Quality-Aware Probing of Uncertain Data with Resource Constraints

Jinchuan Chen and Reynold Cheng

Department of Computing,
Hong Kong Polytechnic University,
Hung Hom, Kowloon, Hong Kong
{csjcchen,csckcheng}@comp.polyu.edu.hk

Abstract. 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 (pdf), as well as the notion of probabilistic queries, which provide statistical guarantees on answer correctness. In this paper, 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.

1 Introduction

In many emerging and important applications like wireless sensor networks and location-based applications, the data obtained from the sensing devices are often imprecise [10,17,18]. Consider a monitoring application that employs a sensor network to obtain readings from external environments. Due to imperfection of physical devices, as well as limited battery power and network delay, it is often infeasible to obtain accurate readings. As a result, the data maintained in the monitoring applications are often contaminated with noises (e.g., sampling and measurement error). The *uncertainty* of these data should be modeled and handled carefully, or else the quality of the services or queries provided to users can be affected [4,10].

One commonly-used uncertainty model assumes that the exact value of a data item is located within a closed region, together with a probability distribution

B. Ludäscher and Nikos Mamoulis (Eds.): SSDBM 2008, LNCS 5069, pp. 491–508, 2008.

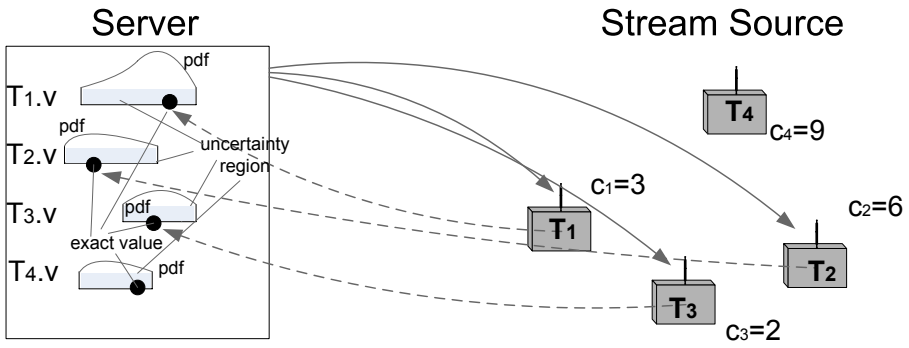

Fig. 1. Probing of Sensor Data for Uncertainty Reduction

function (*pdf*) of that value in the region [10,13,21]. An example is shown in Figure 1, where a monitoring server maintains the pdf of the temperature values acquired from four wireless sensors $(T_1, \ldots, T_4)$. Each of these pdf's is confined within a closed range of possible values, to model the fact that the data has not been updated for an extensive amount of time. These pdf's could be derived through techniques like time-series or model-based analysis [5,10]. In general, a pdf which spans over a larger uncertainty region is more vague (or uncertain) than the one with a smaller region.

To process uncertain data, *probabilistic queries* [4,21] have been proposed. These are the "probabilistic" counterparts of spatial queries, such as range queries and nearest neighbor queries. Probabilistic queries produce *imprecise* results, which are essentially answers that are augmented with probability values to indicate the likelihood of their occurrences. For example, a probabilistic range query, inquiring which of the four sensor data values in Figure 1 have non-zero probabilities of being inside a specified range $[10^oC, 20^oC]$, may produce an answer like: $\{(T_1, 0.9), (T_2, 0.5)\}$. This answer indicates that T_1 (T_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 impreciseness of the data being evaluated. In the previous example, since T_1 has a chance of 0.9 for satisfying the query, we know that T_1 is very likely to be located inside $[10^oC, 20^oC]$. The case of T_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 [4,15]. 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 paper, we define a quality score for a probabilistic range query based on the definition

