Approximate Selection Queries over Imprecise Data

Iosif Lazaridis
Information and Computer Science
University of California, Irvine
Irvine, CA
U.S.A.
iosif@ics.uci.edu

Sharad Mehrotra
Information and Computer Science
University of California, Irvine
Irvine, CA
U.S.A.
sharad@ics.uci.edu

Abstract

We examine the problem of evaluating selection queries over imprecisely represented objects. Such objects are used either because they are much smaller in size than the precise ones (e.g., compressed versions of time series), or as imprecise replicas of fast-changing objects across the network (e.g., interval approximations for time-varying sensor readings). It may be impossible to determine whether an imprecise object meets the selection predicate. Additionally, the objects appearing in the output are also imprecise. Retrieving the precise objects themselves (at additional cost) can be used to increase the quality of the reported answer.

In our paper we allow queries to specify their own answer quality requirements. We show how the query evaluation system may do the minimal amount of work to meet these requirements. Our work presents two important contributions: first, by considering queries with set-based answers, rather than the approximate aggregate queries over numerical data examined in the literature; second, by aiming to minimize the combined cost of both data processing and probe operations in a single framework. Thus, we establish that the answer accuracy/performance tradeoff can be realized in a more general setting than previously seen.

1 Introduction

We examine the problem of evaluating selection queries over imprecisely represented objects. An imprecise object o corresponds to an actual (precise) object ω^o which can be retrieved, at some cost, via a *probe* operation.

If $\mathcal{T}$ is a set of such objects, then we consider the selection query $\sigma_\lambda \mathcal{T}$, where λ is the selection predicate. λ maps objects to the $\{\text{YES}, \text{NO}, \text{MAYBE}\}$ set. When $\lambda(o) = \text{YES}$ then o satisfies λ. When $\lambda(o) = \text{NO}$ then o does not satisfy λ. Finally, when $\lambda(o) = \text{MAYBE}$, o *might* satisfy λ. This can be determined via a probe operation, and $\lambda(\omega^o)$ will return either YES or NO.

The *exact set* of λ over $\mathcal{T}$ is defined as:

$$\mathcal{E}_\mathcal{T}^\lambda = \{\omega^o | o \in \mathcal{T} \wedge \lambda(\omega^o) = \text{YES}\} \tag{1}$$

i.e., it consists of all precise objects which satisfy λ. To give an example, let $o_1 = [1, 3]$, $o_2 = [3, 4]$ and $o_3 = [-2, -1]$ be imprecise objects (intervals) corresponding to $\omega^{o_1} = 2.5, \omega^{o_2} = 3.2, \omega^{o_3} = -1.2$.[1] Then, the exact set for $\lambda(o) = (o \geq 2)$ and $\mathcal{T} = \{o_1, o_2, o_3\}$ is:

$$\mathcal{E}_\mathcal{T}^\lambda = \{\omega^{o_1}, \omega^{o_2}\} = \{2.5, 3.2\} \tag{2}$$

But, while $\lambda(o_2) = \text{YES}$ and $\lambda(o_3) = \text{NO}$, for $\lambda(o_1) = \text{MAYBE}$ we would not be able to ascertain that $\omega^{o_1} \in \mathcal{E}_\mathcal{T}^\lambda$ unless we performed the probe operation that would return ω^{o_1}. This is a cause of what we call *set-based uncertainty* in the output, since we are uncertain as to which objects belong in the exact set of the query.

There is, however, a *value-based uncertainty* as well. Since we only have o_1, o_2, o_3, then our answer set will have some of these imprecise objects, rather than the corresponding $\omega^{o_1}, \omega^{o_2}, \omega^{o_3}$. Hence, even if we knew that only o_1, o_2 satisfy the selection predicate, we would still have to deal with the problem that e.g., $o_1 = [1, 3]$ is only an imprecise representation of the precise $\omega^{o_1} = 2.5$.

