualification probabilities. Thus, they are more likely to be rejected through upper-bound verification. Due to this reason, we have also arranged the UB subroutine to be executed before LB in the verification algorithm, as shown in Algorithm 4.

5. Time Analysis. To get a clearer picture about the performance of each part of our solution, we measure the time costs of k-bound filtering (shown as "Index" in Figures 4.16 and 4.17), k-subset generation (with PCS), verification, as well as refinement. Figures 4.16 show the result under different values of k. In general, most of the time is spent on refinement. This is hardly surprising, because refinement, which performs numerical integration on Equation 4.1, is an expensive process. However, this is already better than doing numerical integration alone (c.f. Figure 4.14). The price to pay for this time drop is to verify the k-subsets before their probabilities are actually evaluated. Although the time spent on verification also increases with k, the time spent is still less than pure numerical integration (c.f. Figure 4.14). We also notice that that k-bound filtering and PCS require the least amount of the time. These two steps add little overhead to the overall query performance. However, their gain, as reflected by Figures 4.7 and 4.9, is significant.

Figure. 4.17 shows the time breakdown of the components for different values of T. Again, the time costs required by k-bound filtering and PCS are the least. For all the methods, their performance improves with an increase of T. This shows that our methods can effectively exploit the query probability threshold.

6. Gaussian Distribution. In the final experiment, we use a Gaussian distribution as the uncertainty pdf for the dataset. For each object, the

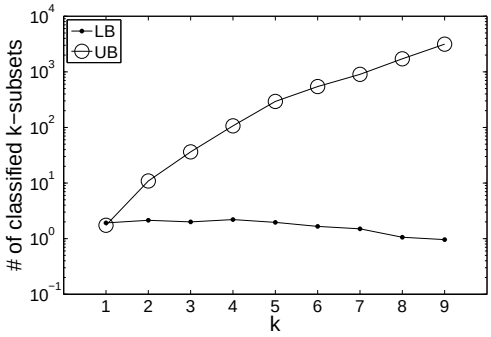


Figure 4.15: LB vs. UB.

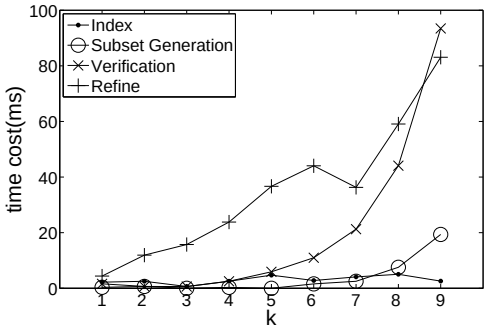


Figure 4.16: Time Analysis (with T=0.1)

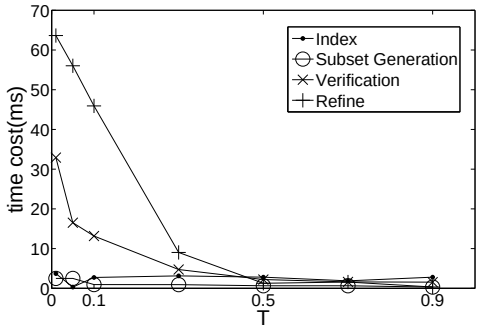


Figure 4.17: Time Analysis (with k=6).

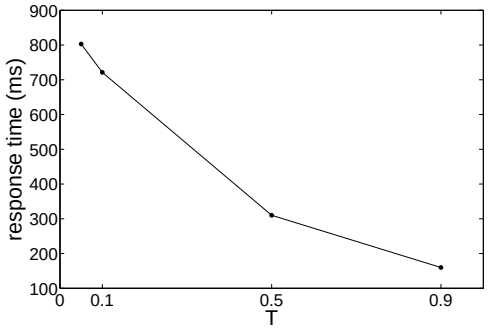


Figure 4.18: Various T on Gaussian Distribution

uncertainty pdf has a mean equal to the center of the uncertainty region, and a variance set to be the square of one-sixth of the edge length, in both x and y dimensions. Each uncertainty pdf is represented by $10 \times 10 = 100$ histogram bars, and the probability of each bar is the integration of the pdf over the area covered. Figure 4.18 illustrates the result of the GVR method for various values of T. We observe that GVR shows a similar trend as that of the uniform pdf (c.f. Figure 4.8). More time is spent on Gaussian pdf, because more histograms are used to model the pdf, which subsequently increases the time for verification and refinement. We have also performed other experiments for Gaussian pdf, and they also reflect similar trends. We thus omit them in this thesis. From these experiments, we can see that our

solution is robust with respect to different types of uncertainty pdfs.

4.6 Chapter Summary

Due to the popular usage of uncertain data in many real applications, uncertainty management has become an important topic in the database community. We studied a useful query, namely, the probability threshold k-NN Query (T-k-PNN) for uncertain databases. Different from the exact database, evaluating T-k-PNN requires probability information, and performs expensive numerical integration. Thus, we proposed various pruning techniques with consideration of both distance and probability constraints. As shown by our experimental results, with the k-bound filtering technique, a lot of unqualified objects can be pruned. The number of k-subsets can be significantly reduced by the PCS algorithm. We further demonstrated the efficient computation of lower and upper bounds of probabilities with the aid of partition information. We will study how these techniques can be extended to support other queries, e.g., reverse-neighbor and skyline queries.

5 Imprecise-Location Dependent Queries

In this chapter, we present our works about efficient evaluation of imprecise location-dependent queries. In LBS, the "location-dependent range query" is quite common, where the user's location is at the center of the query range. Very often, the query issuer's location is imprecise due to measurement error, sampling error, or message delay. He/She may also want to protect his/her privacy by providing a less precise location. In this chapter, we study the efficiency of queries that return probabilistic guarantees for location data with uncertainty. We develop three methods to improve the computational and I/O performance. Experimental simulation over a realistic dataset reveals that our approaches improve the query performance significantly.

5.1 Introduction

In recent years, positioning technologies like GPS, GSM, RF-ID and the Wireless LAN have undergone rapid development [86]. These technologies allow locations of users to be determined accurately, and enable a new class of applications known as *location-based services* (LBS). An important LBS is the E-911 system mandated by the U.S. (correspondingly E-112 in Europe), which requires cell phone companies to provide an accurate location (i.e., within a few hundred feet) of a cell phone user who calls for emergency help [86]. Other LBS applications include downloading driving directions to a gas station, receiving an alarm when a military adversary has crossed the border, retrieving the current locations of family members, and displaying the user's location on the map. All these applications require extensive use of location data [57].

An important issue concerning the LBS is the *uncertainty* of location

data. In particular, location data obtained from physical devices are inherently imprecise due to measurement error, sampling error and network latency [76, 70, 24]. Some recent works (e.g., [8, 27, 41]) have suggested that location privacy can be protected by injecting a controlled degree of spatial uncertainty into location data, so that the actual location is hidden. In many situations, therefore, it is often impossible for the query processing server to get an accurate location value. It is thus reasonable to use a location uncertainty model to describe the imprecision of the data values, and evaluate the location uncertainty in order to provide probabilistic guarantees over the validity of the query answers. A common model for characterizing location uncertainty of an object is a closed region together with a probability distribution of the object in the region [76, 70], as we have illustrated in Section 1.1. Some previous work, such as [76, 70, 24], used this model to compute probabilities of each location object for satisfying a query, including the range and nearest-neighbor queries. The probability values provide confidence guarantees about the query answer, and allow quality of service metrics to be defined [23, 27].

In this chapter, we study the effect of uncertainty on *location-dependent queries*, which takes into account the location of the user who issues the query (called "query issuer") in order to decide the query result [57, 40, 47]. For example, the user may want to know the available cabs within two miles of his/her current location. In addition to the uncertainty of the data being queried, the imprecision of the query issuer's location further affects the validity of the query answer. Our goal is to attempt to quantify the query answer validity by efficiently computing the qualification probability of an object for satisfying this type of query. To our best knowledge, this has not been studied before.

Specifically, we study the imprecise version of the *location-dependent range query*. Based on the location information of the query issuer, together with a range region centered at the query issuer's location, the query returns the identities of all objects that fall within the region. We propose efficient algorithms to compute qualification probabilities. Furthermore, we introduce a set of novel filtering methods to determine the area that can contain candidate objects matching the query. We classify the query according to whether the location data being accessed is (1) *precise* (e.g., locations of gas stations, schools etc.), or (2) *uncertain* (e.g., locations of moving objects). Based on this classification, we develop three fundamental concepts to enhance the evaluation of the qualification probabilities:

- **Query expansion:** Incorporate the uncertainty information of the query issuer's location into the query range by using computational geometry techniques, so that query evaluation can take advantage of traditional query processing methods.

- **Query-Data duality:** By interchanging the role of the query issuer and that of the location object, simplify the qualification probability formula and thus save the query evaluation cost.

- **Use of the probability threshold constraint:** By exploiting the assumption that users are only concerned about answers with high qualification probabilities, special data constructs are pre-computed for uncertain objects in order to facilitate pruning.

The aforementioned methods can be executed efficiently. They can also deal with *any* type of probability distribution about the object's location. We perform detailed experimental evaluations to examine our approaches.

The rest of this chapter is organized as follows. Section 5.2 gives a formal definition of the problems. In Section 5.3 we discuss how to improve the performance of imprecise location-dependent queries. Section 5.4 then presents algorithms that exploit the probability threshold constraint. In Section 5.5 we report our experimental results. Section 5.6 concludes the chapter.

5.2 Problem Definition

In this section we formally define the queries studied in this chapter. We also investigate preliminary solutions for these queries.

5.2.1 Imprecise Location-Dependent Range Queries

In this chapter, we focus on a type of snapshot, location-dependent query – the location-dependent range query. Without loss of generality, we denote o_0 as the identity of the query issuer, and $o_1, \ldots, o_n$ as the identities of the objects being queried. We also assume the uncertainty region is an axis-parallel rectangle. For objects with no uncertainty (called *point objects*), we denote them $s_1, \ldots, s_m$. In particular, object s_i's location is exactly a point (x_i, y_i) (e.g., a non-moving user or a building).

Therefore, given an axis-parallel rectangle $R(x, y)$ with center (x, y), half-width w and half-height h, two types of queries can be defined:

Definition 5.1 *An Imprecise Location-Dependent Range Query over Point Objects (IPQ) returns a set of tuples* $\{(s_i, p_i)|i \in [1, m]\}$ *where* $p_i > 0$ *is the non-zero probability that* s_i*'s location,* (x_i, y_i)*, is inside* $R(x, y)$*, with* $(x, y) \in u_0$*.*

Definition 5.2 *An Imprecise Location-Dependent Range Query over Uncertain Objects (IUQ) returns a set of tuples* $\{(o_i, p_i)|i \in [1, n]\}$ *where* $p_i > 0$ *is*

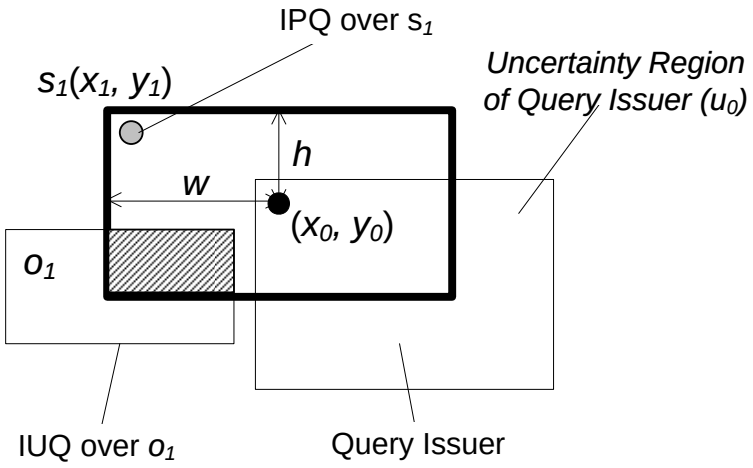


Figure 5.1: Evaluating IPQ and IUQ.

the non-zero probability that o_i is located within $R(x, y)$, with $(x, y) \in u_0$.

Figure 5.1 illustrates the IUQ and IPQ. For convenience, we may use R to represent $R(x, y)$.

Sometimes users are more concerned about answers with sufficiently high probability values. In fact, it is useful to define a "probability threshold constraint", which restricts queries to return answers with probability values higher than a certain pre-defined value. We call this parameter the *probability threshold* (T in short), which is a real value between 0 and 1. The following queries can then be defined:

Definition 5.3 *A Constrained Imprecise Range Query over Point Objects (C-IPQ) returns a set of tuples $\{s_i | i \in [1, m]\}$ such that $p_i \geq T$, where p_i is the qualification probability of s_i for satisfying the corresponding IPQ.*

Definition 5.4 *A Constrained Imprecise Range Query over Uncertain Objects (C-IUQ) returns a set of tuples $\{o_i | i \in [1, n]\}$ such that $p_i \geq T$, where p_i is the qualification probability of o_i for satisfying the corresponding IUQ.*

Later we will show how to use the probability threshold to improve query performance. Table 5.1 describes the notations used for defining the location-dependent queries.

Symbol	Meaning
o_0	Query issuer (an uncertain object)
$f_i(x, y)$	Uncertainty pdf of o_i
s_i	A point object
(x_i, y_i)	Position of s_i
$R(x, y), w, h$	Range query with half-width w and half-height h, centered at (x, y)

Table 5.1: Symbols for IPQ and IUQ.

5.2.2 Basic Evaluation Methods

Let us now present a basic solution for evaluating IPQ and IUQ. They form the foundation for further discussions.

For **IPQ**, the qualification probability of s_i can be obtained by conceptually examining every point $(x_0, y_0) \in u_0$, and then checking whether the location of s_i is within $R(x_0, y_0)$. Figure 5.1, for example, illustrates that s_1 satisfies $R(x_0, y_0)$. The final result can be obtained by integrating the uncertainty pdf of all the points (x, y) in u_0 at which s_i satisfies $R(x, y)$. Formally, we define a boolean function, $b_i(x, y)$ (for $i = 1, \ldots, n$), as follows:

$$b_i(x, y) = \begin{cases} 1 & \text{if } s_i \text{ is inside } R(x, y) \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The qualification probability of s_i for satisfying the IPQ is then given by

$$p_i = \int_{u_0} b_i(x, y) f_0(x, y) dx dy \tag{5.2}$$

For **IUQ**, we examine the probability that an uncertain object o_i satisfies

the query at each point in u_0. This is given by integrating the uncertainty pdf of o_i in the overlapping area of u_i and $R(x, y)$, i.e.,

$$p_i(x, y) = \int_{u_i \cap R(x,y)} f_i(x, y) dx dy \tag{5.3}$$

Figure 5.1 shows that the probability of o_1 satisfying $R(x_0, y_0)$, given that o_0 is at point (x_0, y_0), is the integral of $f_1(x, y)$ over the shaded region. Considering the uncertainty pdf of o_0, the following gives the general formula for computing p_i.

$$p_i = \int_{u_0} p_i(x, y) f_0(x, y) dx dy \tag{5.4}$$

In practice, Equations 5.2 and 5.4 are costly to implement. Both equations may necessitate the use of numerical integration. For example, in order to obtain p_i in Equation 5.2, u_0 is first represented by a set of sampling points; for each sampling point, Equation 5.2 is evaluated. A large number of sampling points will be needed to produce an accurate answer. This is required even if the uncertainty pdf is as simple as a uniform distribution. Let us investigate how this situation can be improved.

5.3 Efficient Evaluation of Imprecise Queries

We just see that straightforward computation of probabilities for IPQ and IUQ can lead to expensive integral operations. This section illustrates how such operations can be avoided, by (1)expanding the range query, and (2) exploiting the duality between the locations of the query issuer and data being queried. We also examine indexing techniques for these queries.

5.3.1 Query Expansion

The first technique performs an inexpensive filtering over objects that have no chance of being included in the query answer. The main idea is to expand the query range R with the query issuer's position information. Any object that does not touch this expanded query range (called the *Minkowski Sum* [9]) can be pruned.

Lemma 5.1 *The qualification probability of a point object (an uncertain object) is non-zero if and only if its location (uncertainty region) lies within (overlaps) the Minkowski Sum of* $R(0,0)$ *and* u_0.

To explain the above lemma, let us consider the IUQ (similar arguments can be applied to IPQ). First, notice that the Minkowski Sum is defined as follows:

$$A \oplus B = \{x + y | x \in A, y \in B\}$$

where A and B are two given polygons, and x and y are points [9]. Conceptually, the Minkowski Sum is the union of all translations of A by a point y located in B. We can view the Minkowski Sum of the query range $R(0,0)$ and the uncertainty region u_0, that is, $R(0,0) \oplus u_0$, as the union of all range queries, by considering all the possible positions of o_0 who resides somewhere in u_0. Notice that here we should use $R(0,0)$ instead of a rectangle centered at the query issuer's location, e.g. $R(x,y)$ with $(x,y) \in u_0$, because the region $R(x,y) \oplus u_0$ may be shifted away from u_0. If the uncertainty region of any object being queried does not overlap $R(0,0) \oplus u_0$, we can be assured that this object does not have any chance of satisfying any range query issued at any position in u_0. Thus we can use $R(0,0) \oplus u_0$ as a query range to obtain objects that have non-zero qualification probability of satisfying

the IUQ (i.e., their uncertainty regions overlap with the query range). For illustration convenience, we will use R to denote $R(0, 0)$ in the following text.

Figure 5.2 illustrates the Minkowski Sum of R and u_0, which can simply be obtained by extending u_0 by w on the left and right, and by h on the top and bottom.[3] Hence, the Minkowski Sum can be derived in a linear time. As shown in the same figure, the expanded query range allows objects with zero qualification probability (i.e., objects O_1) to be pruned immediately.

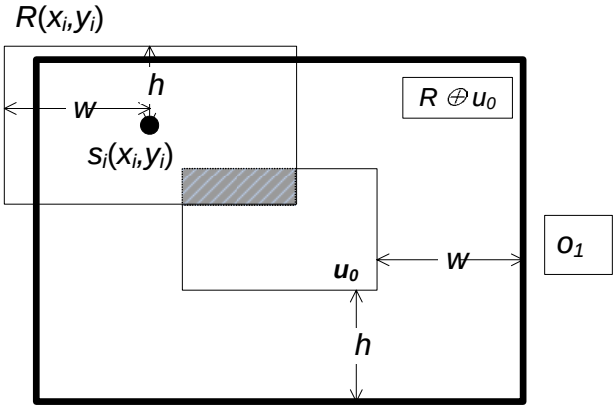


Figure 5.2: Illustrating the evaluation of IPQ. The thick line is the expanded query using the Minkowski Sum.

5.3.2 Query-Data Duality

The second method exploits the fact the role of the query issuer and the data being queried can be changed. Specifically, the following observation describes this "query-data duality" property:

Lemma 5.2 Query-Data Duality *Given two point objects s_i and s_q (with locations (x_i, y_i) and (x_q, y_q) respectively), s_i satisfies $R(x_q, y_q)$ if and only if s_q satisfies $R(x_i, y_i)$.*

[3] If u_0 and R are m-sided and n-sided polygons, the Minkowski Sum is a convex polygon with at most $m + e$ edges, which requires $O(m + e)$ time to compute [9].

Proof : Construct a rectangle with vertices M, s_i, N, and s_q, as shown in Figure 5.3. If s_i satisfies $R(x_q, y_q)$, then $|s_iM| \leq w$ and $|s_iN| \leq h$. This implies $|s_qN| \leq w$ and $|s_qM| \leq h$. Hence s_q must satisfy the query $R(x_i, y_i)$. Conversely, if s_q satisfies $R(x_i, y_i)$, we can construct the same rectangle and prove that $|s_iM| \leq w$ and $|s_iN| \leq h$. Hence s_i must also satisfy $R(x_q, y_q)$. □

In other words, if a point object s_i satisfies the range query issued by s_q, then s_q must also satisfy the range query issued by s_i. This leads us to the following result.

Lemma 5.3 *The qualification probability p_i of a point object s_i for satisfying an IPQ can be computed by*

$$\int_{R(x_i,y_i)\cap u_0} f_0(x, y)dxdy \tag{5.5}$$

Proof : Consider the overlapping region of u_0 and the range query $R(x_i, y_i)$ issued by s_i, as shown in the shaded area in Figure 5.2. Obviously, any point $(x_e, y_e) \in u_0 \cap R(x_i, y_i)$ satisfies the query $R(x_i, y_i)$. Using Lemma 5.2, s_i also satisfies the range query $R(x_e, y_e)$. Moreover, s_i does not satisfy any range queries centered at points outside the overlapping region. Hence only the queries issued at points (x_e, y_e) can have s_i in their answer, and the qualification probability of s_i is simply the integration of the uncertainty pdf of O_0 in the overlapping region, i.e., $\int_{R(x_i,y_i)\cap u_0} f_0(x, y)dxdy$. □

Compared with the original formula for IPQ (i.e., Equation 5.2), we can see that Equation 5.5 is simpler to evaluate. This is because now we do not need to form a query at each point in u_0 and test whether s_i satisfies the query at that point, as in Equation 5.2. More importantly, if the uncertainty pdf of the query issuer is a simple function, the integration operation of

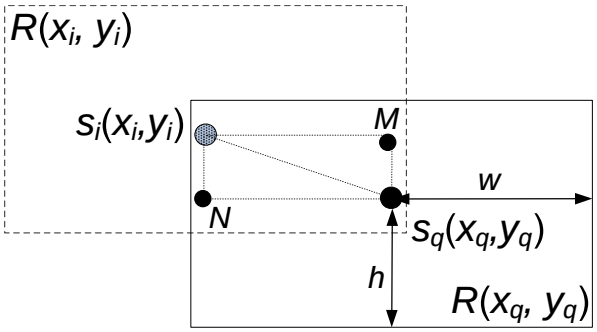


Figure 5.3: The Duality of Query and Data.