of entropy [20]. 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 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 $(T_1, \ldots, T_4)$ 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 1, where $c_1, \ldots, c_4$ are the respective costs for probing $T_1, \ldots, T_4$. 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 $\{T_1\}$, $\{T_2\}$, $\{T_3\}$, $\{T_1, T_2\}$ and $\{T_2, T_3\}$. 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 T_2 and T_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 paper addresses the balance between query quality and the amount of system resources consumed. A few related problems have been studied in [10,16], 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 paper is organized as follows. In Section 2, we present the related work. Section 3 illustrates the system architecture. We discuss the details of quality and resource budget for probabilistic range queries in Section 4. Then we give our solutions in Section 5. We report our experimental results in Section 6. Section 7 concludes the paper.

2 Related Work

In this section, we summarize the work done in probing and evaluation of probabilistic queries.

Probing Plans. 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 [19], efficient algorithms are derived to fetch remote data items in order to generate a satisfactory result quickly. Liu et al. [16] 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 paper, on the other hand, defines a quality metric for probabilistic query answers, and use this measure to devise probing plans. Although Madden et al. [8,10,11] 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 proposes a feasible probing plan that achieves the highest quality under limited resource constraints. Our solutions can be applied to multi-dimensional uncertain data with artibtrary pdfs. We also use the amount of

information gain (i.e. entropy) as the quality metric in our probing solutions, and this has not been studied before.

Probabilistic Queries. There are plenty of recent studies about efficient evaluation of probabilistic queries for large uncertain databases. In [3,4,5], efficient algorithms of evaluating probabilistic nearest-neighbor queries are proposed, which evaluate uncertain location data and provide probabilistic guarantees in answers. In [1], efficient methods for evaluating probabilistic location-dependent queries are studied. Indexing of probabilistic range queries is considered in [7], and the solution is extended to handle multi-dimensional uncertainty in [6,22]. The evaluation of probabilistic queries in sensor networks is considered in [2,10,14]. In this paper, we illustrate our probing techniques by using the probabilistic query evaluation methods in [4]. However, other advanced query evaluation or indexing techniques can also be used together with our probing algorithm.

3 System Architecture

Figure 2 describes the architecture of the system used in this paper. The *Data Manager* caches the value ranges and corresponding pdf of remote 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.

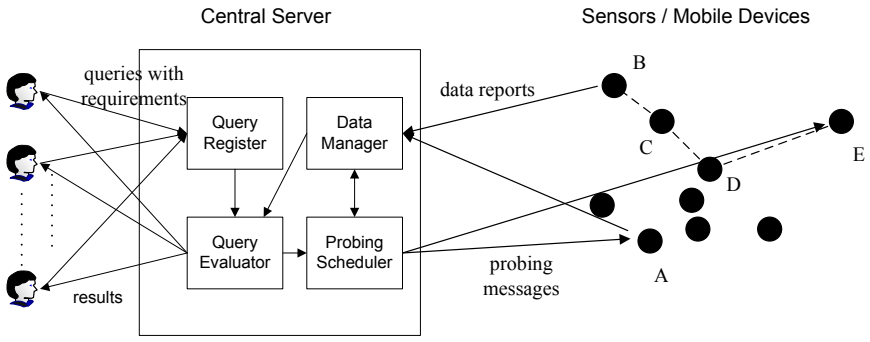

Fig. 2. System Architecture

– 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.

Uncertainty Model. We assume there are D data sources, namely $T_i (i = 1, 2, ..., D)$. Each data source T_i has an actual value, denoted by $T_i.v$, where $T_i.v \in R$. The uncertainty model of each T_i cached at the server consists of an uncertain region $[l_i, u_i]$, together with a pdf f_i. 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.

4 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 paper. Section 4.1 details the definition and evaluation of probabilistic range queries. In Section 4.2 we present a quality metric for probabilistic range queries. Section 4.3 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.