What we propose to do in this paper is to show how results "close to" $\mathcal{E}_\mathcal{T}^\lambda$ can be obtained from sets of imprecise objects stored in a database. The queries will specify both the predicate λ, as well as *answer quality requirements* which the obtained result must satisfy. Such queries are termed *Quality-Aware Queries* (QaQs). There may be more than one feasible answer to a QaQ: our goal is to provide *some* answer which matches its requirements in the most *efficient* manner. This way, we aim to take advantage of the

[1]We use intervals as an example. Our technique works for all models of imprecision that allow us to classify imprecise objects as YES, NO, or MAYBE.

answer quality/performance tradeoff which has traditionally been studied for aggregate queries on numerical data.

Our work's novelty is twofold. First, we consider queries with set-based answers, for the selection operator, which is the first stepping stone towards integration of the quality/performance tradeoff for general relational queries. Approximate answering with set-based results has been treated in the past [4, 9], but not in a prescriptive setting where the user specifies an arbitrary quality requirement and the system tries to match this efficiently. Second, while previous work has focused in minimizing the expensive probe operations only, our goal is to take into account both the read/write and the probing cost.

We must emphasize here that we are proposing an online algorithm for the problem at hand. The selection operator over precise data is usually memoryless, examining one object at a time. Algorithms for some types of selections, namely nearest neighbor searches [18] sometimes use more than $O(1)$ memory to optimize I/O performance. At present, we limit ourselves to constant memory, and demonstrate that it is possible to achieve a significance performance benefit without e.g., ordering the sequence of probe operations in any way. Whether an even more significant performance benefit can be achieved with non-constant memory is a topic for further investigation.

1.1 Applicability

The generic problem setting and description presented so far has numerous real-world applications which occur whenever precise object storage and querying is made difficult because of:

- *Replication Barrier*.— If the precise objects are volatile, i.e., they are updated very frequently (as in web pages), or even continuously (as in sensor readings, or the location of moving objects), then it may be desirable, from a performance perspective, to avoid replicating them at the query processing site precisely.

- *Storage Barrier*.— If the precise objects are too large, then it may be impossible to store them precisely in a single location. For example, archives of multimedia objects, time series, or large documents may be stored in a separate location, or even in tertiary storage. Object summaries, such as feature-based representations for images, compressed time series, or abstracts of documents can be stored at a fraction of the precise object's space.

- *Querying Barrier*.— This occurs when either the query operates on very large databases, in which case precise evaluation may be prohibitively expensive, as in OLAP. Alternatively, the selection predicate, λ may be itself expensive, e.g., testing the Euclidean distance between two time series, or the edit distance between two strings of text.

The factors enumerated above involve our inability to store and query data precisely. A trivial solution is to invest in infrastructure, storage, and processing power and thus deal with the problem indirectly. Since many applications, e.g., for exploratory data analysis do not always require full accuracy, we may accommodate their quality requirements with existing hardware, by providing approximate results at a fraction of the cost needed for the exact ones.

Paper Organization.— The rest of the paper is organized as follows. In Section 2 we develop quality metrics for specifying and reporting the quality of an answer to a Quality-Aware Query. In Section 3 we present our algorithm for evaluating Quality-Aware Queries. In Section 4 we present the optimization framework that allows us to tune the free parameters of the evaluation algorithm. In Section 5 we present experimental results demonstrating the performance/quality tradeoff that our algorithm achieves. In Section 6 we present some related work. We conclude in Section 7, presenting directions of future research.

2 Quality Metrics

Quality Metrics for gauging the accuracy of a set-based result to a query include Match-and-Compare [9], and Earth Mover's Distance [19]. Such measures of quality are essentially *diagnostics* in the sense that they can be used for comparing an approximate result with the precise answer for quantifying the performance of algorithms, or for tuning their parameters before queries are run. A discussion of these and some others can be found most recently in [6].

Our problem setting differs, in that we propose to build a query evaluation system in which the quality metric will be something specified by the user, and correspondingly reported to the user at the time of query evaluation. Hence, there is no way of comparing the approximate result with the precise one, because the latter is never known. We will have to devise a notion of quality which depends on the approximate answer alone, specifying by "how much" this can deviate from the (unknown) exact answer.