Equation 5.2 can be eliminated. As an important example, if $f_0(x, y)$ is a uniform distribution, p_i is simply the fraction of u_0 that overlaps $R(x_i, y_i)$, i.e.,

$$p_i = \frac{\text{Area}(R(x_i, y_i) \cap u_0)}{\text{Area}(u_0)} \tag{5.6}$$

When the position of s_i is at different positions relative to u_0, the exact formula for calculating the overlapping region $u_0 \cap R(x_i, y_i)$ (and hence Equation 5.6) can also vary. In particular, the 2D space can be partitioned into nine regions, and depending on which regions that s_i is located, a different algebraic expression is needed. The details will be illustrated soon in Section 5.3.3.

Lemma 5.3 can also be used to compute the qualification probability of an uncertain object o_i for satisfying the IUQ. We can conceptually treat every point $(x, y) \in u_i$ as a point object, and compute the qualification probability of each individual point (x, y) for satisfying the IPQ (termed $Q(x, y)$), with Lemma 5.3. The qualification probability of o_i is then simply equal to the integral of all these $Q(x, y)$ values, weighted by the uncertainty pdf of o_i at (x, y), i.e.,

$$p_i = \int_{u_i} f_i(x, y) \cdot Q(x, y) dx dy \tag{5.7}$$

Hence, Equation 5.7 provides an alternative to Equation 5.4 for computing IUQ. Although it is not immediately clear which method is better, we note that the performance of Equation 5.7 can be further improved when combined with our results about query expansion.

Lemma 5.4 *The qualification probability p_i of an uncertain object o_i for satisfying an IUQ can be computed by*

$$p_i = \int_{u_i \cap (R \oplus u_0)} f_i(x, y) \cdot Q(x, y) dx dy \tag{5.8}$$

The only difference between this equation and Equation 5.7 is that u_i is replaced by $u_i \cap (R \oplus u_0)$ – which potentially produces a smaller integration region and better computational performance. How is this possible? Observe that for any point $(x_t, y_t) \in u_i - (R \oplus u_0)$, $Q(x_t, y_t)$ must be zero, according to Lemma 5.1. Hence it is fine to focus on the portion of u_i that overlaps the expanded query region. Figure 5.2 illustrates an example, in which the shaded part of u_2 is the region of integration for Equation 5.8.

5.3.3 Calculation of Qualification Probabilities

We have introduced Equation 5.6 to compute the overlapped ratio $Area(u_0 \cap R(x_i, y_i))/Area(u_0)$, which is the basic part of calculating the qualification probability . The exact formula of calculating the overlapped region $u_0 \cap R(x_i, y_i)$ (and hence Equation 5.6) will vary at different positions relative to u_0. In particular, the region of Minkowski Sum can be partitioned into nine regions, and depending on which region that s_i is located, a different algebraic expression is needed. Here we will discuss how to get the nine regions and the formulas of Equation 5.6 in each region.

Figure 5.4 depicts the shapes of these nine regions. The small rectangle is the

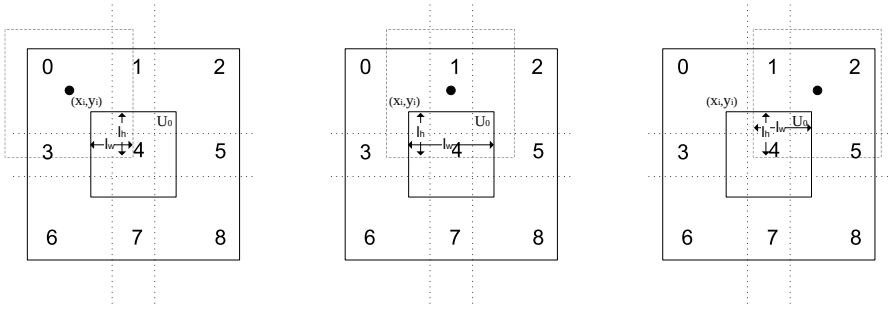


Figure 5.4: Region 0 Figure 5.5: Region 1 Figure 5.6: Region 2

cloaked location of the query issuer and the big rectangle is the Minkowski Sum. A precise object (the black point) with its transformed range query $R(x, y)$ (the dashed square) is located in region 0. The Minkowiski Sum is divided (by the dotted line) into 9 regions (labeled by the numeric values 0 to 8). l_w is the width of the overlapped region ($u_0 \cap R(x_i, y_i)$) and l_h is the height of this overlapped region. Thus Equation 5.6 is exactly equal to $\frac{l_w * l_h}{Area(u_0)}$.

Suppose (x_0,y_0) is at the center of u_0, and the width (height) of u_0 is $2w_0$ ($2h_0$).When the position of the precise object (the black point) varies, the formulas of l_w and/or l_h will change. For example, in region 0, $l_w = x_i + w - (x_0 - w_0)$ and $l_h = (y_0 + h_0) - (y_i - h)$. While in region 1, l_w is $2w_0$ (see Figure 5.5), and the boundary between region 0 and region 1 given by the vertical line is $x = x_0 + w_0 - w$. In region 2, l_w will change again to $x_0 + w_0 - (x_i - w)$ (see Figure 5.6). The boundary between region 1 and region 2 is $x = x_0 - w_0 + w$.

Thus depends on the x-coordinate of the point, l_w will have three different formulas, $x_i + w - (x_0 - w_0)$, $2w_0$ and $x_0 + w_0 - (x_i - w)$. Similarly, l_h will also

Region#	Boundaries (L, R, T, B)	$Q(x,y)$
0	$x = x^l - w$, $x = x^u - w$, $y = y^l + h$, $y = y^u + h$	$\frac{-xy+(y^u+h)x+(x^l-w)y+(w-x^l)(y^u+h)}{Area(u_0)}$
1	$x = x^u - w$, $x = x^l + w$, $y = y^l + h$, $y = y^u + h$	$\frac{-w_0y+w_0(y^u+h)}{Area(u_0)}$
2	$x = x^l + w$, $x = x^u + w$, $y = y^l + h$, $y = y^u + h$	$\frac{xy-(y^u+h)x+(x^u+w)y+(w+x^u)(y^u+h)}{Area(u_0)}$
3	$x = x^l - w$, $x = x^u - w$, $y = y^u - h$, $y = y^l + h$	$\frac{h_0x+h_0(w-x^l)}{Area(u_0)}$
4	$x = x^u - w$, $x = x^l + w$, $y = y^u - h$, $y = y^l + h$	$\frac{w_0h_0}{Area(u_0)}$
5	$x = x^l + w$, $x = x^u + w$, $y = y^u - h$, $y = y^l + h$	$\frac{-h_0x-h_0(x^u+w)}{Area(u_0)}$
6	$x = x^l - w$, $x = x^u - w$, $y = y^l - h$, $y = y^u - h$	$\frac{xy+(h-y^l)x+(w-x^l)y+(w-x^l)(h-y^l)}{Area(u_0)}$
7	$x = x^u - w$, $x = x^l + w$, $y = y^l - h$, $y = y^u - h$	$\frac{w_0y+w_0(h-y^l)}{Area(u_0)}$
8	$x = x^l + w$, $x = x^u + w$, $y = y^l - h$, $y = y^u - h$	$\frac{-xy-(h-y^l)x+(x^u+w)y+(x^u+w)(h-y^l)}{Area(u_0)}$

Table 5.2: Regions and Corresponding $Q(x,y)$

have three different formulas, $y_i + h - (y_0 - h_0), 2y_0$, and $y_0 + h_0 - (y_i - h)$. So finally we get 3*3=9 regions.

In Table 5.2, we list the boundaries of the regions and corresponding formulas of p_i (Equation 5.6). For the ease of presentation, we use $x^l = x_0 - w_0, x^u = x_0 + w_0$ and $y^l = y_0 - h_0, y^u = y_0 + h_0$. Notice that the formulas we discussed above are generated based on the condition $w >= w_0 \wedge h >= h_0$. And l_w will be $2w$ in region 1 if $w < w_0$. That is, based on the relationships of w and w_0, and h and h_0, there are totally 4 cases, $w >= w_0 \wedge h >= h_0$,

$w >= w_0 \wedge h < h_0$, $w < w_0 \wedge h >= h_0$ and $w < w_0 \wedge h < h_0$. However, we don't have to calculate the formulas for all the 4 cases. For example, in region 2, if $R(x, y)$ is smaller than u_0, then we can actually switch their roles – consider $R(x, y)$ as u_0 and u_0 as $R(x, y)$, and use the formula in region 8 instead. Thus here we just list the boundaries and formulas (of p_i) for one case($w >= w_0 \wedge h >= h_0$). In other three cases, there is only little difference.

5.3.4 An Efficient I/O Solution

To improve the I/O performance of query processing, spatial data indexes such as the R-tree [45] and the grid file [65] are often used. In order to utilize these indexes for processing imprecise queries, we first construct an expanded query range online (i.e., find the Minkowski Sum of R and u_0). This expanded query range is then used to query the spatial index. All the point objects or uncertain objects that lie completely outside the expanded range can be pruned. The qualification probabilities of the remaining objects can then be computed by using the lemmas described in Section 5.3.2.

5.4 Constrained Imprecise Queries

So far we have addressed imprecise queries that produce answers with non-zero probabilities. In this section we study how query performance can be improved by only allowing objects with qualification probabilities higher than a pre-defined threshold value to be returned. These queries, called C-IPQ and C-IUQ, are the constrained version of IPQ and IUQ, as defined in Section 5.2. Our main idea is to use pre-computed boxes for uncertain objects, based on their uncertainty pdf, in order to achieve better pruning effects. In particular, we employ the concept of the p-bound [26, 79], as described next.

5.4.1 Pruning Point Objects for C-IPQ

A p-bound of an uncertain object o_i is a function of p, where $p \in [0, 0.5]$. It is composed of four line segments, namely $l_i(p), r_i(p), t_i(p), b_i(p)$ in 2D space, as illustrated by the hatched region in Figure 5.7. The requirement of $l_i(p)$ is that the probability of the location of o_i on the left of $l_i(p)$ has to be exactly equal to p (as shown by the shaded area). Similarly, the probability of o_i on the right of the line $r_i(p)$ is exactly equal to p. The remaining line segments ($t_i(p)$ and $b_i(p)$) are defined analogously. Notice that based on this definition, the boundary of u_i can be represented by $l_i(0), r_i(0), t_i(0)$ and, $b_i(0)$. We will show that if p-bounds have been pre-computed and stored for any value of p, better pruning power can be achieved.

In practice, it is not possible to pre-compute a p-bound for each value of p. Instead, a "U-catalog" is created for each object, which is a table of a small fixed number of tuples $\{v, v\text{-bound}\}$, where $v \in [0, 1]$ [79]. The tuples in the U-catalog can then used for query pruning. For ease of discussions, we assume that p-bound is created for each object, for any value of p. However, we will revisit the issue of U-catalog whenever appropriate.

Next, we define the *p-expanded-query as follows*:

Definition 5.5 *A p-expanded-query is a rectangular region such that any point object lying outside it has a qualification probability of less than p for satisfying the IPQ.*

Figure 5.8 illustrates a p-expanded-query, where by definition s_j has a probability of less than p of satisfying the IPQ. Notice that the Minkowski Sum of R and u_0 is equivalent to a 0-expanded-query, outside which no object has a qualification probability of more than zero. Also for any $p_0 > 0$, the p_0-expanded-query is always enclosed by the 0-expanded-query. In general,

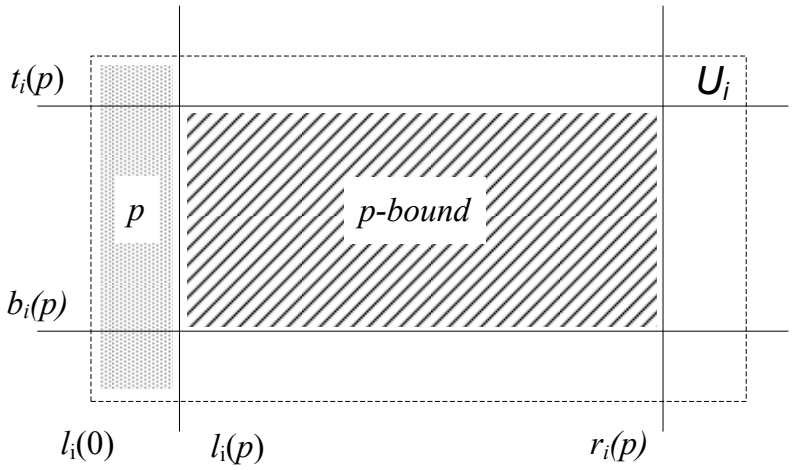


Figure 5.7: Illustrating the p-bound of o_i.

$p_j \geq p_k$ if and only if the p_j-expanded-query is enclosed by the p_k-expanded-query.

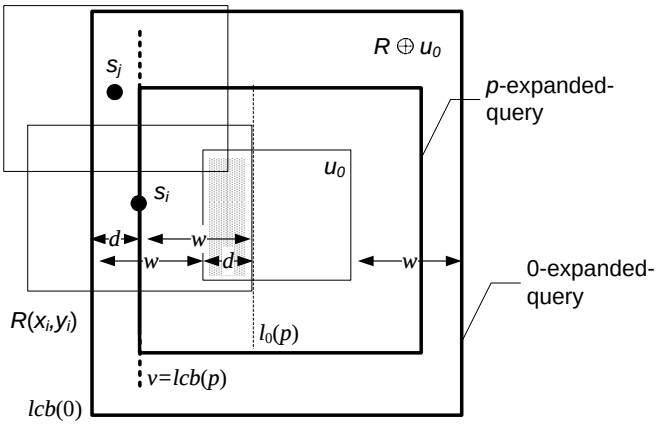


Figure 5.8: Pruning s_i for C-IPQ.

Let us use $lcb(p)$ to denote the left side of a p-expanded-query. The following lemma states an important property of $lcb(p)$.

Lemma 5.5 *$lcb(p)$ is d units from the right of $lcb(0)$, where d is the distance between the two lines $l_0(0)$ and $l_0(p)$ of u_0.*

Proof : Consider a point s_i, which is w units on the left of $l_0(p)$ (Figure 5.8). If an IPQ is issued by O_0, the qualification probability p_i of s_i must be less than or equal to p. This is because the integration of $f_0(x, y)$ over the (shaded) common region of $R(x_i, y_i)$ and u_0 cannot be larger than p, and according to Lemma 5.3 this is exactly equal to p_i. Next, consider the vertical line v intersecting s_i. We claim that line v is $lcb(p)$. This is because for any object s_j lying on v or on the left of v, either (1) $R_i(x, y)$ does not touch u_0 at all, or (2) $R_j(x, y) \cap u_0$ is a portion of the shaded region. Using Lemma 5.3,

$$\begin{aligned} p_j &= \int_{R_j(x,y)\cap u_0} f_0(x, y)dxdy && (5.9) \\ &\leq \int_{\text{shaded region}} f_0(x, y)dxdy && (5.10) \\ &= p && (5.11) \end{aligned}$$

Therefore, any point object on the left of the line v must have a qualification probability less than or equal to v. Hence $v = lcb(p)$. Moreover, as shown in Figure 5.8, $lcb(p)$ is d units from $lcb(0)$. Hence the lemma holds. □

By using Lemma 5.5, we can construct a p-expanded-query easily. As shown in Figure 5.8, $lcb(p)$ is simply the vertical line with a distance of w units from $l_0(p)$. The other sides of the p-expanded-query can be obtained analogously.

The p-expanded-query can be used to prune point objects for C-IPQ. Specifically, we first construct a T-expanded query (where T is the probability threshold of C-IPQ). Then, a point object s_i can be pruned if it lies outside the T-expanded query, since it is guaranteed to have a qualification probability less than T. We note that this pruning is often better than using the Minkowski Sum as described in Section 5.3.1, since the p-expanded-query

usually has a smaller range.

If the U-catalog has to be used to find the p-expanded-query, we can use the maximum value of M in the U-catalog such that $M \leq T$. The M-expanded-query, which encloses T-expanded-query, can then be used for pruning: any object pruned by the M-expanded-query must also be pruned by the T-expanded-query.

5.4.2 Pruning Uncertain Objects for C-IUQ

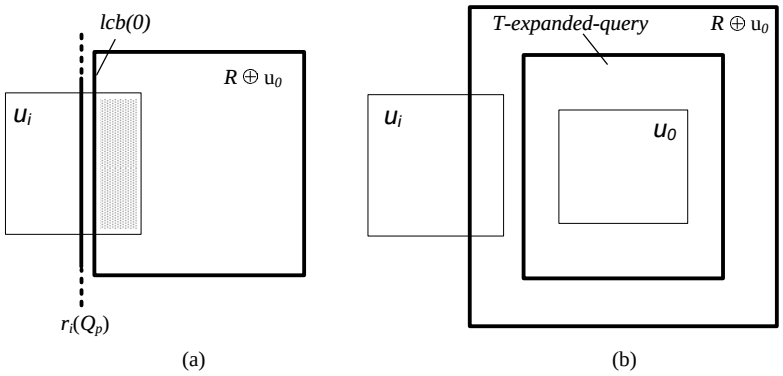


Figure 5.9: Pruning o_i for C-IUQ (a) using the $r_i(T)$ bound of o_i; (b) using the T-expanded-query.

We now investigate how the concept of p-bound and p-expanded-query can be used to facilitate pruning of an uncertain object o_i for C-IUQ. Three pruning strategies are possible.

- **Strategy 1: Use the p-bound of o_i.** Observe that we can prune o_i if the common region of u_i and $R \oplus u_0$ is on the right of $r_i(T)$. As shown in Figure 5.9(a), we can be sure that $\int_{u_i \cap (R \oplus u_0)} f_i(x, y)dxdy \leq T$. From

Lemma 5.4, we have

$$p_i = \int_{u_i \cap (R \oplus u_0)} f_i(x,y) \cdot Q(x,y) dxdy \quad (5.12)$$

$$\leq \int_{u_i \cap (R \oplus u_0)} f_i(x,y) dxdy \quad (5.13)$$

$$\leq T \quad (5.14)$$

Therefore, o_i can be removed from the result. If $r_i(T)$ cannot be found, then we find the maximum value M in the U-catalog of o_i such that $M \leq T$, and then use $r_i(M)$ instead of $r_i(T)$. Notice that $r_i(M)$ is on the right side of $r_i(T)$, so it may be possible that the shaded region crosses $r_i(M)$ but not $r_i(T)$. The same idea can be applied to other dimensions. For example, if $u_i \cap (R \oplus u_0)$ is at the lower part of $b_i(T)$, u_i can be pruned.

- **Strategy 2: Use the p-expanded-query.** o_i can be pruned if u_i is completely outside the T-expanded-query (Figure 5.9(b)). Recall from Definition 5.5 that any point object outside the p-expanded-query cannot have a qualification probability higher than p. Therefore, by Lemma 5.4,

$$p_i = \int_{u_i \cap (R \oplus u_0)} f_i(x,y) \cdot Q(x,y) dxdy \quad (5.15)$$

$$\leq T \int_{u_i \cap (R \oplus u_0)} f_i(x,y) dxdy \quad (5.16)$$

$$\leq T \quad (5.17)$$

If U-catalog is to be used, the M-expanded-query should be chosen, where M is the maximum value in the catalog such that $M \leq T$. Notice that this query range may be larger than that of T-expanded-

query, resulting in less efficient pruning.

Notice that both methods create more pruning opportunities than the query expansion technique described in Section 5.3.1, since o_i can be pruned even if they overlap with the Minkowski Sum of R and u_0.

- **Strategy 3: Use both the p-bound and the p-expanded query.** The third pruning strategy can be used when both Strategy 1 and Strategy 2 do not work. An example scenario is shown in Figure 5.10, where $R \oplus u_0$ crosses the $r_i(T)$ line, and u_i crosses the $lcb_(T)$ line. Hence, both pruning methods cannot be applied. We now show that it is still possible for o_i to be pruned. Let d_{min} be the minimum value in the U-catalog of o_i such that $d_{min} \geq T$ and $R \oplus u_0$ is on the right of $r_i(d_{min})$. Also let q_{min} be the minimum value in the U-catalog of O_0 such that $q_{min} \geq T$ and u_i is on the outside of the q_{min}-expanded query. By using Lemma 5.4, we have

$$\begin{aligned} p_i &= \int_{u_i \cap (R \oplus u_0)} f_i(x,y) \cdot Q(x,y) dxdy && (5.18) \\ &\leq q_{min} \int_{u_i \cap (R \oplus u_0)} f_i(x,y) dxdy && (5.19) \\ &\leq q_{min} \cdot d_{min} && (5.20) \end{aligned}$$

  Therefore, if the product of q_{min} and d_{min} is smaller than T, o_i can be pruned.

The three strategies just described involve some simple condition testing, and the size of an object's U-catalog is usually small (for example, in our experiments, we store six probability values and their p-bounds). Hence these methods can be used to prune uncertain objects efficiently.