4.1 Probabilistic Range Query

The Probabilistic Range Query (PRQ)[4] returns a set of data objects, with the probabilities that their attribute values are in the specified range, called *qualification probabilities*:

Definition 1. Probabilistic Range Query (PRQ): *Given a closed interval* $[a, b]$*, where* $a, b \in R$ *and* $a \leq b$*, a PRQ (denoted by* Q*), returns a set of tuples* (T_i, p_i)*, where* p_i *is the non-zero probability that* $T_i.v \in [a, b]$.

To illustrate, Figure 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. The result of Q_1 becomes: $(A, 0.25)$, $(B, 0.5)$, $(C, 0.75)$.

In general, the value of the qualification probability, i.e., p_i, can be calculated by using the Equation 1 [4].

$$p_i = \int_{R_i} f_i(x)dx \tag{1}$$

where R_i is the overlapping region of the query range $[a, b]$ and $[l_i, u_i]$, and $f_i(x)$ is the uncertainty pdf of item T_i. In Figure 3, we shade the overlapping region of all the data items with the query range of Q_1 and Q_2.

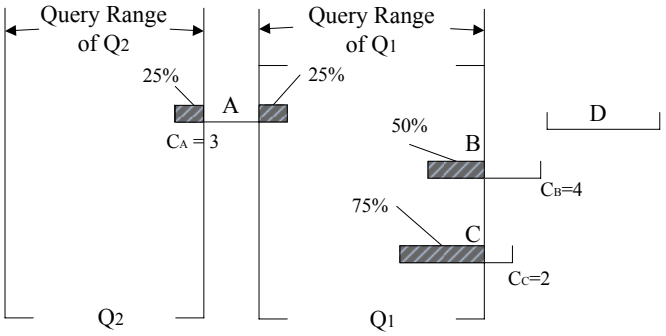

Fig. 3. An Example of Probabilistic Range Query

4.2 Quality Score

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[20]. 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 2. 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:*

$$H(X) = -\sum_{i=1}^{n} p(X_i) log_2 p(X_i) \tag{2}$$

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

$$g_i = -p_i log_2 p_i - (1 - p_i) log_2 (1 - p_i) \tag{3}$$

We then use the sum of the 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 $(T_1, p_1), (T_2, p_2), ..., (T_n, p_n)$, the *quality score*, denoted by H, of this result is defined by

$$H = -\sum_{i=1}^{n} (p_i log_2 p_i + (1 - p_i) log_2 (1 - p_i)) \tag{4}$$

By substituting Equation 3 into Equation 4, we have

$$H = \sum_{i=1}^{n} g_i \tag{5}$$

A larger 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 T_i, its uncertainty region shrinks to a point, and the server knows exactly whether T_i satisfies the range query. Thus, p_i equals to either zero or one. The corresponding ambiguity caused by the answer (T_i, p_i) is then "removed", and the entropy of the overall query result is reduced by an amount given by Equation 3 . We denote this amount of entropy reduction as the *gain* of probing T_i, denoted by g_i. As shown in Equation 3 the value of g_i only depends on the qualification probability of a single object T_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.

4.3 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 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 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

Table 1. Notations

Notation	Description
T	A remote stream source
$T.v$	The exact value of T
$[l, u]$	Lower and upper bounds of $T.v$
f	Probability distribution function of the $T.v$
Q	Probabilistic range query
C	Resource constraint assigned to Q
c	# of messages for probing T
H	Precision quality (entropy)
p	The probability that T satisfies Q
g	The benefit of probing T
n	# of items in the result set

needed for probing an item. We also use c_i to denote the number of messages for probing T_i. We list the notations used in this paper in Table 1.

5 Maximizing Quality with Limited Resources

As we have mentioned in Section 4.2, 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 5.1 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 5.2. We give heuristics which provide close-to-optimal performance in Section 5.3.

5.1 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,$\{p_1, p_2, ..., p_n\}$, of all the items $T_i (i = 1, ..., n)$ such that $p_i > 0$. The cost of probing T_i is c_i. Let the gain obtained by probing T_i be g_i (Equation 3). We formally define the *Single Query (SQ) problem* as follows.

$Maximize \sum_{i=1}^{n} x_i \cdot g_i$
$subject\ to \sum_{i=1}^{n} x_i \cdot c_i \leq C$
$x_i \in \{0, 1\}, i = 1, 2, ..., n$

Here we use an array $\overline{X} = x_1, x_2, ..., x_n$ to record the choices. Initially, all the values of x_i are zero. If item T_i is chosen for probing, we set x_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 $N = \{T_1, T_2, ..., T_n\}$, whose p_i's are all nonzero. Suppose we have found the optimal set $S = \{T_{\gamma_1}, T_{\gamma_2}, ..., T_{\gamma_m}\}$ $(m \leq n \wedge \gamma_i \in [1, n])$ 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. T_{γ_1}, from the candidate item set, and reducing the budget to $C - c_{\gamma_1}$. That is, we consider a subproblem $P(C - c_{\gamma_1}, N/\{T_{\gamma_1}\})$. If $S_1 = S/\{T_{\gamma_1}\} = \{T_{\gamma_2}, ..., T_{\gamma_m}\}$, 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 $P(C - c_{\gamma_1}, N/\{T_{\gamma_1}\})$.

Proof. Suppose S_1 is not the optimal set for $P(C - c_{\gamma_1}, N/\{T_{\gamma_1}\})$, 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 $S_1' \cup \{T_{\gamma_1}\}$. 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 $S_1 \cup \{T_{\gamma_1}\}$, or S. Thus $S_1' \cup \{T_{\gamma_1}\}$ 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 $P(C - c_{\gamma_1}, N/\{T_{\gamma_1}\})$. □

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 $\{T_1, ..., T_i\}$. 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 T_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 T_i, there are two possible choices. Either T_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 $\{T_1, T_2, ..., T_{i-1}\}$ 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 T_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 T_i is not chosen, while Steps 7-9 construct the optimal set and the corresponding gain if T_i is chosen. Another point to notice is, in order to put T_i into the solution set, the cost of probing T_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 $c = (c_1, c_2, ..., c_n)$
An array of gains $g = (g_1, g_2, ..., g_n)$
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** $v[k, i-1] > v[k - c_i, i-1] + g_i$
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** $s[C, n]$

Fig. 4. Algorithm DP for SQ

Using Algorithm DP, we can find an optimal solution for the SQ problem. We will show soon that Algorithm DP 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 DP. The computational 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 DP is $O(n^2C)$.

5.2 Multiple Queries with Shared Budget (MQSB)

In many cases, more than one query are processed at the server simultaneously. A data item T_i may be involved in the results of multiple queries. By probing T_i, all queries containing it in their results will have a better quality. In order to apply Algorithm DP in this scenario, we need to change the method of calculating gain, i.e. Equation 3. Suppose there are m queries, $Q_1, Q_2, ..., Q_m$, we can have a set of m values for T_i, $p_{i1}, p_{i2}, ..., p_{im}$, where $p_{ij} (j = 1, 2, .., m)$ specifies the probability that T_i satisfies Q_j. After getting the exact value of T_i the result precision of these queries will be improved by H_{ij}. Here H_{ij}, the gain for Q_j obtained by probing T_i, is equal to $-p_{ij}log_2p_{ij} - (1 - p_{ij})log_2(1 - p_{ij})$, where $j = 1, 2, ..., m$(Equation 3). The gain of probing T_i is the sum of H_{ij}, or

$$G_i = \sum_{j=1}^{m} H_{ij} \tag{6}$$

For example, as in Figure 3, item A overlaps with the ranges of both Q_1 and Q_2, where $p_{A1} = p_{A2} = 0.25$. Thus $g_A = -2 \cdot (0.25 \cdot log_2 0.25 + 0.75 \cdot log_2 0.75) = 1.62$.