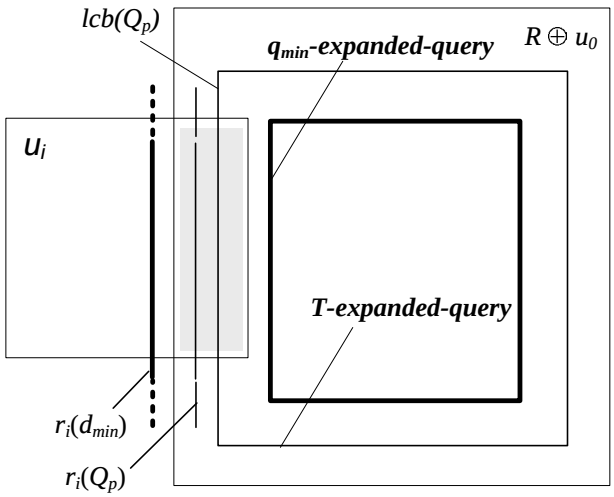


Figure 5.10: Pruning o_i for C-IUQ using both bounding box information of o_i and the expanded query.

5.4.3 Efficient I/O Solutions

To improve the I/O performance of constrained imprecise queries, the steps presented for Section 5.3.4 can be readily used. The only difference is that the expanded query range (i.e., the Minkowski Sum of R and u_0) is replaced by the T-expanded-query. The performance of query pruning is potentially better, since the T-expanded-query is usually smaller than $R \oplus u_0$.

The performance of C-IUQ can be further improved by using a data structure designed for storing uncertain data. Called *Probability Threshold Index* (PTI) in [26], the U-catalog information of uncertain objects under the same parent node of the R-tree is summarized . In each intermediate node, the minimum bounding rectangle (MBR(m)) for each probability value of m in the U-catalog is stored. This MBR(m) should be tight and enclose all the m-bounds for all children in the node. For example, suppose a node X that consists of two objects, O_1 and O_2, and in their U-catalogs the 0.3-bounds are stored. The left sides of their 0.3-bounds are $l_1(0.3)$ and $l_2(0.3)$ respectively.

If $l_2(0.3)$ is on the left of $l_1(0.3)$, then $l_2(0.3)$ is assigned to be the 0.3-bound for the node X.

With the aid of the PTI, we can apply the pruning techniques described in Section 5.4.2 in the index level. As described in Section 5.4.2, the U-catalog of an object can be used for pruning. We can use the same techniques with the U-catalog that resides in the intermediate node. This is because every p-bound stored in this U-catalog must enclose the p-bounds for each children. Moreover, our pruning techniques rely on the fact that the uncertainty region of the objects lie outside the p-bound. In Figure 5.9, for example, if $R \oplus u_0$ is on the right side of the m-bound of the U-catalog in the intermediate node, $R \oplus u_0$ must also be on the right side of $r_i(T)$, assuming u_i is stored under that intermediate node. In other words, if the p-bound in the intermediate node level satisfy the pruning condition, so does its children.

5.5 Experimental Results

We have performed experimental evaluation on the effectiveness of our approaches. We first present our simulation model, followed by the detailed results.

5.5.1 Experiment Setup

In our experiments, we use two realistic data sets, namely *California* and *Long Beach*.[4] The California data set contains 62K points. The Long Beach data set contains 53K rectangles. The objects in both data sets occupy a 2D space of $10,000 \times 10,000$ units. We use the California data set as a point object database, and the Long Beach data as an uncertain object database. One scenario could be that a police wants to find the suspect vehicles within

[4]Available at `http://www.census.gov/geo/www/tiger/`.

some distance from him. We also assume that the uncertainty pdf of any uncertain object (including the query issuer) is a uniform distribution. Each uncertain object is associated with a U-catalog, which consists of ten p-bounds for the probability values $0, 0.1, \ldots, 1$. The size of an R-tree node is 4K. Unless stated otherwise, R-tree is used for data indexing.

For each imprecise query tested here, both the uncertainty region of the query issuer (u_0) and the range query (R) have square shapes. For convenience, we denote the size of u_0 and R as u and w. Here, the term "size" refers to the half of the length of a square. By default, $u = 250$ and $w = 500$. The center point of u_0 is uniformly distributed in the data space. We also assume the probability threshold of an imprecise query is 0 (i.e., $T = 0$).

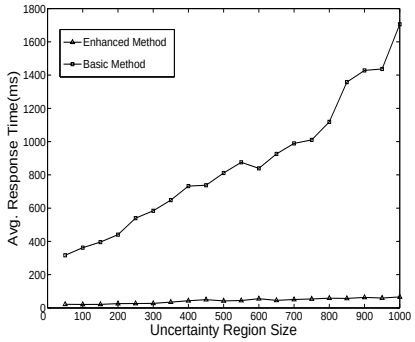


Figure 5.11: Basic vs. Enhanced (IUQ)

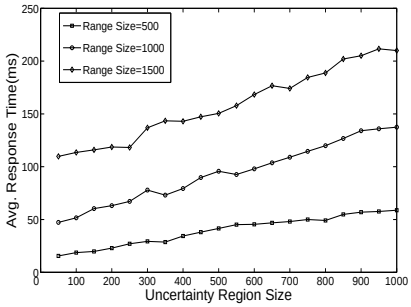


Figure 5.12: $Time$ vs. c (IPQ)

The performance metric used here is the total amount of time for executing a query (called "response time", $Time$ in short). Each data point is an average over 500 runs. All our experiments are run on a sunfire4800 server with four US-III+900MHz CPUs and 4096MB of memory. We use the R-tree provided by the Spatial Index Library version 0.44.2b [63].We also use this library to implement the PTI. Our simulation is written in $j2sdk1.4.2\_11$. Table 5.3 summarizes the parameters used in the experiments.

Param	Default	Meaning
u	250	Size of u_0
w	500	Size of range query
$f_i(x, y)$	$1/\|u_i\|$	o_i's uncertainty pdf (uniform)
T	0	Probability threshold
$Time$	$--$	Query response time

Table 5.3: Parameters and baseline values.

5.5.2 Results

Comparison with the basic solution. We first compare the performance of the basic solution described in Section 5.2.2, with our other techniques. Our experiments reveal that the basic solution is much more costly. Figure 5.11, for example, illustrates that the basic solution for calculating qualification probability (Equation 5.4) performs much worse than our solution (Equation 5.8). Similar results can be concluded for other queries. Next, we focus on the methods developed in Sections 5.3 and 5.4.

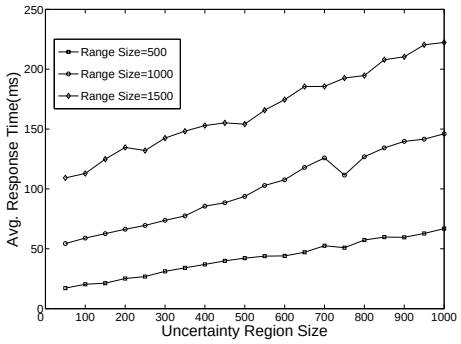


Figure 5.13: $Time$ vs. c (IUQ)

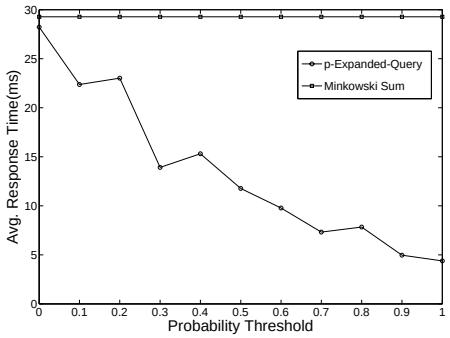


Figure 5.14: $Time$ vs. T(C-IPQ)

IPQ and IUQ. Let us examine the performance of IPQ using the methods developed in Section 5.3. Figure 5.12 presents the relationship between $Time$ and u, under different values of w. We observe that $Time$ varies from 20 to 220 ms, under a wide range of values of u and w. Also, $Time$

increases with both parameter values. Recall that the expanded query range is essentially the Minkowski Sum of R and u_0. It is enlarged when either w or u increases. As a result, more candidates are captured by the expanded query and more probability computations are required. Figure 5.13 shows a similar behavior for IUQ.

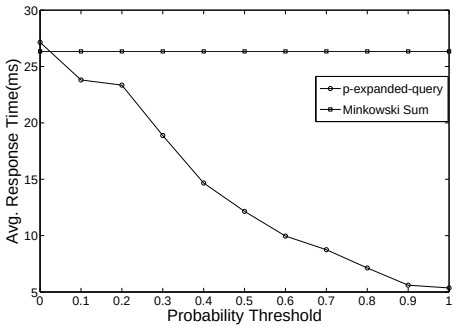


Figure 5.15: $Time$ vs. T(C-IUQ)

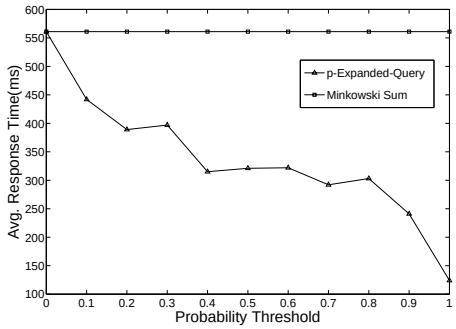


Figure 5.16: $Time$ vs. T (C-IPQ)

C-IPQ and C-IUQ. Next, we compare the performance of C-IPQ, using (1) the Minkowski-Sum, and (2) the p-expanded-query, which takes into account the probability threshold (T) of the C-IPQ. Figure 5.14 shows the performance under different values of T. In general, the performance of p-expanded-query improves with an increasing value of T. As discussed in Section 5.4.1, the p-expanded-query shrinks with a higher value of T. Hence the number of candidates that satisfy the p-expanded-query decreases, rendering a much better performance than using the Minkowski Sum alone (e.g., three times of improvement at $T = 0.6$).

In Figure 5.15 we investigate the performance of C-IUQ. The R-tree is used with the Minkowski Sum, while the PTI is used with the p-expanded-query. Again, we see the benefits of using p-expanded-query and PTI over methods that do not utilize probability threshold constraints, because they

allow much more data pruning for all values of T. For example, at $T = 0.6$, the performance gain is around 60%. Notice that this gain is smaller compared to C-IPQ. The reason is that it is usually harder to prune uncertain objects, whose uncertainty regions may occupy large amount of space, than point objects.

Non-Uniform Distribution. Finally, we examine the performance of queries when the uncertainty pdf is not uniform. We choose the Gaussian distribution, which is a common distribution used in modeling location uncertainty [76, 26]. For each object, the mean of the Gaussian distribution is the center of its uncertainty region, while the variance is one-sixth of the size of its uncertainty region.

When the uncertainty pdf is non-uniform, it is more expensive to evaluate than uniform distribution. In particular, Equations 5.5 and 5.8 may not have closed-form solutions, and numerical techniques are often required. We have used the Monte-Carlo technique for evaluation, where the positions of the query issuer and uncertain objects are sampled for a number of times, and the average result is obtained. In order to achieve an accurate result, we have performed a sensitivity analysis: we need at least 200 samples for evaluating a C-IPQ, and 250 samples for C-IUQ. Figure 5.16 shows the result for C-IPQ. We again see that the use of p-expanded-query achieves a better performance by exploiting the probability threshold constraint.

5.6 Chapter Summary

Computing qualification probabilities for imprecise location-dependent queries by using their definitions directly can be expensive. In this chapter, we proposed several techniques for improving the query performance. The use of the Minkowski Sum as the expanded query range enables pruning techniques

developed for traditional range query processing (such as pruning with a R-tree) to be used. The Query-Data Duality Theorem simplifies the cost of computing qualification probabilities for both IPQ and IUQ. When the Minkowski Sum and the Query-Data Duality Theorem is combined appropriately with the probability threshold constraint, more pruning opportunities can be created. In particular, if the probability information about the uncertain objects is pre-computed and stored in the PTI, pruning based on the objects' uncertainty information can be done in the index level.

In our future work, we will study how the proposed techniques could be extended for queries and uncertainty regions with non-rectangle shapes. Here we just give some comments on this possible extension. First of all, the Basic solution in Section 5.2.2 can work for queries and uncertainty regions with arbitrary shapes. For our efficient solutions, the query expansion method in Section 5.3.1 applies the Minkowski Sum of query region R and user location u_0, which is actually not restricted to the geometric shapes of R or u_0. The query-data duality property (Section 5.3.2) also exists for queries with non-rectangle shapes. For example, suppose R is the query issued by a user related to his/her location, we can generate a region R' wich is symmetrical to R with respect to the middle point of the line segment connecting the user location and the concerned data. Lemma 5.2 will still hold. The only change will come from Section 5.3.3, where most equations are based on rectangle shape. We will investigate how this part could be generalized in the near future. Finally, the efficient solutions for processing constrained C-IPQ and C-IUQ in Section 5.4 can be easily extended for queries and uncertainty regions with non-rectangle shapes, by simply approximating the orginal queries and uncertainty regions with appropriate MBR. All the pruning methods in that part can still work.

6 Cleaning Attribute-Uncertainty Data

We have presented our work for improving the efficiency of evaluating probabilistic queries over uncertain data. Now we begin to address the issues on optimizing query answer quality by reducing data uncertainty. In particular, we focus on cleaning uncertain data under the attribute-uncertainty model in this chapter. We will also disscuss how to clean tuple-uncertainty data in next chapter. In applications like sensor network monitoring and location-based services, due to limited network bandwidth and battery power, a system cannot always acquire accurate and fresh data from the external environment. To capture data errors in these environments, recent researches have proposed to model uncertainty as a probability distribution function. In this chapter, we present an entropy-based metric to quantify the degree of ambiguity of probabilistic query answers due to data uncertainty. Based on this metric, we develop a new method to improve the query answer quality. The main idea of this method is to acquire (or probe) data from a selected set of sensing devices, in order to reduce data uncertainty and improve the quality of a query answer. Given that a query is assigned a limited number of probing resources, we investigate how the quality of a query answer can attain an optimal improvement. To improve the efficiency of our solution, we further present heuristics which achieve near-to-optimal quality improvement. We generalize our solution to handle multiple queries. An experimental simulation over a realistic dataset is performed to validate our approaches.

6.1 Introduction

In many emerging and important applications like wireless sensor networks and location-based applications, the data obtained from the sensing devices

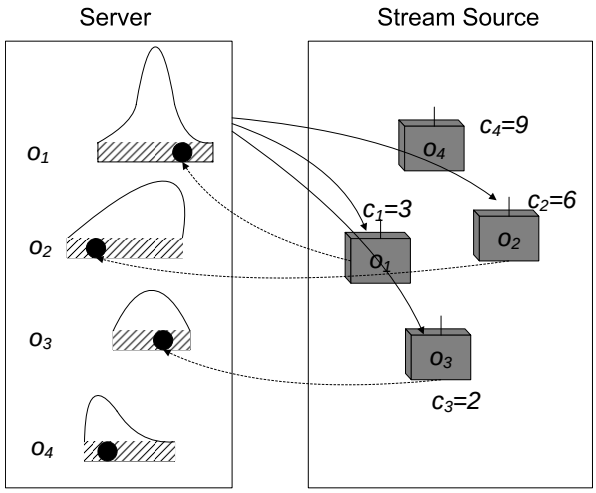


Figure 6.1: Probing of Sensor Data for Uncertainty Reduction.

are often imprecise [33, 67, 62]. To process uncertain data, probabilistic queries have been proposed, which produce imprecise results. For example, a probabilistic range query, inquiring which of the four sensor data values in Figure 6.1 have non-zero probabilities of being inside a specified range $[10^oC, 20^oC]$, may produce an answer like: $\{(o_1, 0.9), (o_2, 0.5)\}$. This answer indicates that o_1 (o_2) has a chance of 0.9(respectively 0.5) for having a value between $[10^oC, 20^oC]$.

How can we interpret the probability values of these query answers? Intuitively, these values reflect the *ambiguity* of a query result, due to the imprecision of the data being evaluated. In the previous example, since o_1 has a chance of 0.9 for satisfying the query, we know that o_1 is very likely to be located inside $[10^oC, 20^oC]$. The case of o_2 is more vague: it could either be inside or outside the specified range, with equal probabilities. In general, a query answer may consists of numerous probability values, making it hard for a query user to interpret the likelihood of their answers. A *quality metric* is desired, which computes a real-valued score for a probabilistic query

answer [23, 56]. This metric serves as a convenient indicator for the user to understand how vague his/her answer is, without the need of interpreting all the probabilities present in the answer. For example, if the score of his/her query answer is high, the user can immediately understand that the quality of his/her answer is good. In this chapter, we define a quality score for a probabilistic range query based on the definition of entropy [73]. This metric quantifies the degree of query answer uncertainty by measuring the amount of information presented in a query.

More importantly, the quality score definition enables us to address the question: "how can the quality of my query answer be improved?" Let us consider the sensor network example in Figure 6.1 again. Suppose that the sensors have not reported their values for a long time. As a result, the sensor data kept in the server have a large degree of uncertainty. Consequently, the query answer quality is low (i.e., the query answers are vague), and a user may request the server to give him/her an answer with a higher quality. To satisfy the user's request, the system can acquire (or **probe**) the current data from the sensors, in order to obtain more precise information (i.e., possibly with a smaller uncertainty interval). A higher quality score for the query user's answer can then potentially be attained. In fact, if all the items $(o_1, \ldots, o_4)$ are probed, then the server will have up-to-date knowledge about the external world, thereby achieving the highest query quality.

In reality, it is unlikely that a system can always maintain an accurate state of the external environment, since probing a data item requires precious resources (e.g., network bandwidth and energy). It is thus not possible for the system to probe the data from all the sources in order to improve the quality of a query request. A more feasible assumption is that the system assigns to the user a certain amount of "resource budget", which limits the

maximum amount of resources invested for a particular query. The question then becomes "how can the quality of a probabilistic query be maximized with probing under tight resource constraints?" To illustrate, let us consider Figure 6.1, where $c_1, \ldots, c_4$ are the respective costs for probing $o_1, \ldots, o_4$. The cost value of each sensor may represent the number of hops required to receive a data value from the sensor. Let us also assume that a query is associated with a resource budget of 8 units. If we want to improve the quality for this query, there are five *probing sets*, namely $\{o_1\}$, $\{o_2\}$, $\{o_3\}$, $\{o_1, o_2\}$ and $\{o_2, o_3\}$. Each of these sets describe the identities of the sensors to be probed. Moreover, the total sum of their probing costs is less than 8 units. Now, suppose the probing of o_2 and o_3 will yield the highest quality improvement. Then the system only needs to probe these two sensors, to ensure the maximum benefit.

Since testing the possible candidates in a brute-force manner requires an exponential-time complexity, we propose a polynomial-time solution based on dynamic programming. We also present a greedy solution to enhance scalability. Our experimental results show that the greedy solution achieves almost the same quality as the dynamic-programming solution. We study this problem for probabilistic range queries, which return the items within a user-defined region. This query is one of the most important queries commonly found in location-based services and sensor applications. Our solution can generally be applied to any multi-dimensional uncertain data, where the pdf's are arbitrary.

The problem studied in this chapter addresses the balance between query quality and the amount of system resources consumed. A few related problems have been studied in [33, 61], where probing plans are used to direct the server to acquire the least number of data items required to achieve the

highest quality. However, these work do not consider the issue of maximizing quality under limited system resources allocated to a user. We further consider the scenario in which a group of query users share the same resource budget. This represents the case when a system allocates its resources to users with the same priority. We explain how our basic solution (tailored for a single query) can be extended to address this. To our understanding, this has not been studied before.

To summarize, our major contributions are:

1. We propose an entropy-based quality metric for probabilistic range queries.

2. We develop optimal and approximate solutions that maximize the quality of a probabilistic query under limited resource constraints.

3. We extend our solution to handle the case where multiple query users share the same resource budget.

4. We conduct extensive experiments with realistic datasets to validate the performance of our algorithms.

The rest of this chapter is organized as follows. Section 6.2 illustrates the system architecture. We discuss the details of quality and resource budget for probabilistic range queries in Section 6.3. Then we give our solutions in Section 6.4. We report our experimental results in Section 6.5. Section 6.6 concludes the chapter.

6.2 System Architecture

Figure 6.2 describes the architecture of the system used in this chapter. The *Data Manager* caches the value ranges and corresponding pdf of remote

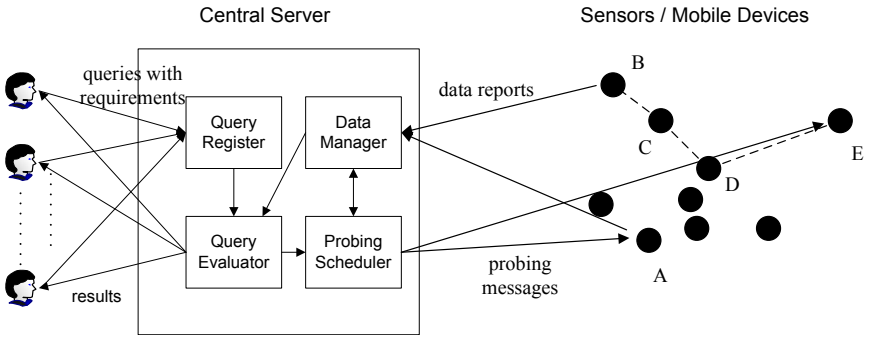


Figure 6.2: System Architecture

sensors. The *Query Register* receives queries from the users. The *Query Evaluator* evaluates the queries based on the information stored in the *Data Manager*. The *Probing Scheduler* is responsible for generating a *probing set* for each query – essentially the set of sensors to be probed. The benefits and costs of probing actions will be taken into account by the *Probing Scheduler* in deciding the what sensors to be consulted. More specifically, a user query is handled in four major steps:

- Step 1. The query is evaluated by the *Query Evaluator* based on the data cached in the *Data Manager*.
- Step 2. The *Probing Scheduler* decides the content of probing set.
- Step 3. The *Probing Scheduler* sends probing commands to the sensors defined in the probing set.
- Step 4. The *Query Evaluator* reevaluates the query based on the refreshed data returned to the *Data Manager*, and returns results to the query issuer.

We assume after probing, the value of the data item becomes "precise" (i.e., has a pdf value equal to one inside an infinitesimally-thin uncertain

region). For simplicity, we illustrate our solution with an uncertainty model for one-dimensional data, but our methods can easily be extended to handle multi-dimensional data.

6.3 Quality and Resource Budget of Probabilistic Queries

In this section, we present the notion of *Quality Score* for probabilistic range query, the query that we extensively study in this chapter. In Section 6.3.1 we present a quality metric for probabilistic range queries. Section 6.3.2 then discusses the metric of resource constraints, called *Resource Budget*, which is assigned to each query as the maximum amount of resources allowed in the process of query evaluation.

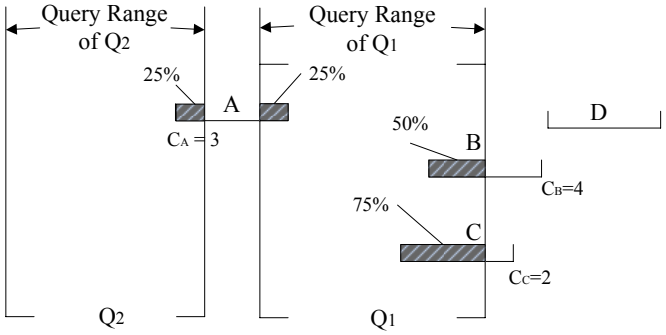


Figure 6.3: An Example of Probabilistic Range Query

6.3.1 Quality Score

The Probabilistic Range Query (PRQ)[23] returns a set of data objects, with the probabilities that their attribute values are in the specified range. To illustrate, Figure 6.3 shows two PRQ's on data items A, B, C and D. The uncertainty region of each item is shown. For query Q_1, three items (A, B and

C) are included in the result; item D is excluded since its uncertainty region does not overlap with the query range, yielding zero qualification probability.

Let us now present a metric to measure the quality of the answer of a probabilistic range query. This metric is based on the notion of information entropy[73]. As a brief review, the information entropy measures the average number of bits required to encode a message, or the amount of information carried in the message:

Definition 6.1 Entropy: *Let X_1, ..., X_n be all possible messages, with respective probabilities $p(X_1)$, ..., $p(X_n)$ such that $\sum_{i=1}^{n} p(X_i) = 1$. The entropy of a message $X \in \{X_1, ..., X_n\}$ is:*

$$H(X) = -\sum_{i=1}^{n} p(X_i) log_2 p(X_i) \tag{6.1}$$

Recall that in the answer of PRQ, each value p_i describes the probability that object o_i satisfies it. Thus there are two possible events: (1) o_i satisfies the PRQ with a probability as p_i; (2) o_i does not satisfy the PRQ with a probability of $1 - p_i$. Therefore, the ambiguity of o_i for satisfying a PRQ is

$$g_i = -p_i log_2 p_i - (1 - p_i) log_2 (1 - p_i) \tag{6.2}$$

We then use the sum of the nagative entropy values for all the objects that satisfy the PRQ with non-zero probabilities as the quality metric. More specifically, for a result containing n answers $(o_1, p_1), \cdots, (o_n, p_n)$, the *quality score*, denoted by H, of this result is defined by

$$H = \sum_{i=1}^{n} (p_i log_2 p_i + (1 - p_i) log_2 (1 - p_i)) \tag{6.3}$$

By substituting Equation 6.2 into Equation 6.3, we have

$$H = -\sum_{i=1}^{n} g_i \tag{6.4}$$

A lower value of H implies a lower quality. In particular, H is equal to zero if the result is precisely known, which happens when all the p_i's are equal to zero or one. The range of H is $[0, n]$. Notice that after probing item o_i, its uncertainty region shrinks to a point, and the server knows exactly whether o_i satisfies the range query. Thus, p_i equals to either zero or one. The corresponding ambiguity caused by the answer (o_i, p_i) is then "removed", and the entropy of the overall query result is reduced by an amount given by Equation 6.2 . We denote this amount of quality improvement as the *gain* of probing o_i, denoted by g_i. As shown in Equation 6.2 the value of g_i only depends on the qualification probability of a single object o_i. Moreover, the gain is only non-zero for we choose items that have qualification probabilities in (0,1), and the gain of probing a set of items is simply equal to the sum of their gains.

6.3.2 Resource Budget

We now present the *resource budget* model of a query, which limits the amount of resources that can be used to probe the sensing devices for this query.

In general, there are several types of important resources for a wireless sensor network, such as network bandwidth and the battery power used to transmit data. Here we use a single metric, namely the number of transmitted messages, to measure the cost. The number of transmitted messages for probing an item is the major source of consumption of the important resources. The more number of times the sensors are probed, the more amount

Symbol	Description
Q	Probabilistic range query
C	Resource constraint assigned to Q
c	# of messages for probing o
H	Precision quality (entropy)
g_i	The benefit of probing o_i $(i = 1 \cdots n)$

Table 6.1: Symbols for Cleaning Attribute-Uncertainty Data

of network bandwidth and battery power is required. Thus, we assume the server assigns to a query the maximum number of transmitted messages allowed as its *resource budget*, denoted as C.

The transmission cost of a data item can vary among the sensors. For example, a message generated from a sensor may need different number of hops to reach the base station. Figure 6.2 shows that four hops are required for probing item E (the dashed path), whereas only one hop is needed to probe item A. Thus probing E will cost more than A. We assume the server knows how many messages are needed for probing an item. We also use c_i to denote the number of messages for probing o_i. We list the notations used in this chapter in Table 6.1.

6.4 Maximizing Quality with Limited Resources

As we have mentioned in Section 6.3.1, probing items that have non-zero qualification probabilities can often improve the quality of a query result. In general, there can be a tremendous number of objects present in the answer. Moreover, the amount of resource budget available probing is limited. In this section, we discuss how query quality can be maximized with limited resource budgets.

In Section 6.4.1 we present the *Single Query* (SQ) problem, where we explain how probing can be done efficiently for a query with limited resource

budgets. We then extend our solution to support a more complicated and practical scenario, i.e. *Multiple Queries with Shared Budget* (MQSB), in Section 6.4.3. We give heuristics which provide close-to-optimal performance in Section 6.4.4.

6.4.1 Single Query (SQ)

In this scenario, only one query, Q, needs to be considered when choosing sensors. Suppose based on the cached data, the Query Evaluator has calculated the qualification probabilities,$\{p_1, \cdots, p_n\}$, of all the items $o_i (i = 1, \cdots, n)$ such that $p_i > 0$. The cost of probing o_i is c_i. Let the gain obtained by probing o_i be g_i (Equation 6.2). We formally define the *Single Query (SQ) problem* as follows.

Maximize

$$\sum_{i=1}^{n} b_i \cdot g_i \tag{6.5}$$

Subject to

$$\sum_{i=1}^{n} b_i \cdot c_i \leq C \tag{6.6}$$

Here we use an array $\{b_k | k = 1, \cdots, n\}$ to record the choices. Initially, all the values of b_i are zero. If item o_i is chosen for probing, we set b_i to 1.

To solve the SQ problem, we use dynamic programming. We observe that this problem has the *optimal substructure*, meaning that the optimal solutions of subproblems can be used to find optimal solutions of the SQ problem. Let us rewrite the SQ problem as $P(C, N)$, which is associated with a resource budget C and items $N = \{o_1, \cdots, o_n\}$, whose p_i's are all nonzero. Suppose we have found the optimal set $S = \{o_{\gamma_1}, \cdots, o_{\gamma_m}\}$ $(m \leq n \wedge \gamma_i \in [1, n])$ for $P(C, N)$: among all the subsets of N whose costs are not larger than C, S

is the one with the highest gain. Now we define a subproblem by randomly removing an item, e.g. o_{γ_1}, from the candidate item set, and reducing the budget to $C - c_{\gamma_1}$. That is, we consider a subproblem $P(C - c_{\gamma_1}, N/\{o_{\gamma_1}\})$. If $S_1 = S/\{o_{\gamma_1}\} = \{o_{\gamma_2}, \cdots, o_{\gamma_m}\}$, is the optimal set for this subproblem, the SQ problem can be solved by using the dynamic programming framework. Next we prove that S_1 must be the optimal set for $P(C - c_{\gamma_1}, N/\{o_{\gamma_1}\})$.

Proof : Suppose S_1 is not the optimal set for $P(C - c_{\gamma_1}, N/\{o_{\gamma_1}\})$, then we can find another set $S_1' \neq S_1$ which meets two requirements: (1) the cost of probing S_1' is not larger than $C - c_{\gamma_1}$ and (2) the gain of probing S_1' is higher than that of probing S_1. Consider the set $S_1' \cup \{o_{\gamma_1}\}$. Its cost is not larger than $C - c_{\gamma_1} + c_{\gamma_1} = C$. The gain of probing it is higher than that of probing the set $S_1 \cup \{o_{\gamma_1}\}$, or S. Thus $S_1' \cup \{o_{\gamma_1}\}$ should be a better choice than S for the overall problem, which violates the condition that S is the optimal set. So S_1 must be the optimal set for $P(C - c_{\gamma_1}, N/\{o_{\gamma_1}\})$. □

6.4.2 Algorithm DP.

In this algorithm, we look for the optimal set for each subproblem denoted by $P(k, i)$, where the resource budget equals to k and the candidate item set is $\{o_1, ..., o_i\}$. There are totally $n \cdot C$ subproblems. For the subproblems with zero budget or empty candidate set, the optimal set is also an empty set. We use an array s to store the optimal sets for the subproblems, where $s[k, i]$ is the optimal set for the subproblem $P(k, i)$. Each element of s, e.g. $s[k, i]$, is also an array, where $s[k, i][j] = 1$ if o_j is chosen for probing, and zero otherwise. We also use an array v to store the gain by probing the optimal set $s[k, i]$. For each data item o_i, there are two possible choices. Either o_i is not chosen and $s[k, i-1]$ is considered as the optimal set for $P(k, i)$, or this item is put into the solution set which contributes g_i to the solution

gain but decrease the budget remaining for items $\{o_1, \cdots, o_{i-1}\}$ to $k - c_i$. The optimal set for the subproblem $P(k - c_i, i - 1)$ is $s[k - c_i, i - 1]$ with the gain $v[k - c_i, i - 1]$. Thus if o_i is chosen, the maximum possible gain is $v[k - c_i, i - 1] + g_i$. In Step 3, the gains of these two possible choices are compared, and the one with larger gain is taken as the optimal solution for current subproblem $P(k, i)$. Steps 4-5 handle the case that o_i is not chosen, while Steps 7-9 construct the optimal set and the corresponding gain if o_i is chosen. Another point to notice is, in order to put o_i into the solution set, the cost of probing o_i, i.e. c_i, must be not larger than the remaining budget k. Step 3 also tests whether this condition is satisfied.

```
   input  : An array of probing costs c = (c_1, ..., c_n)
   input  : An array of gains g = (g_1, ..., g_n)
   input  : The resource budget C
   output: The optimal set
 1 for i ← 1 to n do
 2     for k ← 1 to C do
 3         if c_i > k or v[k, i − 1] > v[k − c_i, i − 1] + g_i then
 4             v[k, i] ← v[k, i − 1];
 5             s[k, i] ← s[k, i − 1];
 6         else
 7             v[k, i] ← v[k − c_i, i − 1] + g_i;
 8             s[k, i] ← s[k − c_i, i − 1];
 9             s[k, i][i] ← 1;
10 return s[C, n];
```

Algorithm 5: Algorithm DP for SQ

Using Algorithm 5, we can find an optimal solution for the SQ problem. We will show soon that Algorithm 5 can also be used to solve the MQSB problem, with little change to the calculation of *gain*.

Complexity. There are two for-loops in Algorithm 5. The computa-

tional complexity is thus $O(nC)$. The algorithm requires the storage of intermediate results, i.e. the optimal sets and corresponding gains for the subproblems. The variable s is a $3D$ array with space complexity of n^2C, while v is a $2D$ array with the size of nC. Thus the memory complexity of Algorithm 5 is $O(n^2C)$.

6.4.3 Multiple Queries with Shared Budget (MQSB)

In many cases, more than one query are processed at the server simultaneously. A data item o_i may be involved in the results of multiple queries. By probing o_i, all queries containing it in their results will have a better quality. In order to apply Algorithm 5 in this scenario, we need to change the method of calculating gain, i.e. Equation 6.2. Suppose there are m queries, $Q_1, \cdots, Q_m$, we can have a set of m values for o_i, $p_{i1}, \cdots, p_{im}$, where $p_{ij}(j = 1, \cdots, m)$ specifies the probability that o_i satisfies Q_j. After getting the exact value of o_i the result precision of these queries will be improved by g_{ij}. Here g_{ij}, the gain for Q_j obtained by probing o_i, is equal to $-p_{ij}log_2p_{ij} - (1 - p_{ij})log_2(1 - p_{ij})$, where $j = 1, \cdots, m$(Equation 6.2). The gain of probing o_i is the sum of g_{ij}, or

$$G_i = \sum_{j=1}^{m} g_{ij} \tag{6.7}$$

For example, as in Figure 6.3, item A overlaps with the ranges of both Q_1 and Q_2, where $p_{A1} = p_{A2} = 0.25$. Thus $g_A = -2 \cdot (0.25 \cdot log_20.25 + 0.75 \cdot log_20.75) = 1.62$.

Suppose the server needs to process multiple queries in batches, and these queries share a single resource budget C. We denote this scenario as *Multiple Queries with Shared Budget, MQSB*. The formal definition of MQSB has the

same form as that of SQ. The only difference is the use of G_i (Equation 6.7) to replace g_i (Equation 6.2). Therefore, Algorithm 5 is also suitable for solving MQSB. Moreover, the approximate solutions, which will be discussed in Section 6.4.4, can also be used for MQSB.

Complexity of DP (MQSB). Compared with the SQ scenario, the inputed data size for the DP algorithm will be larger in the MQSB scenario. There are m queries evaluated concurrently in the MQSB scenario. If we let n to be the average size of the result sets for these m queries, the DP algorithm needs to process nm data items. Moreover, there would be extra cost of computing the gains by using Equation 6.7 in the MQSB scenario, which is $O(nm)$. Thus, the computational complexity of Algorithm 5 would be $O(nmC + nm) = O(nmC)$ in the MQSB scenario, and the memory complexity is $O((nm)^2C)$.

6.4.4 Approximate Solutions

The dynamic programming solution, Algorithm DP, can find the optimal sets. However, its complexity can be quite high. To enhance its scalability, we design a greedy algorithm. The general idea of Greedy is to make a locally optimal choice. Every unit of cost should be allocated to the items which can produce maximum benefit. To achieve this objective, we define a new metric to describe the amount of gain obtained by consuming a unit of resource. This metric is called *efficiency*, denoted by e_i. Equation 6.8 shows how to compute the value of e_i.

$$e_i = \frac{g_i}{c_i} \tag{6.8}$$

```
   input  : An array of probing costs c = (c_1, ..., c_n)
   input  : An array of gains g = (g_1, ..., g_n)
   input  : The resource budget C
   output: The probing set
1  d ← sort(c, g);
2  B ← C;
3  for i ← 1 to n do
4      if B ≥ c_{d[i]} then
5          s[d[i]] ← 1;
6          B ← B − c_{d[i]};
7  return s;
```

Algorithm 6: Algorithm Greedy

In Step 1 of the Greedy algorithm, the items are sorted by their efficiencies in descending order. The sorted indices are stored in an array d. Initially, the remaining budget, i.e. B, is set to the value of C. We then check the items sequentially in the order stored in d. If the remaining budget is not smaller than the cost of probing this item (Step 4), it is put into array s (Step 5) and the remaining budget is reduced by its cost (Step 6). Step 7 returns the probing set stored in s.

The Greedy algorithm has a time complexity of $O(n \log n)$ (to sort the items). The space requirement for Greedy is $O(n)$. It is thus more efficient than DP. However Greedy does not guarantee an optimal set can be found.We will compare the performance of these two algorithms in Section 6.5.

6.4.5 Random and MaxVal

We also develop two other simpler heuristics, called *Random* and *MaxVal*. The Random solution chooses items randomly until the resource budget is exhausted. The MaxVal heuristic probes items sequentially in descending order of their gains until the resource budget is exhausted.

Algorithm	Computational Complexity	Space Complexity
DP	$O(nC)$	$O(n^2C)$
Greedy	$O(n \log n)$	$O(n)$
Random	$O(n)$	$O(n)$
MaxVal	$O(n \log n)$	$O(n)$

Table 6.2: Complexity of Four Algorithms (SQ)

Algorithm	Computational Complexity	Space Complexity
DP	$O(nmC)$	$O((nm)^2C)$
Greedy	$O((nm) \log(nm))$	$O(nm)$
Random	$O(nm)$	$O(nm)$
MaxVal	$O((nm) \log(nm))$	$O(nm)$

Table 6.3: Complexity of Four Algorithms (MQSB)

Table 6.2 compares the complexities of the above algorithms in the SQ scenario. For MQSB, the complexities of the optimal and approximate solutions are listed in Table 6.3. They are derived by substituting the value of n in Table 6.2 by nm.

6.5 Experimental Results

We have performed experimental evaluation on the effectiveness of our approaches. We first present our simulation model, followed by the detailed results.

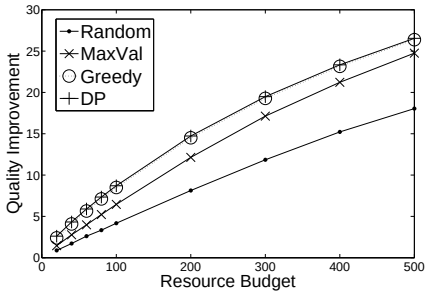


Figure 6.4: Quality Improvement vs. Resource Budget (SQ)

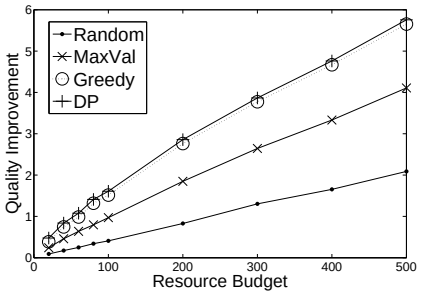


Figure 6.5: Quality Improvement vs. Resource Budget (MQSB)

6.5.1 Experimental Setup

We use a realistic data set, called Long Beach[5], which contains 53K rectangles, and each represents a region in the Long Beach country. The objects occupy a 2D space of $10,000 * 10,000$ units. We use the Long Beach data as an uncertain object database. We also assume that the uncertainty pdf of any uncertain object is a uniform distribution.

The cost of probing each item (i.e. c_i) are uniformly distributed in $[1, 10]$. The resource budget, C, ranges from 20 to 500. The performance metric is the result quality improved by probing a set of result items. Each data point is an average over 50 runs. Our experiments are run on a PC with 2.4GHz CPU and 512MB of main memory. Our simulation is written in j2sdk1.4.2_11.

6.5.2 Results

Effectiveness Analysis. Figure 6.4 compares the quality improvement using different probing strategies for the SQ problem. The x-axis is the value of resource budget which ranges from 20 to 500. The y-axis is the improved quality of query results. As shown in Figure 6.4, DP always outperforms MaxVal and Random. This is because DP derives the probing set with optimal resource utilization. The performance of Greedy is close to DP; in fact, DP performs about only 2% to 3% better than Greedy. This is because that the quality-aware probing problem is a variant of the knapsack problem [29], and it has been shown in [35] that the average performance of a greedy solution is close to the optimal one.

Figure 6.5 illustrates similar results for MQSB. In these experiments, 10 queries are executed concurrently in a batch.

[5]Available at http://www.census.gov/geo/www/tiger/

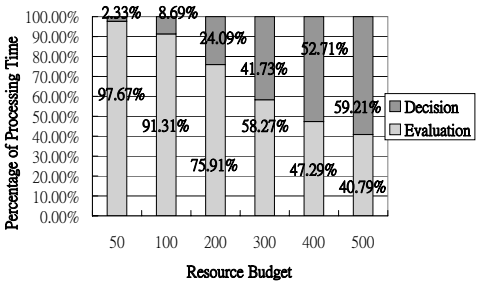


Figure 6.6: Time Spent in Different Phases during Query Processing (SQ).