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) to replace g_i (Equation 3). Therefore, Algorithm DP is also suitable for solving MQSB. Moreover, the approximate solutions, which will be discussed in Section 5.3, 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 in the MQSB scenario, which is $O(nm)$. Thus, the computational complexity of Algorithm DP would be $O(nmC + nm) = O(nmC)$ in the MQSB scenario, and the memory complexity is $O((nm)^2C)$.

5.3 Approximate Solutions

Greedy. 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 7 shows how to compute the value of e_i.

$$e_i = \frac{g_i}{c_i} \tag{7}$$

Input An array of probing costs $c = (c_1, c_2, ..., c_n)$
An array of gains $g = (g_1, g_2, ..., g_n)$
The resource budget C
Output The optimal set

1. $d := sort(c, g)$
2. $b := C$
3. **for** $i := 1$ **to** n **do**
4. **if** $b \geq c_{d[i]}$
5. $s[d[i]]:=1$
6. $b := b - c_{d[i]}$
7. **return** s

Fig. 5. 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.

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.

Table 2 compares the complexities of the above algorithms in the SQ scenario.

Table 2. Complexity of Four Algorithms (SQ)

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 3. Complexity of Four Algorithms (MQSB)

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

For MQSB, the complexities of the optimal and approximate solutions are listed in Table 3. They are derived by substituting the value of n in Table 2 by nm.

6 Experimental Results

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

6.1 Experiment Settings

We use a realistic data set, called Long Beach[1], 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.

6.2 Results

Effectiveness Analysis. Figure 6 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

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

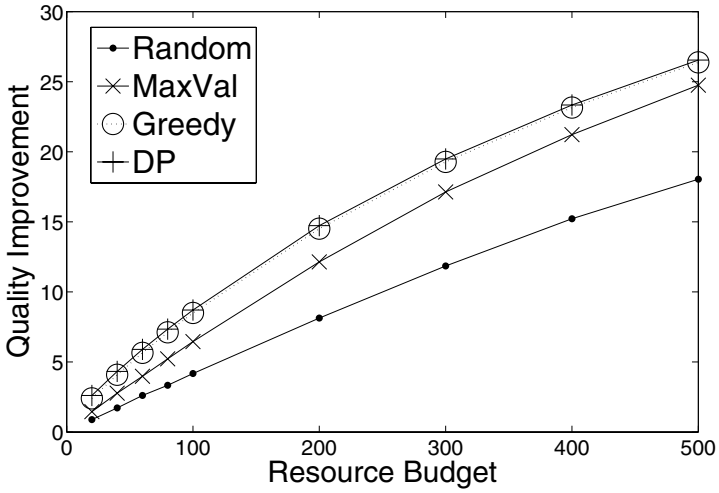

Fig. 6. Quality Improvement vs. Resource Budget (SQ)

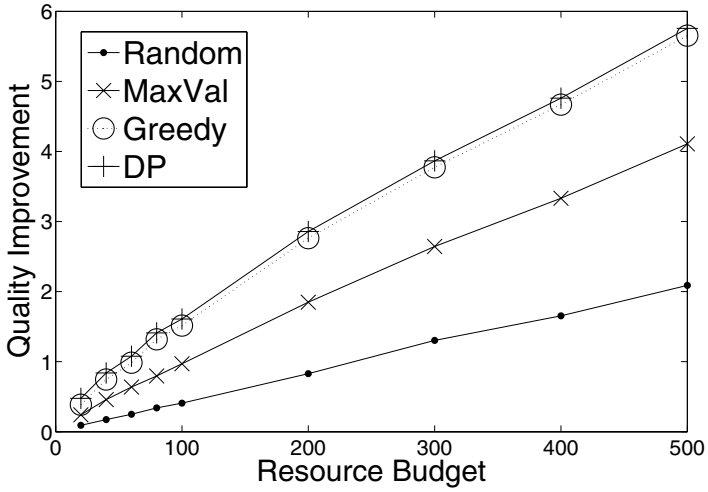

Fig. 7. Quality Improvement vs. Resource Budget (MQSB)

results. As shown in Figure 6, 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 [9], and it has been shown in [12] that the average performance of a greedy solution is close to the optimal one.