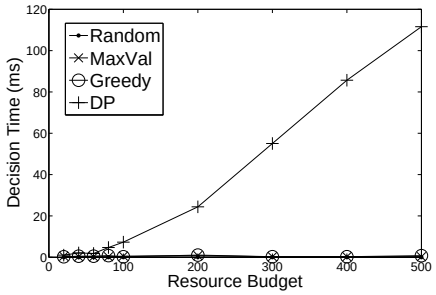


Figure 6.7: Decision Time vs. Resource Budget (SQ)

Performance Analysis. Figure 6.6 shows a decomposition of the time spent in the server: (1) Evaluation - the time required by the Query Evaluator to compute the initial results (Step 1 in Section 6.2), and (2) Decision - the time for deciding the probing set contents (Step 2 in Section 6.2). We have ignored the processing time of Step 4 since after probing, the qualification probabilities will become either zero or one for the data items in the probing set, and no extra effort is needed to compute their qualification probabilities. Here, we use DP to find optimal probing set in the Decision step. As shown in Figure 6.6, the Decision step costs more time as the resource budget increases. The reason is that more choices are available with a larger resource budget.

Figure 6.7 shows the time spent in the Decision step for the SQ problem. DP uses more time to find the optimal probing sets, and the decision time of DP increases fast with the resource budget. The decision time for heuristics (i.e. Greedy, MaxVal and Random) are much less. The results in MQSB are similar and are omitted here.

Compared with DP, Greedy gets similar quality improvement with less time. In fact, Greedy performs very well under a large batch of queries in the MQSB scenario. Figure 6.8 illustrates the time required for finding the probing sets using the Greedy algorithm. The resource budget is set to 100.

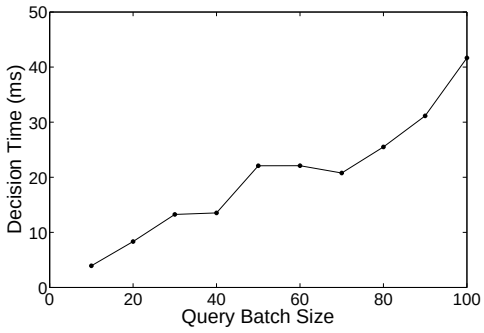


Figure 6.8: Scalability of MQSB (Greedy).

The number of queries evaluated in a batch varies from 10 to 100. As shown in the figure, the decision time increases gracefully with the query batch size.

6.6 Chapter Summary

The evaluation of probabilistic queries over uncertain data has attracted plenty of research interest in recent years. In this chapter, we investigated the problem of optimizing the quality of probabilistic query answers with limited resources. We further extended our solution to handle the case where the resource budget is shared among multiple queries. While the DP algorithm provides an optimal solution in polynomial times, our experiments show that the Greedy heuristic can achieve close-to-optimal performance in less time. In the future, we will investigate this problem for other types of queries.

7 Cleaning Tuple-Uncertainty Data

In this chapter, we will address the issues of cleaning *tuple-uncertainty* data. Similarly as in Section 6, our objective is to optimize result quality with a limited amount of resources. For this purpose, we present the *PWS-quality* metric under the *possible world semantics*, which is a universal measure that quantifies the ambiguity of query answers over uncertain data under tuple-uncertainty. We study how PWS-quality can be efficiently evaluated for two major query classes: (1) queries that examine the satisfiability of tuples independent of other tuples (e.g., range queries); and (2) queries that require the knowledge of the relative ranking of the tuples (e.g., `MAX` queries). We then propose a polynomial-time solution to achieve an optimal improvement in PWS-quality. Other fast heuristics are presented as well. Experiments, performed on both real and synthetic datasets, show that the PWS-quality metric can be evaluated quickly, and that our cleaning algorithm provides an optimal solution with high efficiency. To our best knowledge, this is the first work that develops a quality metric for a probabilistic database, and investigates how such a metric can be used for data cleaning purposes.

7.1 Introduction

In this chapter, we address the problem of cleaning tuple-uncertainty data, i.e. *probabilistic databases* [6, 30], for achieving better query or service quality, under a limited budget. Despite the importance of uncertain database cleaning, relatively little work has been done in the area (e.g., [67, 33, 51, 4, 19]). The techniques for cleaning attribute-uncertainty data in Section 6 cannot be readily utilized here. The reasons are two-folds. Firstlly, the query semantics are different. The queries over attribute-uncertainty data retrieves entities,

while those over tuple-uncertainty data are fed with tuples. Futhermore, the quality metric in Section 6 is designed only for probabilistic range query. In this chapter, we propose *PWS-quality*, a universal measure quantifing the ambiguity of query answers over uncertain data under tuple-uncertainty. Our main idea is to make use of the query information to decide the set of data items to be cleaned. By operating on these data, the quality of the answers returned to the user can attain the highest improvement. We develop our solution based on the *probabilistic database* [6, 30], a popular model for uncertain data cleaning and integration [3, 84, 53, 36].The main challenges that we address include: (1) define a sound and general quality metric over query results; (2) develop efficient methods to compute this metric; and (3) devise efficient and optimal cleaning algorithms.

To illustrate, Table 7.1 shows a relation in a probabilistic database, which stores the quotations of four products (with IDs a, b, c and d), collected from webpages by using some automatic schema matching methods. An attribute called *existential probability* (Prob. in short) is used to indicate the confidence of the existence of each tuple. A tuple is also associated with an "x-tuple" [3], which represents a distribution of alternatives. For example, product a has a 0.7 chance for offering a price of \$120, and a 0.3 chance for having a quotation of \$80. Now consider a `MAX` query: "Return the tuple with the highest `price`". Due to data imprecision, this query can produce imprecise answers. Table 7.2 shows the query result, which contains the IDs of the tuples, and their non-zero qualification probabilities for being the correct answers.

Based on the answer probabilities, a real-valued "quality score" can be defined to capture the degree of ambiguity of a query answer. For example, the score of the `MAX` query result (Table 7.2) is -1.73 (according to our

Product ID	Tuple ID	Price ($)	Prob.
a	a_1	120	0.7
a	a_2	80	0.3
b	b_1	110	0.6
b	b_2	90	0.4
c	c_1	140	0.5
c	c_2	110	0.3
c	c_3	100	0.2
d	d_1	10	1

Table 7.1: Uncertain database example.

Tuple	Qualification Probability
a_1	0.35
b_1	0.09
c_1	0.5
c_2	0.09
c_3	0.024

Table 7.2: Results of the `MAX` query on Table 7.1.

quality metric). Suppose Table 7.1 is partially cleaned (e.g., by consulting the companies about the actual prices of the products). Table 7.3 shows one possible scenario, where the uncertainties associated with x-tuples a and c are removed. In this table, only one tuple exists for each of a and c, and the existential probability of this tuple is equal to one. The new result of the `MAX` query is shown in Table 7.4, with a lower ambiguity, or an improved quality score of -0.97. In the extreme, if all the x-tuples are cleaned, the quality score becomes the highest (a value of zero with our metric).

Product ID	Tuple ID	Price ($)	Prob.
a	a_2	80	1
b	b_1	110	0.6
b	b_2	90	0.4
c	c_3	100	1
d	d_1	10	1

Table 7.3: A partially-cleaned instance of Table 7.1.

How should such a query quality metric be defined? Although some quality measures have been proposed before, they are either catered for specific

Tuple	Qualification Probability
b_1	0.6
c_3	0.4

Table 7.4: Results of the `MAX` query on Table 7.3.

types of queries (e.g., [23, 33, 26, 74]), or not designed for use in a probabilistic database (e.g., [32, 42]). To solve these problems, we propose the *PWS-quality*. This metric provides a *universal measure* of query quality (i.e., can be used by any queries) for the probabilistic database. It is essentially an entropy function [73], which returns a real-valued score for conveniently indicating the amount of impreciseness in query answers. The PWS-quality also enables efficient data cleaning solutions, as we will show in this chapter.

Another salient feature of PWS-quality is that it assumes the *Possible-World Semantics* (or PWS in short). The PWS provides a formal interpretation of the probabilistic data model [30], where a database is viewed as a set of deterministic database instances (called *possible worlds*), each of which contains a set of tuples extracted from each x-tuple. An example possible world for Table 7.1 contains the tuples $\{a_1, b_2, c_3, d_1\}$. Query evaluation algorithms for a probabilistic database should follow the notion of PWS, i.e., the results produced should be the same as if the query is evaluated on all the possible worlds [33]. Analogously, the PWS-quality score is calculated based on the query results obtained from all the possible worlds.

One apparent problem about the PWS-quality is that it is inefficient to calculate. This is because evaluating this measure requires examining all possible worlds, the number of which can be exponentially large [6, 30]. Interestingly, we observe that it is not often necessary to examine all the database instances; the PWS-quality can, in fact, be computed by using the query answers returned to the user. This is true for a broad class of queries known as the *entity-based* query [23]. This kind of query has the property

that the final answer returned to the user contains the IDs of the tuples that satisfy it, as well as their qualification probabilities (e.g., Table 7.2). We study two representative examples of entity-based queries, namely, the range query and the `MAX` query. Both queries are used in many applications. For example, in a sensor-monitoring application, a range query can be: "Return the IDs of the sensors whose temperature values are within $[10^oC, 20^oC]$". In the movie database, a `MAX` query can be: "Return the ID of the movie-viewer whose rating is the highest". We show that the PWS-quality of these two queries can be quickly computed by using query answer information. Our methods are effective because a query answer can be efficiently generated by existing query evaluation and indexing algorithms, and the complexity of our technique is linear to the size of the query answer.

The PWS-quality also serves as a useful tool for solving the data cleaning problem. Given the set of x-tuples to be cleaned, we prove that there is always a monotonic increase in the expected value of PWS-quality. This helps us to formulate the data cleaning problem as: choose the subset X of x-tuples such that (1) the increase in the expected quality of cleaning the x-tuples in X is the highest; and (2) the total cost of cleaning X does not exceed a given budget. This problem is challenging because calculating the expected quality improvement of X requires the processing of all combinations of the tuples in X. Moreover, a naïve approach of finding the optimal set X requires the testing of different combinations of x-tuples in the database, rendering an exponential time complexity. To solve these problems, we convert the PWS-quality expression into an "x-form" – a linear function of the probability information of the x-tuples. The x-form allows us to compute the expected quality improvement for cleaning a set of x-tuples easily. Moreover, it has the same format for both the range and the `MAX` queries (with different param-

eters), so that only a single solution is needed to support both queries. To find the optimal solution without testing all combinations of x-tuples from the whole database, we show that it is only necessary to select the x-tuples whose tuples appear in a query answer. We then model the cleaning task as an optimization problem, and develop a dynamic-programming-based algorithm, in order to deduce the optimal set of x-tuples in polynomial time. We also propose other approximate heuristics (such as the greedy algorithm). Our algorithms serve both the range and the `MAX` queries. They also support databases that contain tuples with the same attribute value.

We have performed detailed experimental evaluation to examine our approaches. For both real and synthetic datasets, the results show that PWS-quality can be efficiently computed. Moreover, x-tuples can be quickly selected to achieve an optimal improvement in expected quality. Among the heuristics, the greedy algorithm provides a close-to-optimal performance with the highest efficiency.

Figure 7.1 illustrates a system design that incorporates our solution. The query engine, upon receiving a user's request, produces a probabilistic query answer. This information is passed to the *quality manager*. Inside this module, the *quality evaluator* computes the PWS-quality score. It then sends the necessary information to the *data cleaning algorithm*, which derives the optimal set of x-tuples to be cleaned (or "cleaning set"), with the available budget considered. The *cleaning manager* is responsible for performing the sanitization activity (e.g., requesting the selected sources to report their updated values). The query, when executed again, will then have an improvement in the expected quality. [6] Notice that the quality manager is decoupled from the query engine, since it only requires the probability information of the

[6]When the query is re-run on the refreshed database, a full evaluation is not needed, as explained in Section 7.2.

answer tuples. Another issue is that the PWS-quality score is also sent to the user. This real-valued rating provides an intuitive way for the user to understand the degree of ambiguity in his results, without interpreting the numerous probability values that may appear in his query answers. In this chapter, we focus on the design of the quality evaluator and the data cleaning algorithm.

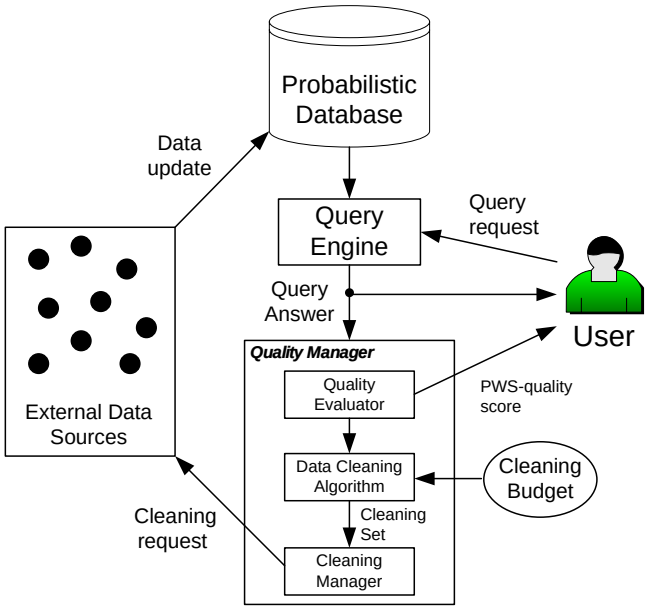


Figure 7.1: The framework of our solution.

The rest of this chapter is organized as follows. In Section 7.2, we discusses the system assumed in this chapter, as well as the data and query models. In Section 7.3 we present the formal notion of the PWS-quality, and efficient methods for evaluating it. Section 7.4 describes the quality-based cleaning method and other heuristics. We present our experimental results in Section 7.5, and conclude this chapter in Section 7.6.

7.2 Data and Query Models

We now describe the probabilistic database model (Section 7.2.1), and the types of queries studied in this chapter (Section 7.2.2).

7.2.1 The Probabilistic Database Model

A probabilistic database D contains m entities known as the "x-tuples" [6, 33, 3]. We denote the k-th x-tuple by τ_k, where $k = 1, \ldots, m$. We also assume that the x-tuples are independent of each other. Each x-tuple is a set of tuples t_i, which represent a distribution of values within the x-tuple. There are a total of n tuples in D. Each tuple has four attributes: $(\mathrm{ID}_i, v_i, e_i, x_i)$. Here ID_i is an unique identifier of t_i, and v_i is a real-valued attribute used in queries (called the *querying attribute*). For simplicity, we assume v_i is one-dimensional, but our solutions can generally be extended to support multi-dimensional attributes. The attribute e_i is the existential probability of t_i – the probability that t_i exists in the real world. Each tuple belongs to one of the x-tuples, and $x_i = \{k|k = 1, \ldots, m\}$ denotes that t_i belongs to the k-th x-tuple.

Within the same x-tuple, the existence of tuples is mutually exclusive. We also assume that the sum s_k of all e_i's of the tuples in the same x-tuple τ_k is equal to 1. If s_k is less than 1, we conceptually augment a "null" tuple to τ_k, whose querying attribute has a value equal to $-\infty$, and existential probability equal to $1 - s_k$. This null tuple is only used for completeness in proofs; they do not exist physically. In Table 7.1, for example, there are four x-tuples (a, b, c, d). The "Price" and "Prob." columns represent the querying attribute and existential probability respectively.

7.2.2 Queries

We study two types of entity-based queries: *non-rank-based* and *rank-based* [23]. In a non-rank-based query, a tuple's qualification probability is independent of the existence of other tuples. For example, queries whose selection clauses involve only the attributes of a single tuple belong to this query class. Another good example is the range query:

Definition 7.1 Probabilistic Range Query (PRQ). *Given a closed interval* $[a, b]$*, where* $a, b \in \Re$ *and* $a \leq b$*, a PRQ returns a set of tuples* (t_i, p_i)*, where* p_i*, the qualification probability of* t_i*, is the non-zero probability that* $v_i \in [a, b]$*.*

For a rank-based query, a tuple's qualification probability is dependent on the existence of other tuples. Examples of this class include the `MAX/MIN` query and the nearest-neighbor query. We study the `MAX` query in this chapter.

Definition 7.2 Probabilistic Maximum Query (PMaxQ). *A PMaxQ returns a set of tuples* (t_i, p_i)*, where* p_i*, the qualification probability of* t_i*, is the non-zero probability that* $v_i \geq v_j$*, where* $j \neq i \wedge j = 1, \ldots, n$*.*

Although the answers returned by both queries have the same form, their PWS-quality scores are computed in a different way, as illustrated in the next section. We will also discuss how PWS-quality can be computed for other entity-based queries (e.g., nearest-neighbor queries). Table 7.5 shows the symbols used in uncertain data cleaning.

Let us now briefly explain how PRQ and PMaxQ can be evaluated efficiently (without consulting the possible worlds). A PRQ can be computed by examining each tuple t_i, and testing whether its querying attribute, v_i, is

Symbol	Description
Data model	
D	A probabilistic database
τ_k	An x-tuple of D, with $k = 1, \ldots, m$
t_i	A tuple of D with $i = 1, \ldots, n$
ID_i	A unique identifier of t_i
v_i	Querying attribute of t_i
e_i	Existential probability of t_i
x_i	The ID of the x-tuple (k) that contains t_i
Query model	
Q	A probabilistic query
p_i	Qualification prob. of t_i for Q
P_k	Qualification prob. of τ_k for Q
Quality Metrics	
r_j	A distinct PW-result ($j = 1, \ldots, d$)
q_j	Prob. of occurrence of r_j
$S(D, Q)$	PWS-quality of Q on database D
Data Cleaning	
C	Cleaning budget for Q
c_k	Cost of cleaning τ_k
X	Set of x-tuples to be cleaned
$I(X, D, Q)$	Quality improvement of cleaning X

Table 7.5: Symbols for cleaning probabilistic databases.

within $[a, b]$. If this is not true, then t_i's qualification probability, p_i, must be zero. Otherwise, $p_i = e_i$, its existential probability. The probability of t_i for satisfying the PMaxQ is the product of (1) its existential probability and (2) the probability that x-tuples other than the one that t_i belongs to do not have a tuple with value larger than v_i. Indexing solutions (e.g., B-tree and R-tree) can be built on the querying attributes in order to improve the query performance.

Also, as mentioned in Section 7.1, a query may need to be re-evaluated after cleaning is done. However, this round of query evaluation can be done more efficiently. In particular, only the x-tuples whose tuples appear in the query answer of the first evaluation round need to be considered. We will explain these details in Section 7.4.5 after we discuss the issues of data

cleaning.

7.3 The PWS-Quality

To understand this metric, let us first review how the possible world semantics (PWS) are used to evaluate a query. As shown in Figure 7.2, a probabilistic database is expanded to a set of *possible worlds* (Step 1). The query is then issued on each possible world, producing answers that we call *PW-results* (Step 2). In Step 3, the PW-results are combined to produce the final query answer. For example, a possible world in Table 7.1 is the set W of tuples whose IDs are a_1, b_2, c_3 and d_1. If a `MAX` query is issued on `price`, the PW-result of W is a_1 (since it has the largest `price`), with a probability of $0.7 \times 0.4 \times 0.2 \times 1 = 0.056$. In the final query answer, the qualification probability of a_1 is equal to the sum of the probabilities that a_1 is the answer in all the possible worlds, that is, 0.35.

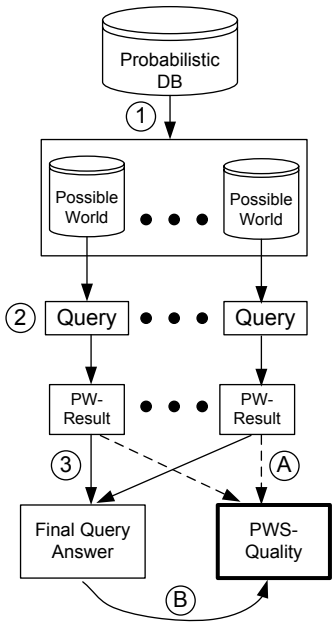


Figure 7.2: PWS and PWS-Quality.

The PWS-quality is essentially a function of the PW-results (computed in Step A in Figure 7.2). Since the form of the queries for computing the PW-results is not specified, the PWS-quality can be applied to *any* type of queries. The major problem of this approach is that there can be an exponentially large number of possible worlds [6, 33], so as the PW-results. Computing PWS-quality can thus be very costly. To address this problem, we show how PWS-quality can be evaluated by using the tuple information in the final query answers, instead of the PW-results (Step B), for PRQ and PMaxQ. Since the number of tuples in the query answer set is much smaller than that of the possible worlds, the computation efficiency of PWS-quality is also better.

We now examine the definition of PWS-quality (i.e., Step A) in Section 7.3.1. Then we propose a better method (i.e., Step B) in Section 7.3.2. Sections 7.3.3 and 7.3.4 outline the proofs of the new method for PRQ and PMaxQ.

7.3.1 Evaluating the PWS-Quality

The PWS-quality is essentially the entropy [73] of the PW-results produced in Step 2 of Figure 7.2. Let $\{r_1, \ldots, r_d\}$ be the set of d distinct PW-results. Also, let q_j be the probability that r_j is the actual answer (we call q_j the *PW-result probability* of r_j).

Definition 7.3 *The* **PWS-quality** *of a query Q evaluated on a database D, denoted by $S(D, Q)$, is:*

$$S(D, Q) = \sum_{j=1}^{d} q_j \log q_j \tag{7.1}$$

Notice that the base of the log() function is 2. Moreover, the sum of the PW-result probabilities must be equal to one (i.e., $\sum_{j=1}^{d} q_j = 1$). By using this fact and comparing with the entropy function [73], it can be shown that the PWS-quality (Equation 7.1) is the negated value of the entropy of the PW-results. The entropy, a popular function for measuring uncertainty in the information theory literature, is adopted here to quantify the impreciseness of query answers. The value of the PWS-quality score ranges from $-\log d$ (i.e., the most ambiguous result) to zero (i.e., a single PW-result).

The problem of computing PWS-quality in this way is that we need to know all the PW-result probabilities. This may represent a performance bottleneck, since the number of PW-result possibilities, derived from possible worlds, can be exponentially large. This is true for a PRQ, where each PW-result contains a unique set of tuples whose querying attributes are inside the query range. For a PMaxQ, the number of PW-results can also be combinatorial, if more than one tuple contain the same querying attribute value. For instance, in Table 7.1, tuples b_1 and c_2 have the same `price` (i.e., \$110). We then have $2^2 - 1 = 3$ PW-results that contain one or more of these tuples as the answer (i.e., $\{b_1\}, \{c_2\}, \{b_1, c_2\}$), derived from the possible worlds where all tuples with `price` above \$110 are excluded. Let us investigate how PWS-quality can be evaluated more efficiently for these queries.

7.3.2 The x-Form of the PWS-Quality

The PWS-quality can in fact be computed by using the probability information of the tuples in the query answer (Step B of Figure 7.2). In particular, the PWS-quality (for both PRQ and PMaxQ) can be converted to an expression known as the *x-form*. An x-form is essentially a sum of some function g evaluated on each x-tuple τ_k, and g can be computed efficiently based on the

probability information of the tuples in τ_k. For notational convenience, we use $Y(x)$ to denote the function $x \log x$. We also let P_k be the qualification probability of τ_k (i.e., its probability for satisfying the query). Since tuples belonging to the x-tuple are mutually exclusive, we have

$$P_k = \sum_{t_i \in \tau_k} p_i \tag{7.2}$$

The following lemma presents an alternative formula of PWS-quality.

Lemma 7.1 *The x-form of the PWS-quality is given by:*

$$S(D, Q) = \sum_{k=1}^{m} g(k, D, Q) \tag{7.3}$$

For **PRQ**,

$$g(k, D, Q) = \sum_{t_i \in \tau_k} p_i \log e_i + Y(1 - P_k) \tag{7.4}$$

For **PMaxQ**, *let the i-th tuple of τ_k be $t_{k,i}$, sorted in descending order of $v_{k,i}$. If $t_{k,i}$ has existential probability $e_{k,i}$ and qualification probability $p_{k,i}$, then,*

$$g(k, D, Q) = \sum_{i=1}^{|\tau_k|} (p_{k,i} \log e_{k,i} + \omega_{k,i} \log(1 - \sum_{j=1}^{i} e_{k,j})) \tag{7.5}$$

where

$$\omega_{k,i} = \begin{cases} (1 - \sum_{j=1}^{i} e_{k,j})(\frac{p_{k,i}}{e_{k,i}} - \frac{p_{k,i+1}}{e_{k,i+1}}) & i < |\tau_k| \\ 0 & i = |\tau_k| \end{cases} \tag{7.6}$$

Lemma 7.1 states that the PWS-quality is the sum of some function g for $k = 1, \ldots, m$. Each g is a function of the existential and qualification probabilities of tuples within x-tuple τ_k. Interestingly, even though PRQ and PMaxQ have different semantics, their PWS-quality function has a common form (i.e., Equation 7.3). Evaluating the x-form of PMaxQ needs some

preprocessing, by sorting the tuples within the same x-tuple according to the querying attributes. An example of tuples sorted in this way is shown in Table 7.1.

Given that the values of $g(k, D, Q)$ are available, the x-form of both queries can be computed by iterating on the whole table of x-tuples, in $O(m)$ times. For PMaxQ, an additional average cost of $O(n \log \frac{n}{m})$ may be needed to sort the tuples. This is still faster than using an exponential number of PW-result probability values to evaluate the PWS-quality. The x-form is also useful to solve the data cleaning problem, to be presented in Section 7.4. We next show a useful fact.

Lemma 7.2 *$g(k, D, Q) < 0$ if and only if there exists $t_i \in \tau_k$ such that $p_i \in (0, 1)$. Otherwise, $g(k, D, Q) = 0$.*

Proof : First of all, $g(k, D, Q)$ is obviously zero if $p_i = 0$, for $\forall t_i \in \tau_k$, which means no tuples of this x-tuple can satisty the query Q. Secondly, the case $p_i = 1$ implies that $e_i = 1$, which means the x-tuple τ_k contains only one tuple and must satisfy the query. Therefore, τ_k is clean, and brings no uncertainty to the query answer. For all other cases, i.e. there exists $t_i \in \tau_k$ such that $p_i \in (0, 1)$, it is not certain whether the x-tuple satisfies the query or not, and $g(k, D, Q)$, the uncertainty caused by τ_k will be non-zero. □

Given an x-tuple τ_k, the above states that $g(k, d, Q)$ is less than zero if there exists a tuple $t_i \in \tau_k$, such that its qualification probability, p_i, is neither zero nor one. More importantly, an x-tuple whose tuples' qualification probabilities are either zero or one *does not* need to be included in computing the PWS-quality (Equation 7.3). Thus, we do not need to examine the whole database. Instead, we can just pick the x-tuples that satisfy the conditions stated in Lemma 7.2. This is exactly the set of x-tuples whose tuples in the

final query answer have qualification probabilities not equal to one. If we are given the query answer (which are produced by the query engine), then the set of x-tuples required to compute the PWS-quality can be derived easily.

We remark that the x-forms for PRQ and PMaxQ can also be used by other entity-based queries. Particularly, the x-form of PRQ can be used by other non-rank-based queries, whose selection conditions only involve the attributes of a single tuple. The x-form of the PMaxQ can also be used by `MIN` and nearest-neighbor queries, by using a different sorting criterion on the query attribute. Next, we explain briefly how the x-form is obtained for PRQ and PMaxQ.

7.3.3 Deriving the x-Form for PRQ

We now show the proof of the x-form expression for PRQ. Here, a distinct PW-result r_j is essentially a set of tuples t_i's that satisfy the PRQ (i.e., $v_i \in [a, b]$) in one or more possible worlds. Note that r_j cannot possess more than one tuple from the same x-tuple, since each possible world only contains one of the tuples selected from each x-tuple. The probability q_j of getting r_j is then equal to:

$$q_j = \prod_{t_i \in r_j} e_i \prod_{\tau_k \cap r_j = \emptyset} (1 - P_k) \tag{7.7}$$

Equation 7.7 is the product of: (1) the probability all tuples t_i's that satisfy the PRQ are in r_j (i.e., $\prod_{t_i \in r_j} e_i$), and (2) the probability that other x-tuples satisfy the PRQ but do not appear in r_j(i.e., $\prod_{\tau_k \cap r_j = \emptyset}(1 - P_k)$). We then substitute this into Equation 7.1, which becomes:

$$S(D, Q) = \sum_{j=1}^{d} q_j \Big(\sum_{i=1 \wedge t_i \in r_j}^{n} \log e_i + \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{m} \log(1 - P_k) \Big) \tag{7.8}$$

Now, we claim that

$$\sum_{j=1}^{d} \sum_{i=1 \wedge t_i \in r_j}^{n} q_j \log e_i = \sum_{i=1}^{n} p_i \log e_i \tag{7.9}$$

To understand why, notice that the summation orders of the left side of Equation 7.9 can be reversed, which becomes:

$$\sum_{i=1}^{n} \sum_{j=1 \wedge t_i \in r_j}^{d} q_j \log e_i \tag{7.10}$$

Observe that the qualification probability p_i (of tuple t_i) can be obtained by summing up the distinct PW-result probabilities (q_j's), where result r_j contains t_i. That is,

$$p_i = \sum_{j=1 \wedge t_i \in r_j}^{d} q_j \tag{7.11}$$

This is an important equation because it allows us to replace all q_j's with p_i's in the expression. Since there are at most m p_i's in the final query answer, the PWS-quality can be computed faster than using q_j's, the number of which is exponential. Thus, Equation 7.9 is correct.

Next, we prove that

$$\sum_{j=1}^{d} \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{m} q_j \log(1 - P_k) = \sum_{k=1}^{m} Y(1 - P_k) \tag{7.12}$$

Again, by reversing the order of summation, the left side of 7.12 becomes

$$\sum_{k=1}^{m} \sum_{j=1 \wedge \tau_k \cap r_j = \emptyset}^{d} q_j \log(1 - P_k) \tag{7.13}$$

Notice that $\sum_{j=1 \wedge \tau_k \cap r_j = \emptyset}^{d} q_j$ is the probability that τ_k does not appear in any

of the distinct results, which is equal to $1 - P_k$. Thus, Equation 7.12 can be obtained. Finally, by using Equations 7.8,7.9 and 7.12, the x-form of PRQ (Equation 7.3 and Equation 7.4) is proved.

Notice that our proof still holds for other queries that involve different selection constraints, by replacing the condition that a tuple is included in a possible result (i.e., "$v_i \in [a, b]$") with the required conditions (e.g., "$v_i < a \vee v_i > b$"). That is, the x-form of PRQ can be generalized to a non-rank-based query, which tests whether a tuple satisfies it based on the tuple's own attributes.

7.3.4 Deriving the x-Form for PMaxQ

The derivation of the x-form for PMaxQ is similar to that of PRQ, with the following major differences: (1) all tuples in an x-tuple are assumed to be sorted in descending order, and (2) the PW-result probability q_j has a different formula. Our solution also handles the scenario where more than one tuple with the same querying attribute value exist.

For convenience, let the i-th tuple of τ_k be $t_{k,i}$, sorted in descending order of $v_{k,i}$. A distinct PW-result r_j for PMaxQ is then a set of tuples t_i's that have the same maximum value, in one or more possible worlds. Suppose $r_j.v$ is the value shared by tuples in result r_j. The probability q_j of getting r_j is then equal to:

$$q_j = \prod_{t_i \in r_j} e_i \prod_{\tau_k \cap r_j = \emptyset} Pr(\tau_k < r_j.v) \tag{7.14}$$

where $Pr(\tau_k < r_j.v)$ is the probability that τ_k has a tuple with querying attribute value smaller than $r_j.v$. Equation 7.14 is the product of: (1) the probability that all tuples t_i's in r_j exist (i.e., $\prod_{t_i \in r_j} e_i$), and (2) the probability that all other x-tuples have at least a tuple with a value smaller than

$r_j.v$ (i.e., $\prod_{\tau_k \cap r_j = \emptyset} Pr(\tau_k < r_j.v)$). Moreover, since all tuples of an x-tuple are sorted in descending order, we can rewrite $Pr(\tau_k < r_j.v)$ as:

$$Pr(\tau_k < r_j.v) = 1 - \sum_{l=1}^{s(j,k)} e_{k,l} \tag{7.15}$$

where $s(j,k)$ is some integer inside $[1, |\tau_k|]$, such that $v_{k,s(j,k)}$ is the smallest value not smaller than $r_j.v$.

By substituting Equation 7.14 into $\log q_i$, Equation 7.1 becomes:

$$S(D,Q) = \sum_{j=1}^{d} q_j (\sum_{i=1 \wedge t_i \in r_j}^{n} \log e_i + \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{m} \log(Pr(\tau_k < r_j.v))) \tag{7.16}$$

Similarly as the proof in Section 7.3.3, we can have

$$\sum_{j=1}^{d} \sum_{i=1 \wedge t_i \in r_j}^{n} q_j \log e_i = \sum_{i=1}^{n} p_i \log e_i \tag{7.17}$$

Next we prove that

$$\begin{aligned} &\sum_{j=1}^{d} \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{m} q_j \log(Pr(\tau_k < r_j.v)) \\ &\quad = \sum_{k=1}^{m} \sum_{i=1}^{|\tau_k|} \omega_{k,i} \log(1 - \sum_{j=1}^{i} e_{k,j}) \end{aligned} \tag{7.18}$$

By substituting Equation 7.15 into $\log Pr(\tau_k < r_j.v)$, the left part of Equation 7.18 becomes:

$$\sum_{j=1}^{d} \sum_{k=1 \wedge \tau_k \cap r_j = \emptyset}^{m} q_j \log(1 - \sum_{l=1}^{s(j,k)} e_{k,l}) \tag{7.19}$$

By swapping the summation orders and noticing that $s(j,k)$ is just some number in $[1, |\tau_k|]$, Equation 7.19 can be simplified to the form of Equations 7.3 and 7.5.

The rest is to show that $\omega_{k,i}$ is equivalent to Equation 7.6. First, notice that $\omega_{k,i}$ is just a sum of q_j's (for PW-results r_j). In particular,

$$\omega_{k,i} = \sum_{v_{k,i+1} < r_j.v \leq v_{k,i}} q_j \tag{7.20}$$

This is because the value i in the term $\log(1 - \sum_{j=1}^{i} e_{k,j})$ of Equation 7.5 represents the i-th position of the tuple in the x-tuple τ_k (i.e., $t_{k,i}$) which has the smallest querying attribute larger than $r_j.v$, which must be between $(v_{k,i+1}, v_{k,i}]$, or else i cannot be the position where $r_j.v$ is just larger than $v_{k,i+1}$. Notice that if $i = |\tau_k|$, no tuples $t_{h,l}$ can satisfy this condition, and so $\omega_{k,|\tau_k|} = 0$.

Next, we claim that

$$\sum_{r_j.v \leq v_{k,i}} q_j = (1 - \sum_{l=1}^{i} e_{k,l}) \cdot \frac{p_{k,i}}{e_{k,i}} \tag{7.21}$$

To prove Equation 7.21, note that its left side is the probability that the answer to PMaxQ has a querying attribute value smaller than $v_{k,i}$. This is equal to the product of the occurrence probabilities of two events: E_1, the event that all tuples that do not belong to τ_k and whose values are larger than $v_{k,i}$ do not exist; E_2, the event that none of the tuples $\{t_{k,1}, \ldots, t_{k,i}\}$ exist. The probability that E_1 is true ($Pr(E_1)$) can be derived by using the fact that $p_{k,i} = Pr(E_1) \cdot e_{k,i}$. Thus, $Pr(E_1) = \frac{p_{k,i}}{e_{k,i}}$. Since $Pr(E_2) = 1 - \sum_{j=1}^{i} e_{k,j}$, Equation 7.21 can be obtained.

Finally, Equation 7.20 can be written as

$$\sum_{v_{h,l} \leq v_{k,i}} p_{h,l} - p_{k,i+1} - \sum_{v_{h,l} \leq v_{k,i+1}} p_{h,l} \tag{7.22}$$

By substituting Equation 7.22 with the result of Equation 7.21, we prove that Equation 7.6 is correct.

We can easily adapt the x-form of PMaxQ to other rank-based queries. For example, for `MIN` queries, we can sort the tuples within an x-tuple in ascending order, and change the comparison signs accordingly. As another example, the x-form for the nearest-neighbor query can be derived by ordering the tuples according to the Euclidean distance of their querying attributes from the query point.

7.4 Cleaning Uncertain Data

Let us now discuss how the PWS-quality can be used to facilitate the cleaning of uncertain data. Section 7.4.1 presents the formal definition of this problem. In Sections 7.4.2 and 7.4.3, we describe an efficient solution that can be applied to the queries under study. Several heuristics that provide efficient solutions are presented in Section 7.4.4. We also investigate how to efficiently reevaluate a query on the cleaned database in Section 7.4.5.

7.4.1 Problem Definition

Recall that our goal is to select the most appropriate set of x-tuples to be cleaned, under a stringent budget, in order to achieve the highest expected quality improvement. Formally, let us define an operation called *clean*(τ_k):

Definition 7.4 *Given an x-tuple τ_k, clean(τ_k) replaces τ_k with an x-tuple that contains a single tuple: $\{ID_i, v_i, 1, k\}$, such that ID_i and v_i are the cor-*

responding identifier and querying attribute value of some tuple t_i that belongs to τ_k.

Essentially, τ_k becomes "certain" after *clean*(τ_k) is performed. Only one of the tuples in the original x-tuple is retained, with existential probability changed to one. The value of the new tuple depends on the cleaning operation. In Table 7.1, for example, after *clean*(a) is performed, a contains a single tuple $\{a_2, 80, 1, a\}$, derived from a_2, with a `price` of \$80 and existential probability of 1.

Cleaning an x-tuple may involve a cost. For example, if an x-tuple represents a sensor reading in a sensor monitoring application, then the cost of cleaning this x-tuple (by probing the sensor to get the latest value) can be the amount of battery power required for that sensor's value to be shipped to the base station. We use c_k, a natural number, to capture the cost of performing *clean*(τ_k). We also assume that a query Q is associated with a *budget* of C units, where C is a natural number. This value limits the maximum amount of cleaning effort that can be used to improve the quality of Q. In sensor monitoring, C can be the total amount of energy allowed for probing the sensors. The value of C may be based on the amount of system resource available, or the priority of the query user.

Our goal is to obtain the set of x-tuples that, under a given budget, yields the most significant expected improvement in PWS-quality. This set of x-tuples is then selected to be cleaned. Specifically, let X be any set of x-tuples chosen from database D. Without loss of generality, let $X = \{\tau_1, \ldots, \tau_{|X|}\}$. Also, let $\vec{t}$ be a "tuple vector" of $|X|$ dimensions, where the k-th dimension of $\vec{t}$ is a tuple that belongs to the k-th x-tuple of X. For example, if $X = \{\tau_1, \tau_2\}$, where $\tau_1 = \{t_0, t_3\}$ and $\tau_2 = \{t_2, t_5\}$, then two possible values of $\vec{t}$ are $\{t_0, t_5\}$ and $\{t_3, t_2\}$.

Now, let $D'(\vec{t})$ be the new database obtained, after $clean(\tau_k)$ is performed on each x-tuple τ_k in X, which produces tuples described in $\vec{t}$. The expected quality of cleaning a set X of x-tuples is then equal to:

$$E(S(D'(\vec{t}), Q)) = \sum_{\vec{t} \in \tau_1 \times \ldots \times \tau_{|X|}} \Pi_{t_i \in \vec{t}} e_i \cdot S(D'(\vec{t}), Q) \tag{7.23}$$

For every tuple vector in $\tau_1 \times \ldots \times \tau_{|X|}$, Equation 7.23 calculates the probability that the new database $D'(\vec{t})$ is obtained (i.e., $\Pi_{t_i \in \vec{t}} e_i$) and the PWS-quality score of query Q evaluated on $D'(\vec{t})$ (i.e., $S(D'(\vec{t}), Q)$).

Definition 7.5 *The quality improvement of cleaning a set X of x-tuples is*

$$I(X, D, Q) = E(S(D'(\vec{t}), Q)) - S(D, Q) \tag{7.24}$$

Our problem can now be formulated as follows:

Definition 7.6 The Data Cleaning Problem. *Given a budget of C units, choose a set X of x-tuples from D such that $I(X, D, Q)$ attains the highest value.*

A straightforward way of solving this problem is to obtain the powerset of all x-tuples in D. For each element (a set X of x-tuples) of the powerset, we test whether the total cost of cleaning the x-tuples in X exceeds the budget C. Among those that do not, we select the set of x-tuples whose quality improvement is the highest.

This solution is inefficient for two reasons. First, given a set X of x-tuples, computing Equation 7.24 requires the consideration of all tuple vectors of X, which are the combinations of tuples selected from the x-tuples in X. Second, the number of sets of x-tuples to be examined is exponential. We

tackle the first problem in Section 7.4.2. The second problem is addressed in Section 7.4.3.

7.4.2 Evaluating Quality Improvement

Equation 7.24 can be computed more easily by using the x-form of PWS-quality, as shown by the following lemma.

Lemma 7.3 *The quality improvement of cleaning a set X of x-tuples is:*

$$I(X, D, Q) = -\sum_{k=1}^{|X|} g(k, D, Q) \tag{7.25}$$

where $g(k, D, Q)$ is given by Equations 7.4 and 7.5, for PRQ and PMaxQ respectively.

Proof : By using the x-form (Equation 7.3), we can rewrite $E(S(D'(\vec{t}), Q))$ as

$$\sum_{k=1}^{|X|} E(g(k, D'(\vec{t}), Q)) + \sum_{k=|X|+1}^{m} E(g(k, D'(\vec{t}), Q)) \tag{7.26}$$

For both PRQ and PMaxQ, we claim that:

$$g(k, D'(\vec{t}), Q)) = 0, \textit{for } k = 1, \ldots, |X| \tag{7.27}$$

$$E(g(k, D'(\vec{t}), Q)) = g(k, D, Q), \textit{for } k = |X| + 1, \ldots, m \tag{7.28}$$

By using Equations 7.27 and 7.28, Equation 7.26 becomes $\sum_{k=|X|+1}^{m} g(k, D, Q)$. Together with Equations 7.24 and 7.3, we can see that Equation 7.25 is correct. The following sketches the proof of Equations 7.27 and 7.28 for PRQ and PMaxQ.