Figure 7 illustrates similar results for MQSB. In these experiments, 10 queries are executed concurrently in a batch.

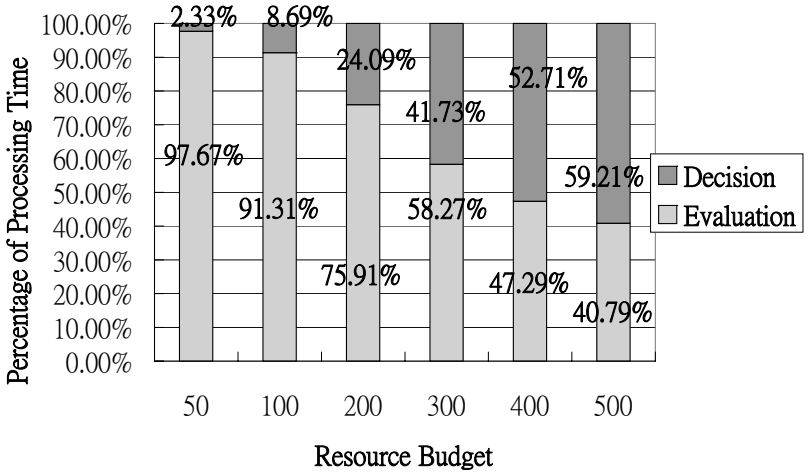

Fig. 8. Time Spent in Different Phases during Query Processing (SQ)

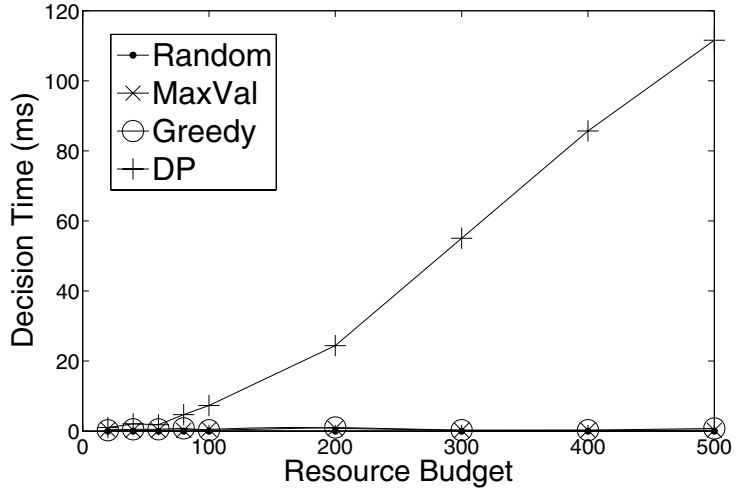

Fig. 9. Decision Time vs. Resource Budget (SQ)

Performance Analysis. Figure 8 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 3), and (2) Decision - the time for deciding the probing set contents (Step 2 in Section 3). 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 8, the Decision