PRQ: First, notice that the new database $D'(t)$ contains a single tuple for every $\tau_k \in X$, whose existential probability is 1, and qualification probability

is either 0 or 1. Using this fact and Equation 7.4, we can see Equation 7.27 is true for every $k \in [1, |X|]$. For Equation 7.28, observe that $g(k, D'(\vec{t}), Q)$ is just some function (Equation 7.4) of p_i's and e_i's for $t_i \in \tau_k$, where $\tau_k \notin X$. As discussed in Section 7.2.2, the value of p_i for PRQ is either e_i or zero. Since these values of p_i's and e_i's are not changed by any cleaning operations on the x-tuples in X, Equation 7.28 holds for $k = |X| + 1, \ldots, m$.

PMaxQ: Let the existential and qualification probabilities of each tuple $t_{i,k}$ for the new database $D'(\vec{t})$ be $e'_{k,i}(\vec{t})$ and $p'_{k,i}(\vec{t})$ respectively. Then, After $clean(\tau_k)$, only one tuple ($t_{k,1}$) in τ_k can exist in $D'(\vec{t})$. Since $\omega_{k,1} = 0$ (Equation 7.5), we have $g(k, D'(\vec{t}), Q) = p'_{k,1}(\vec{t}) \log e'_{k,1}(\vec{t})$. Moreover, $e'_{k,1}(\vec{t}) = 1$. Thus, $g(k, D'(\vec{t}), Q) = 0$ and the proof for Equation 7.27 is complete.

To prove Equation 7.28, note that by using Equation 7.5, $E(g(k, D'(\vec{t}), Q))$ can be written as:

$$\sum_{i=1}^{|\tau_k|} (E(p'_{k,i}(\vec{t})) \log e_{k,i} + (\frac{E(p'_{k,i}(\vec{t}))}{e_{k,i}} - \frac{E(p'_{k,i+1}(\vec{t}))}{e_{k,i+1}}) Y (1 - \sum_{j=1}^{i} e_{k,j})) \quad (7.29)$$

Next, we claim that

$$E(p'_{k,i}(\vec{t})) = p_{k,i}, \forall k > |X| \quad (7.30)$$

In order to compute $E(p'_{k,i}(\vec{t}))$ directly, $p'_{k,i}(\vec{t})$ needs to be evaluated for every vector $\vec{t} \in \tau_1 \times \ldots \times \tau_{|X|}$. Furthermore, computing each $p'_{k,i}(\vec{t})$ involves querying on every possible world in $D'(\vec{t})$. Thus, $E(p'_{k,i}(\vec{t}))$ is just some function of all the PW-result probabilities queried on D, and this is the same function for $p_{k,i}$. Thus, Equation 7.30 is correct. Finally, by substituting Equation 7.30 into Equation 7.29, we obtain Equation 7.28.

Equation 7.25 reveals three important facts. First, the quality improvement, $I(X, D, Q)$, is non-negative (since $g(k, D, Q)$ is non-positive). This implies that the expected quality monotonically increases with the perfor-

mance of the *clean(τ_k)* operation. Second, the task of computing $I(X, D, Q)$ is made easier (compared with Equation 7.24), since $g(k, D, Q)$ can be computed in polynomial time. If these g values have been stored (e.g., in a lookup table) during the process of computing the x-form of the PWS-quality (Equation 7.3), then $I(X, D, Q)$ can be evaluated by a table lookup. Third, Equation 7.25 can be applied to both PRQ and PMaxQ, since $g(k, D, Q)$ have been derived for both queries in Section 7.3.2. Let us see how these results can be used to develop an efficient data cleaning algorithm.

7.4.3 An Optimal and Efficient Data Cleaning Algorithm

We now address the second question: to find out the set B of x-tuples that leads to the optimal expected quality improvement in PWS-quality, is it possible to avoid enumerating all the combinations of x-tuples in the whole database? To answer this, we first state the following lemma:

Lemma 7.4 *For any x-tuple $\tau_k \in B$, τ_k must satisfy the condition: there exists $t_i \in \tau_k$ such that (t_i, p_i) appears in the final answer of Q, with $p_i \in (0, 1)$.*

Proof : Consider an x-tuple τ_j, whose tuples' qualification probabilities are either zero or one. We can show that τ_j does not need to be included in B. Suppose by contradiction that $\tau_j \in B$. According to Lemma 7.2, $g(j, d, Q) = 0$. By using Lemma 7.3, we can see that including τ_j in B has no effect on the quality improvement i.e., $I(B, D, Q)$. Thus, it is unnecessary to include τ_j in B.

In fact, by excluding τ_j, the remaining x-tuples that we need to consider for cleaning are those that contain at least a tuple t_i with the following conditions: (1) t_i appears in the final query answer, and (2) $p_i \in (0, 1)$.

For example, for the `MAX` query evaluated on Table 7.1, the optimal set B can be derived from the result of the `MAX` query (Table 7.2), which contains the tuples from x-tuples a,b, and c, but not d. Correspondingly, B is the subset of the x-tuples $\{a, b, c\}$. Thus, Lemma 7.4 reduces the search space to the x-tuples whose tuples appear in the query answer. It also means that the input of our data cleaning algorithm can be the tuples contained in the query answer (c.f. Figure 7.1).

We now focus on the x-tuples that satisfy the conditions of Lemma 7.4. Let Z be the number of these x-tuples. We use τ_k (where $k = 1, ..., Z$) to denote these x-tuples.

An Optimization Problem. We now present an efficient algorithm that provides an optimal solution to the data cleaning problem. This algorithm can be applied to entity-based queries, including PRQ and PMaxQ. We assume the values of $g(k, D, Q)$ have been obtained for all values of $k = 1, \ldots, Z$. For notational convenience, we also use g_k to represent $g(k, D, Q)$ (since D and Q are constant parameters). Then, Definition 7.6 can be reformulated as an optimization problem $P(C, M)$, where $M = \{\tau_1, \ldots, \tau_Z\}$ is the set of candidates to be considered, and C is the budget assigned to the query:

Maximize

$$-\sum_{k=1}^{Z} b_k \cdot g_k \tag{7.31}$$

Subject to

$$\sum_{k=1}^{Z} b_k \cdot c_k \leq C \tag{7.32}$$

Here $\{b_k | k = 1, \ldots, Z, b_k = 0|1)$ is a bit vector of length Z, encoding the IDs of x-tuple(s) chosen from M to be cleaned. Particularly, $b_k = 1$

if x-tuple τ_k is selected, and $b_k = 0$ otherwise. Equation 7.31 is the total quality improvement for cleaning a set of x-tuples (where τ_k is chosen if $b_k = 1$), which is the same as Equation 7.25. The optimization constraint is described in Equation 7.32, which requires that the total cost of cleaning the set of x-tuples cannot be more than C. Note that since Equation 7.31 (or Equation 7.25) is true for PRQ and PMaxQ, the solution to this problem can be applied to both queries.

We further note that $P(C, M)$ is essentially a variant of the 0/1 knapsack problem [29], which can be solved by using dynamic programming techniques, e.g. Algorithm 5. Therefore we skip the details of the solution. The time and space complexities of the optimal solution are respectively $O(CZ)$ and $O(CZ^2)$.

7.4.4 Heuristics for Data Cleaning

To further improve the efficiency of data cleaning, we have developed three other heuristics:

1. Random: This is the simplest heuristic, where x-tuples are selected randomly until the query budget is exhausted.

2. MaxQP: Compute the qualification probability P_k for each x-tuple τ_k, using Equation 7.2. Then, choose the x-tuples in descending order of P_k (where $P_k \neq 1$) until the total cost exceeds C. The rationale is that selecting x-tuples with higher qualification probabilities may have a better effect on the PWS-quality than those with small values.

3. Greedy: Let $f_k = \frac{g_k}{c_k}$. Select x-tuples with the highest values of f_k such that the maximum total cost is less than C. Intuitively, f_k is the quality improvement of $clean(\tau_k)$ per unit cost. The choice of x-tuples is decided by the amount of quality improved and the cost required.

The MaxQP and Greedy heuristics can be extended to support large query answer sets. Specifically, if the number of x-tuples to be considered by the data cleaning algorithm is too large to be stored in the main memory, disk-based algorithms (e.g., [66]) can be used to sort the x-tuples. Then, the x-tuples that rank the highest can be retrieved. In our experiments, since the main memory is large enough to hold the tuples returned to a user, our data cleaning algorithms are executed on the main memory.

7.4.5 Incremental Query Processing

As a sidenote, consider a query Q which has just been evaluated on the database D. After the database is cleaned, the re-running of Q does not require the examination of the whole database. This is because only the tuples that appear in the query result of the pre-cleaned database need to be handled. In particular, Let D' be the new database after cleaning is done, and $R(D)$ be the set of tuples appearing in the answer of Q on D. We claim that the tuples that are included in the answer of Q being evaluated on D', i.e., $R(D')$, must be a subset of $R(D)$. We term this *incremental query processing*, since the evaluation of Q on D' can be based upon the answer set $R(D)$, instead of the whole database D'.

To see this, let u be a tuple which does not appear in $R(D)$. Then, u must not satisfy Q in any possible worlds generated from D. Now, consider a possible world W that appears in D. After cleaning (Definition 7.4) is completed, two cases can occur: (1) W consists of the same set of tuples; or (2) W is eliminated, where one or more tuples that belong to W are removed due to cleaning. Hence, the set of possible worlds of D' must be a subset of those in D. This implies u still cannot satisfy Q in any possible world of D'. Consequently, u must not be in $R(D')$. Hence, in the second round of query

evaluation, we only need to examine the tuples that appear in $R(D)$, instead of D'. This can reduce the effort of evaluating a query on D' significantly.

Next, let us investigate how PRQ and PMaxQ on D' make use of the above observation.

PRQ: We only need to re-evaluate the probabilities of the tuples belonging to the x-tuples in the cleaning set, since these x-tuples are chosen from the answer set $R(D)$ (according to Lemma 7.4). Qualification probabilities of tuples that do not appear in the cleaning set remain unchanged.

PMaxQ: Let t_m be the tuple that consists of the maximum attribute value among all the cleaned tuples. We claim that only tuples whose values larger than or equal to v_m need to be considered. Note that after cleaning, the existential probability of t_m becomes one. Thus all tuples with attribute values smaller than v_m, which has no chance to be the maximum object, can be removed from $R(D)$. The remaining tuples are inserted to $R(D)$, with qualification probabilities recomputed.

7.5 Results

We now discuss the experimental results. Section 7.5.1 describes the settings of our experiments. We present our findings in Section 7.5.2.

7.5.1 Experimental Setup

We have used a synthetic dataset in our experiments. This dataset contains $10K$ objects (e.g., products), each of which has a 1D attribute y (as a price collected automatically from web pages), in the domain $[0, 10,000]$. The value of y follows *attribute uncertainty* described in [87, 23], where y has two components: "uncertainty interval" $y.L$ and "uncertainty pdf" $y.U$. The center of $y.L$ is uniformly distributed in the domain, and the range of $y.L$ is

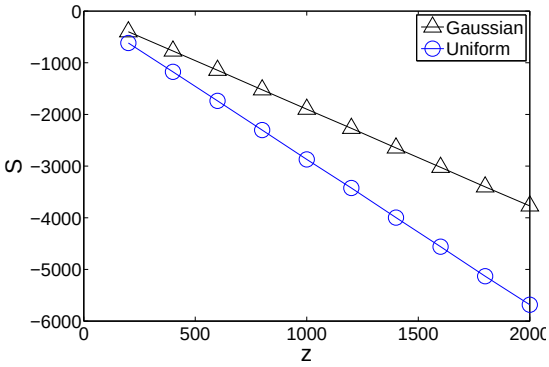


Figure 7.3: Quality vs. z.

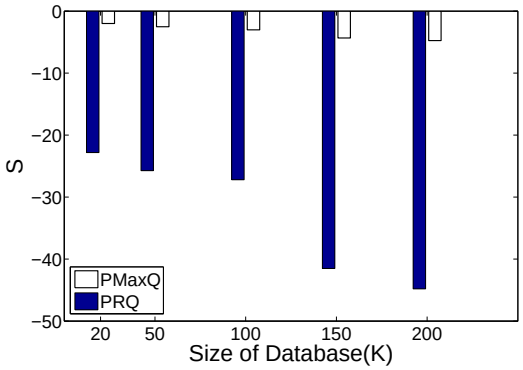


Figure 7.4: Quality vs. Database Size.

uniformly distributed in the range of $[60, 100]$. The uncertainty pdf $y.U$ is a Gaussian distribution defined on $y.L$, with the mean equal to the center of $y.L$, and the variance of 100 units. To store these objects in a probabilistic database, we discretize $y.U$ by obtaining its histogram representation, where the probabilities of 10 equal "histogram bars" within $y.L$ are computed. Each object is then treated as an x-tuple, under which 10 tuples are created, whose querying attributes are the mean values of the histogram bars, and the existential probabilities are the probabilities computed for the histogram bars. Our synthetic database thus has $10K$ x-tuples, or $100K$ tuples.

We also perform experiments on a real dataset [64], which contains some uncertainty in the viewers' ratings for specific movies. The table has $4,999$ x-tuples, or $10,037$ tuples. It has five attributes: `<movie-id,customer-id,date,rate, confidence>`, where `<movie-id,customer-id>` is the key of the x-tuple, and `confidence` records the existential probability of a tuple.

To model the data cleaning problem, for both datasets we attach a "cleaning cost" attribute to each x-tuple. This cost is an integer, uniformly distributed in the range of $[1, 10]$. The query budget has a default value of 30 units.

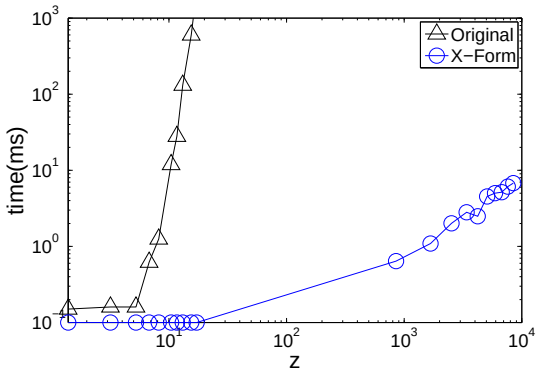


Figure 7.5: The x-Form (PRQ).

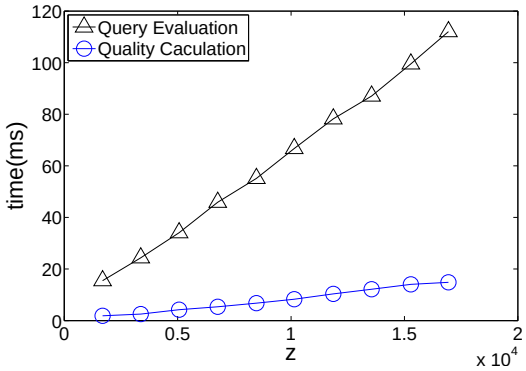


Figure 7.6: Query vs. Quality Evaluation Time.

We have implemented both PRQ and PMaxQ for the synthetic dataset. For PRQ, the query range has a width of 20 units, and its position is evenly distributed in the domain. For the real dataset, the PRQs use `<date,rate>` as a $2D$ querying attribute. We also implement a probabilistic nearest-neighbor query (PNNQ) with a random 4D query point q on the dimensions `<movie-id,customer-id,date,rate>`. Notice that a PNNQ is essentially a PMaxQ by ranking on the Euclidean distance of each data point from q. Thus our algorithms can also be used by a PNNQ.

We have used an R-tree based on the codes in [63] to index the querying attributes, in order to improve the efficiency of computing the query answers. The two primary metrics used for evaluation are: (1) PWS-quality score; and (2) quality evaluation time. Notice that metric (2) does not include the time required for evaluating the query answer.

Each data point is the average of 100 queries. Unless stated otherwise, the results are based on the synthetic dataset. Our codes, implemented in J2SE 1.5.0_09, are run on a PC with an Intel T2400 CPU of 1.83GHz and 1GB memory.

7.5.2 Results

We now present the results. First of all, we discuss the performance of PWS-quality in quantifying uncertainty. We then examine the x-forms of PWS-quality, and present the results on data cleaning. Finally, we report the results on the real dataset.

1. Expressiveness of PWS-Quality

We first investigate how well PWS-quality can quantify the ambiguity of query results. We use z to denote the number of distinct tuples in the query answer, whose qualification probabilities are non-zero. Figure 7.3 shows the quality score of PRQ (S) under a wide range of z. We see that the quality score decreases (i.e., a degradation in quality) when z increases. Intuitively, the larger the value of z, the more tuples are in the query answers, implying a more uncertain answer. Thus, the PWS-quality naturally reflects the vagueness in a query answer.

In the same graph, we present the PWS-quality for two different uncertainty pdfs ($y.U$) of the attribute y in our dataset. As we can see, the uniform pdf generally demonstrates a lower quality score than its Gaussian counterpart. This is not surprising, since a uniform pdf has a larger entropy (i.e., more uncertain) than a Gaussian pdf. Consequently, the query answer becomes more ambiguous, as illustrated by the lower quality scores.

Next, we compare the quality scores of PRQ and PMaxQ in five different databases. For fairness, we compare the scores only for the queries that produce the same number of tuples (with a 1% difference) in their answers in the same database. As shown in Figure 7.4, PRQ scores lower than PMaxQ across all the database samples. The reason is that the PWS-quality is an entropy function of PW-result probabilities (Equation 7.1). On average, the PRQ (which finds tuple(s) with querying attribute(s) in a specified range)

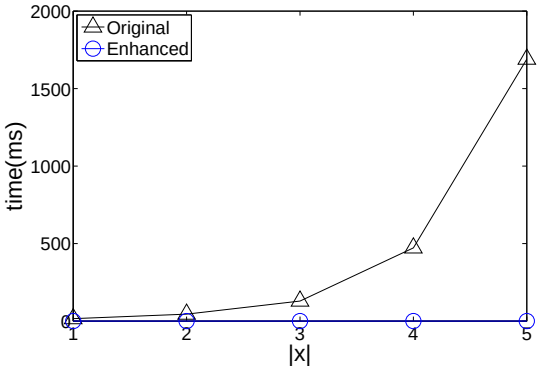


Figure 7.7: Evaluation Time of Quality Improvement.

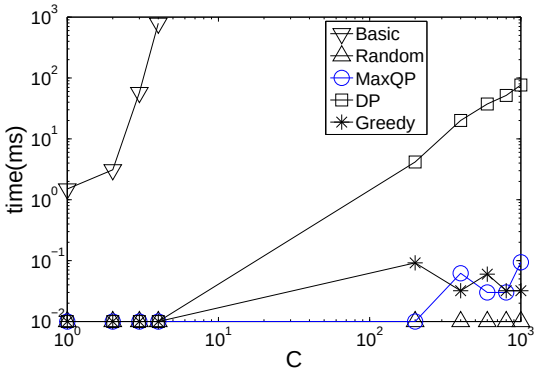


Figure 7.8: Time for selecting x-tuples (PMaxQ).

yields more PW-results than PMaxQ (which finds tuple(s) that gives the maximum value). Hence, the answer of PRQ is also more uncertain than PMaxQ, as shown by our results.

2. Evaluation of PWS-Quality

Sanity Check. We first verify the correctness of the x-form by running several experiments. We found that the relative difference between the x-form and the original definition of PWS-quality (Equation 7.1) is in the order of 10^{-4} or less. For PRQ, at $z = 3.18$, the relative difference is $2.08e-6$. For PMaxQ, that difference is $3.99e-6$ at $z = 77.46$. The slight difference is due to the precision loss at computing small probability values.

Evaluation Time. Figure 7.5 compares the time required for calculating the x-form and the original definition of PWS-quality for PRQ. The amount of time for both methods increases with z, since more result probabilities have to be considered. However, the x-form needs much shorter time to evaluate than the original definition. This follows from the fact that the x-form (Equation 7.4) runs in polynomial time, whereas the original definition (Equation 7.1) requires an exponential time complexity.

Figure 7.6 shows the time required to compute the x-form for PRQ, and

the query evaluation time. We notice that the former needs no more than 10% of the time required by the latter. The quality evaluation time for PMaxQ, not shown here, requires an average of 0.16 ms, or 1.6% of the query evaluation time. The difference in the evaluation time of a query and its quality may actually be larger, since in these experiments we have constructed indexes to speed up the query evaluation. Thus computing PWS-quality adds little overhead to the query evaluation process.

Effect of duplicate tuples. We also study the effect of "duplicate tuples" on computing the PWS-quality of PMaxQ. These are the tuples whose querying attribute values are the same. We modify the synthetic database by treating attribute values within a range of ± 1 as a single value. As a result, each querying attribute value is associated with an average of 6.33 tuples. We found that computing PWS-quality with the original definition takes 259.58 ms to complete, while x-form can finish the job in 3.48 ms. The huge difference (98.6% improvement) is due to the fact that the original definition has to consider a large number of PW-results due to the duplicate tuples, but the x-form only needs to iterate over the x-tuples in the answer once. We also test with other databases; since the results are similar, they are not reported here.

3. Data Cleaning

Next, we examine the results for data cleaning. We assume that the quality of the queries being tested has been obtained. Moreover, prior to cleaning, all the values of $g(D, k, Q)$, computed during the evaluation of the x-forms, have been stored in a lookup table. We first compare the performance of the "enhanced" method in calculating quality improvement (Equation 7.25), with its "original" definition (i.e., Definition 7.5). Figure 7.7 shows the results for both methods on a given set X of x-tuples, with $|X| = 1, \ldots, 5$.

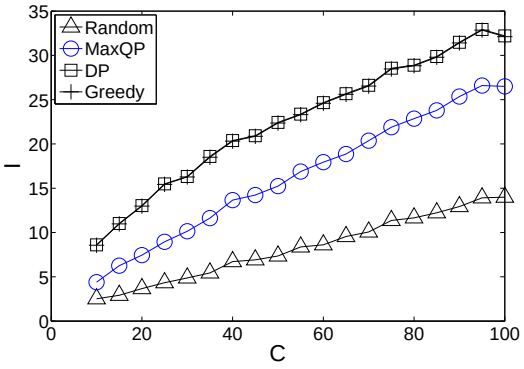


Figure 7.9: I vs. C (PRQ).

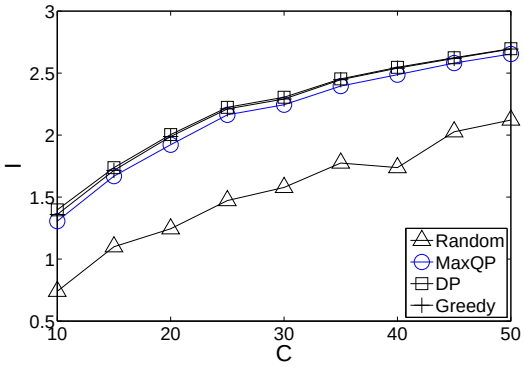


Figure 7.10: I vs. C (PMaxQ).

The time required by the original definition increases sharply with $|X|$. The enhanced method just needs to sum up the $g(D, k, Q)$ values for the x-tuples $\tau_k \in X$, and these values can be retrieved from the lookup table. Thus, its execution time is much less. Thus, the enhanced method will be used in our subsequent experiments.

We then study the time required to compute the quality improvement, using the methods presented in Sections 7.4.3 and 7.4.4. Here, the *Basic* method means the quality improvement of each member of the powerset of all x-tuples is examined, and then the set of x-tuples that yields the highest improvement is chosen. Figure 7.8 examines the amount of time (in log scale) for different methods under a wide range of query budgets used by the PMaxQ. We see that *Basic* performs the worst. The *DP* method provides an optimal solution in polynomial time, and so it is faster than *Basic*. However, its time is higher than other heuristics (i.e., *Random*, *MaxQP* and *Greedy*). The results for PRQ are similar and so they are skipped here.

Figures 7.9 and 7.10 examine the quality improvement (I) for PRQ and PMaxQ respectively. Since both *Basic* and *DP* give the optimal solution, for clarity we only show the result for *DP*. Although *DP* performs the best, it is worth notice that both *Greedy* and *MaxQP* come close to it. This is because

Greedy selects the x-tuple according to the cost and the benefit of cleaning it, while *MaxQP* gives priority to x-tuples with higher qualification probabilities. These factors are important to deciding the optimal solution. Moreover, the data cleaning problem is a variant of the knapsack problem [29], and it has been shown in [35] that the average performance of a greedy solution is close to the optimal one. *Random* does not consider any of these factors at all, and thus it performs the worst. Observe that *MaxQP* is better in PMaxQ than in PRQ. In PMaxQ, by considering a x-tuple with the highest qualification probability, the tuple that remains in that x-tuple may have a chance to give the highest value than all other tuples, yielding a high-quality result; and so the expected improvement in quality is also higher than the case of PRQ.

We also compare the quality improvement (I) of PRQ and PMaxQ, relative to their original quality (S). We assume the DP algorithm is used to obtain the x-tuples. The results, shown in Figure 7.11, reveals that under a fixed query budget and query answer size, the relative quality improvement ($I/|S|$) for PMaxQ is consistently higher than that of PRQ. The main reason is that the PMaxQ has fewer distinct PW-results than the PRQ. This implies there are less tuples in the answer of a PMaxQ than that of a PRQ. Cleaning an x-tuple for PMaxQ, therefore, has more impact than cleaning an x-tuple for PRQ. In environments where multiple queries are concurrently executed, a system one may therefore choose to place more effort on PMaxQ than on PRQ.

4. Results on the Real Dataset

We now present selected results for the real dataset. Figure 12(a) shows the quality of PRQ under different values of z. Similar to Figure 7.3, the quality of PRQ worsens as z increases. On the other hand, PNNQ has an average score of -0.86. The reason for the high quality obtained by PNNQ

is that the dataset also has a high quality: on average, each x-tuple has 2 tuples, and 33% of the x-tuples have no uncertainty (i.e., they only have one single tuple). Thus, it is easy to obtain an unambiguous answer for PNNQ.

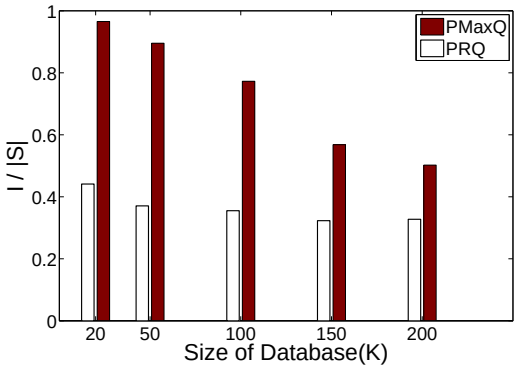
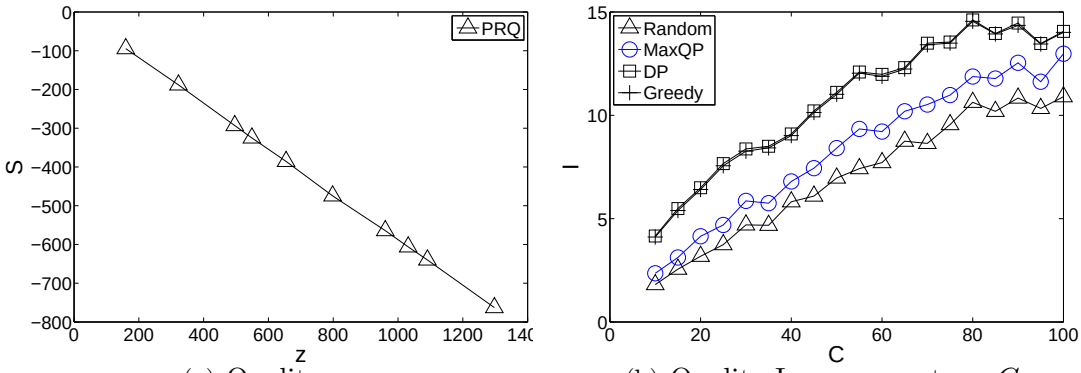


Figure 7.11: PRQ vs. PMaxQ ($I/|S|$).

We also observe that the PWS-quality scores of the queries in the real dataset are generally higher than those obtained for the synthetic dataset. To understand why, for each x-tuple, we have measured the entropy of the existential probabilities of all tuples in a x-tuple. We found that the average of these entropy values over the real dataset is 0.78, which is lower than that of the synthetic dataset (1.85). Thus, the real dataset is generally less uncertain than the synthetic one, and the PWS-quality scores for the real dataset are also better.

Finally, Figure 12(b) shows the quality improvement of PRQ under different query budgets. The results are similar to those for the synthetic data (Figure 7.9). We have also measured the quality improvement for the PNNQ. Since its original quality score is high, the data cleaning algorithms does not have much effect on the quality. Thus, we do not show their results here.



(a) Quality vs. z. (b) Quality Improvement vs. C.

Figure 7.12: Results on Real Data Set

7.6 Chapter Summary

The management of uncertain and probabilistic databases has become an important topic in emerging applications. In this chapter, we investigated a cleaning problem for these databases, with the goal of optimizing the expected quality improvement under a limited budget. To accomplish this task, we designed the PWS-quality metric to quantify query answer ambiguities. We showed how PWS-quality can be efficiently computed for common entity-based queries (PRQ and PMaxQ). We also illustrated that it is possible to develop optimal and efficient solutions around this metric.

We plan to extend our solutions to support other kinds of queries, e.g., top-k query. We will also examine other cleaning models, e.g., a cleaning request that may not be immediately accomplished. We can also investigate how to perform optimal cleaning where an x-tuple, after cleaning, becomes a set of tuples with arbitrary distributions. It is also interesting to study how cleaning can be done on databases where the uncertainty of attributes is given by a continuous distribution (e.g., [76, 70]).

8 Conclusions and Future Works

The uncertain database plays a more and more important role in many arising applications. We addressed two essential issues of managing imprecise data, i.e. querying and cleaning uncertain database. We pointed out the difficulties of evaluating several important kinds of probabilistic queries, and proposed efficient approaches for answering them. We also investigated the techniques of optimizing query answer quality by cleaning uncertain database.

We proposed a series of approaches in order to improve the efficiency of evaluating the imprecise location-dependent queries. The Minkowski Sum and the expanded query can help pruning the data objects with no chances to satisfy the query requirements. The observation of the query-data duality property enables the reduction of the cost to compute the qualification probabilities. If the probability information of the data objects are stored in the index, more pruning opportunities can be created.

We identified the problem of high computational complexity for PNN evaluation, and proposed to study the C-PNN, a variant of PNN. We developed probabilistic verifiers to reduce the chance of calculating qualification probabilities, and the incremental refinement algorithm for facilitating probability computation.

We also studied the probability threshold k-NN Query (T-k-PNN) for uncertain databases. We proposed various pruning techniques with consideration of both distance and probability constraints. With the k-bound filtering technique, a lot of unqualified objects can be pruned. The number of k-subsets can be significantly reduced by the PCS algorithm. We further demonstrated the efficient computation of lower and upper bounds of probabilities with the aid of partition information, based on the techniques for evaluating C-PNN, i.e. verification and incremental refinement.

Finally, we discussed the issues of cleaning the uncertain database, with the goal of optimizing the expected quality improvement under a limited budget. To accomplish this task, we designed the PWS-quality metric to quantify query answer ambiguities. We showed how PWS-quality can be efficiently computed for common entity-based queries (PRQ and PMaxQ). We also illustrated that it is possible to develop optimal and efficient solutions around this metric.

8.1 Future Works

Recently, researchers have proposed several probabilistic queries with complex ranking semantics, such as Skyline [69, 58, 90] and reverse nearest neighbor [60]. We plan to extend the techniques proposed in this thesis for processing these queries. Furthermore, it might be possible to find a unified framework for evaluating a variety of ranking-based probabilistic queries like Min/Max, NN, and Skyline etc.

We noticed that the continuous version of the probabilistic queries has attracted attentions from the database community [82]. The evaluation of the continuous queries must be finished in short time otherwise the values of the data may change and the results will be meaningless. It thus brings a tighter requirement for the evaluation cost. We are currently working on an idea of reusing the answers of the continuous queries in order to improve the query execution performance.

Another arising issue of uncertain data management is to process queries over uncertain graph, e.g. XML or road network [52, 16]. Researchers have proposed several methods for representing the uncertain data as probabilistic graph. Efficient methods are needed for processing queries over these graph with uncertainty. In particular, traditional techniques for querying graph

could be incorporated with the approaches of handling probabilistic queries.

We are also looking for the opportunities of effectively and efficiently conducting uncertain data integration and cleaning [34, 31, 53]. We are now trying to extend our solutions to support other kinds of queries, e.g., top-k query, and examine other cleaning models.

References

[1] The orion databae system. http://orion.cs.purdue.edu/.

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *SIGMOD Rec.*, 16(3):34–48, 1987.

[3] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: a system for data, uncertainty, and lineage. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 1151–1154. VLDB Endowment, 2006.

[4] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 30, Washington, DC, USA, 2006. IEEE Computer Society.

[5] L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 713–724, New York, NY, USA, 2007. ACM.

[6] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowl. and Data Eng.*, 4(5):487–502, 1992.

[7] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 953–964, 2006.

[8] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[9] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications, 2nd ed.* Springer Verlag, 2000.

[10] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. In *VLDB '08: Proceedings of the 34th international conference on Very large data bases*. VLDB Endowment, 2008.

[11] C. Böhm, M. Gruber, P. Kunath, A. Pryakhin, and M. Schubert. Prover: Probabilistic video retrieval using the gauss-tree. In *ICDE '07:Proceedings of the 23rd International Conference on Data Engineering*, pages 1521–1522, 2007.

[12] C. Böhm, A. Pryakhin, and M. Schubert. The gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *ICDE '06:Proceedings of the 22nd International Conference on Data Engineering*, page 9, 2006.

[13] B. P. BUCKLES and F. E. PETRY. A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems*, 7:213–226, 1982.

[14] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases*, pages 71–81, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[15] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC '77: Proceedings of the ninth*

annual ACM symposium on Theory of computing, pages 77–90, New York, NY, USA, 1977. ACM.

[16] L. Chang, J. X. Yu, and L. Qin. Query ranking in probabilistic xml data. In *EDBT'09: Proceedings of the 12nd International Conference on Extending Database Technology*, pages 156–167, 2009.

[17] M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei. Probabilistic reverse nearest neighbor queries on uncertain data. Technical Report UNSW-CSE-TR-0816, July 2008.

[18] J. Chen and R. Cheng. Efficient evaluation of imprecise location-dependent queries. In *ICDE '07:Proceedings of the 23rd International Conference on Data Engineering*, pages 586–595, 2007.

[19] J. Chen and R. Cheng. Quality-aware probing of uncertain data with resource constraints. In *SSDBM '08:Proceedings of the 20th International Conference on Scientific and Statistical Database Management*, pages 491–508, 2008.

[20] R. Cheng, J. Chen, M. F. Mokbel, and C.-Y. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE '08: Proceedings of the 24th International Conference on Data Engineering*, pages 973–982, 2008.

[21] R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees. In *VLDB '08: Proceedings of the 34th international conference on Very large data bases*, pages 722–735. VLDB Endowment, 2008.

[22] R. Cheng, L. Chen, J. Chen, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *EDBT'09: Pro-*

ceedings of the 12nd International Conference on Extending Database Technology, pages 672–683, 2009.

[23] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD Conference*, pages 551–562, 2003.

[24] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1112–1127, 2004.

[25] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. *Inf. Syst.*, 32(1):104–130, 2007.

[26] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB '04:Proceedings of the 30th international conference on Very large data bases*, pages 876–887, 2004.

[27] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar. Preserving user location privacy in mobile data management infrastructures. In *Proc. of the 6th Workshop on Privacy Enhancing Technologies*, June 2006.

[28] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

[29] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[30] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[31] A. de Keijzer and M. van Keulen. Imprecise: Goodis-good-enough data integration. In *ICDE '08: Proceedings of the 24th International Conference on Data Engineering*, pages 1548–1551, 2008.

[32] M. de Rougemont. The reliability of queries. In *PODS '95: Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 286–291, New York, NY, USA, 1995. ACM.

[33] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 588–599. VLDB Endowment, 2004.

[34] T. Detwiler, W. Gatterbauer, B. Louie, D. Suciu, and P. Tarczy-Hornoch. Integrating and ranking uncertain scientific data. In *ICDE'09: Proceedings of the 25th International Conference on Data Engineering*, pages 1235–1238, 2009.

[35] G. Diubin. The average behaviour of greedy algorithms for the knapsack problem: general distributions. *Mathematical Methods of Operations Research*, 57(3), 2003.

[36] X. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 687–698. VLDB Endowment, 2007.

[37] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[38] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[39] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani. Streaming algorithms for robust, real-time detection of ddos attacks. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 4, Washington, DC, USA, 2007. IEEE Computer Society.

[40] B. Gedik and L. Liu. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *EDBT '04: the 9th International Conference on Extending Database Technology*, pages 67–87, 2004.

[41] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *ICDCS*, pages 620–629, 2005.

[42] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 227–234, New York, NY, USA, 1998. ACM.

[43] G. Grahne. Dependency satisfaction in databases with incomplete information. In *VLDB '84: Proceedings of the 10th International Conference on Very Large Data Bases*, pages 37–45, San Francisco, CA, USA, 1984. Morgan Kaufmann Publishers Inc.

[44] J. Grant. Null values in a relational data base. *Inf. Process. Lett.*, 6(5):156–157, 1977.

[45] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM.

[46] H. Hu and D. L. Lee. Range nearest-neighbor query. *IEEE Trans. on Knowl. and Data Eng.*, 18(1):78–91, 2006.

[47] S. Ilarri, E. Mena, and A. Illarramendi. Location-dependent queries in mobile contexts: Distributed processing using mobile agents. *IEEE Trans. Mob. Comput.*, 5(8):1029–1043, 2006.

[48] T. Imieliński and W. Lipski. Incomplete information in relational databases. *Journal of the Association for Computing Machinery*, 31(4):761–79, 1984.

[49] T. Imieliński, S. Naqvi, and K. Vadaparty. Incomplete object—a data model for design and planning applications. In *SIGMOD '91: Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, pages 288–297, New York, NY, USA, 1991. ACM.

[50] G. S. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 512–523. VLDB Endowment, 2003.

[51] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *MobiDE '06: Proceedings of the Fifth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 43–50, 2006.

[52] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic xml. In *VLDB '07:Proceedings of the 33rd international conference on Very large data bases*, pages 27–38, 2007.

[53] C. Koch and D. Olteanu. Conditioning probabilistic databases. In *VLDB '08: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2008.

[54] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 804–815. VLDB Endowment, 2004.

[55] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA '07:Proceedings of the International Conference on Database Systems for Advanced Applications*, pages 337–348, 2007.

[56] I. Lazaridis and S. Mehrotra. Approximate selection queries over imprecise data. In *ICDE*, 2004.

[57] D. L. Lee, J. Xu, B. Zheng, and W.-C. Lee. Data management in location-dependent information services. *IEEE Pervasive Computing*, 1(3):65–72, 2002.

[58] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 213–226, New York, NY, USA, 2008. ACM.

[59] X. Lian and L. Chen. Probabilistic group nearest neighbor queries in uncertain databases. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):809–824, 2008.

[60] X. Lian and L. Chen. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. Accepted to appear in Very Large Data Bases Journal (VLDBJ), 2009.

[61] Z. Liu, K. C. Sia, and J. Cho. Cost-efficient processing of min/max queries over distributed sensors with uncertainty. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 634–641, New York, NY, USA, 2005. ACM.

[62] V. Ljosa and A. K. Singh. Apla: Indexing arbitrary probability distributions. In *ICDE*, pages 946–955, 2007.

[63] M.Hadjieleftheriou. Spatial index library version 0.44.2b.

[64] A. moving rating database. http://infolab.stanford.edu/trio/code/index.html.

[65] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

[66] M. Nodine and J. Vitter. Greed sort: An optimal sorting algorithm for multiple disks. *JACM*, 42(4), 1995.

[67] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2003. ACM.

[68] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB'00: Proceedings of 26th International Conference on Very Large Data Bases*, pages 144–155. Morgan Kaufmann, 2000.

[69] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 15–26. VLDB Endowment, 2007.

[70] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *SSD '99: Proceedings of the 6th International Symposium on Advances in Spatial Databases*, pages 111–132, London, UK, 1999. Springer-Verlag.

[71] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. *Inf. Sci.*, 34(2):115–143, 1984.

[72] Y. Qi, S. Singh, R. Shah, and S. Prabhakar. Indexing probabilistic nearest-neighbor threshold queries. In *Proceedings of Workshop on Management of Uncertain Data (MUD)*, 2008.

[73] C. Shannon. *The Mathematical Theory of Communication.* University of Illinois Press, 1949.

[74] A. Silberstein, R. Braynard, C. S. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 68, 2006.

[75] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. E. Hambrusch. Indexing uncertain categorical data. In *ICDE '07: Proceedings of the 23rd International Conference on Data Engineering*, pages 616–625, 2007.

[76] P. A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*, pages 310–337. Springer Verlag, 1998.

[77] M. Soliman, I. Ilyas, and K. Chang. Top-k query processing in uncertain databases. In *ICDE '07: Proceedings of the 23rd International Conference on Data Engineering*, pages 896–905, 2007.

[78] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 79–96, London, UK, 2001. Springer-Verlag.

[79] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 922–933. VLDB Endowment, 2005.

[80] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 287–298. VLDB Endowment, 2002.

[81] Y. Tao, X. Xiao, and R. Cheng. Range search on multidimensional uncertain data. *ACM Trans. Database Syst.*, 32(3):15, 2007.

[82] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajec-

tories. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 874–885, 2009.

[83] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. The geometry of uncertainty in moving objects databases. In *EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology*, pages 233–250, London, UK, 2002. Springer-Verlag.

[84] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic xml approach to data integration. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 459–470, Washington, DC, USA, 2005. IEEE Computer Society.

[85] Y. Vassiliou. Null values in data base management a denotational semantics approach. In *SIGMOD '79: Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 162–169, New York, NY, USA, 1979. ACM.

[86] J. Warrior, E. McHenry, and K. McGee. They know where you are. *Spectrum*, 40(7):20– 25, July 2003.

[87] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distrib. Parallel Databases*, 7(3):257–387, 1999.

[88] E. Wong. A statistical approach to incomplete information in database systems. *ACM Trans. Database Syst.*, 7(3):470–488, 1982.

[89] M. Zemankova and A. Kandel. Implementing imprecision in information systems. *Inf. Sci.*, 37(1-3):107–141, 1985.

[90] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. Yu. Probabilistic skyline operator over sliding windows. In *ICDE'09: the 25th International Conference on Data Engineering*, pages 1060–1071, 2009.