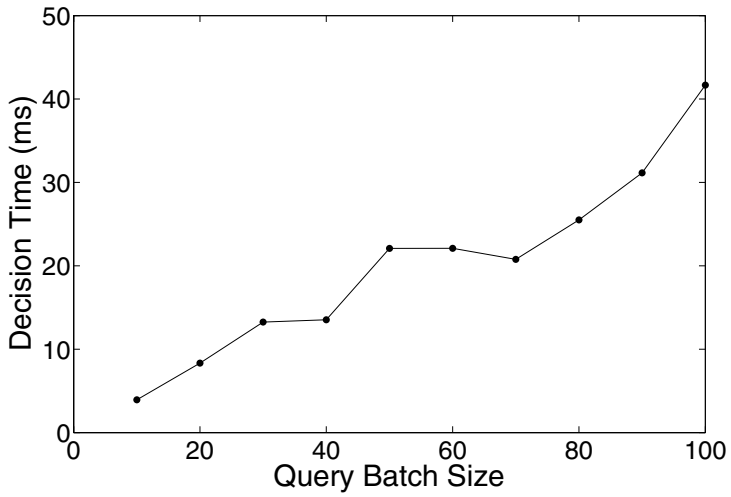

Fig. 10. Scalability of MQSB (Greedy)

step costs more time as the resource budget increases. The reason is that more choices are available with a larger resource budget.

Figure 9 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 10 illustrates the time required for finding the probing sets using the Greedy algorithm. The resource budget is set to 100. 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.

7 Conclusions

The evaluation of probabilistic queries over uncertain data has attracted plenty of research interest in recent years. In this paper, 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.

Acknowledgements

The work described in this paper was supported by the Research Grants Council of the Hong Kong SAR, China (Project No. PolyU 5133/07E), and the Germany/HK Joint Research Scheme (Project No. G_HK013/06). We would like to thank the anonymous reviewers for their insightful comments and suggestions.

References

1. Chen, J., Cheng, R.: Efficient evaluation of imprecise location-dependent queries. In: ICDE (2007)
2. Cheng, R., Chan, E., Lam, K.Y.: Efficient quality assurance of probabilistic queries in sensor networks. Sensor Network and Configuration: Fundamentals, Techniques, Platforms, and Experiments (May 2006)
3. Cheng, R., Chen, J., Mokbel, M., Chow, C.-Y.: Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In: ICDE (2008)
4. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proc. ACM SIGMOD (2003)
5. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. Information Systems Journal (2006)
6. Cheng, R., Xia, Y., Prabhakar, S., Shah, R.: Change tolerant indexing on constantly evolving data. In: ICDE (2005)
7. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.S.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: Proc. VLDB (2004)
8. Chu, D., Deshpande, A., Hellerstein, J., Hong, W.: Approximate data collection in sensor networks using probabilistic models. In: ICDE (2006)
9. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. The MIT Press, Cambridge (2001)
10. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: VLDB (2004)
11. Despande, A., Hong, W., Guestrin, C., Madden, S.R.: Exploiting correlated attributes in acquisitional query processing. In: ICDE (2005)
12. Diubin, G.: The average behaviour of greedy algorithms for the knapsack problem: general distributions. Mathematical Methods of Operations Research 57(3) (2003)
13. Pfoser, D., Jensen, C.: Capturing the uncertainty of moving-objects representations. In: Proc. SSDBM (1999)
14. Han, S., Chan, E., Cheng, R., Lam, K.Y.: A statistics-based sensor selection scheme for continuous probabilistic queries in sensor network. Real Time Systems Journal (RTS) (2006)
15. Lazaridis, I., Mehrotra, S.: Approximate selection queries over imprecise data. In: ICDE (2004)
16. Liu, Z., Sia, K.C., Cho, J.: Cost-efficient processing of min/max queries over distributed sensors with uncertainty. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897. Springer, Heidelberg (2006)
17. Ljosa, V., Singh, A.: APLA: Indexing arbitrary probability distributions. In: ICDE (2007)

18. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: SIGMOD (2003)
19. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: VLDB (2000)
20. Shannon, C.: The Mathematical Theory of Communication. University of Illinois Press (1949)
21. Sistla, P.A., Wolfson, O., Chamberlain, S., Dao, S.: Querying the uncertain position of moving objects. In: Temporal Databases: Research and Practice. Springer, Heidelberg (1998)
22. Tao, Y., Xiao, X., Cheng, R.: Range search on multidimensional uncertain data. ACM Transactions on Database Systems 32(15) (2007)