Moreover, while complex measures of quality such as the ones enumerated above try to summarize the overall accuracy of a result in a single number, they do little to make the notion of quality intuitively understandable to users of the system. Remember, that users must specify their quality requirements, and they must do so using some metric which has a simple, and definite real-world meaning. In particular they should be able to specify both their need for set-based accuracy of the result, i.e., whether objects appear in the output or not, as well as for the accuracy of the values of the object that do appear (value-based accuracy).

2.1 Set-Based Accuracy: Precision and Recall

We have chosen to use two quality metrics for set-based accuracy: precision and recall, which are commonly used to gauge the quality of answer sets in Information Retrieval [3]. Precision is the fraction of objects in the output that are in the exact set. Recall is the fraction of objects in the exact set that are in the output. Both should be ideally 1.

The answer given in response to a QaQ will consist of a set of objects, noted $\mathcal{A}^{\lambda}_{\mathcal{T}}$. This will include both imprecise objects, as well as precise ones returned from probes. In the sequel, we will drop the $\lambda, \mathcal{T}$ from the notation with the undestanding that they are implied. We note the precision p and recall r of $\mathcal{A}$ as:

$$p = \begin{cases} \frac{|\mathcal{A} \cap \mathcal{E}|}{|\mathcal{A}|} & \text{if } |\mathcal{A}| \neq 0 \\ 1 & \text{if } |\mathcal{A}| = 0 \end{cases} \quad (3)$$

$$r = \begin{cases} \frac{|\mathcal{A} \cap \mathcal{E}|}{|\mathcal{E}|} & \text{if } |\mathcal{E}| \neq 0 \\ 1 & \text{if } |\mathcal{E}| = 0 \end{cases} \quad (4)$$

The definition of intersection in the above is $\mathcal{A} \cap \mathcal{E} = \{x | x \in \mathcal{A} \wedge (x \in \mathcal{E} \vee \omega^x \in \mathcal{E}\}$.[2] The answer ought to satisfy:

$$p \geq p^q \quad (5)$$

$$r \geq r^q \quad (6)$$

p^q, r^q are the precision, and recall *tolerances* or *requirements* of the query. There are situations where a query might request a higher level of recall, or even perfect recall ($r^q = 1$), e.g., for the query: "retrieve all sensors whose temperature value is above critical threshold T". In that case, we care primarily that *all* such sensors be retrieved, because to miss some might imply, e.g., a potential accident. We do not care so much that *only* such sensors be retrieved. In a different scenario, we are trying to find *some* patients from a medical database whose electro-cardiogram time series exhibits some pattern XYZ. This information will be used to select candidates for testing a new drug. We do not care to retrieve *all* patients in the world who present pattern XYZ, but the ones we do, must *definitely* do so. In this case, high or perfect precision ($p^q = 1$) is required.

2.2 Value-Based Accuracy: Laxity

The answer to the selection query will consist, as we have seen, of the set $\mathcal{A}$. To quantify the value-based quality of this set, we will use the *laxity* measure. This represents the amount of uncertainty that an imprecise object contains. The laxity of an object x is noted as $l(x)$. By convention $l(x) \in \mathbb{R}^+$ and $l(x) = 0$ implies that there is no uncertainty about the value of x. Thus, always $l(\omega^o) = 0$.

For example, if precise objects are real values and imprecise ones are intervals containing these, then laxity may be defined for interval $o = [l, h]$ as $l(o) = h - l$. The definition of laxity will depend on the type of object being represented, and the model of imprecision employed. For example, an alternative representation for a real value is via a density distribution. A parameter of that distribution (e.g., standard deviation for normal data) may be used as laxity.

The user will specify the value-based requirement as:

$$l^{max} = \begin{cases} max_{x \in \mathcal{A}} l(x) \leq l^{max}_q & \text{if } \mathcal{A} \neq \emptyset \\ 0 & \text{if } \mathcal{A} = \emptyset \end{cases} \quad (7)$$

This means that the maximum laxity l^{max}, over all objects in the answer set must be below a bound l^{max}_q. For example, we might ask: "retrieve the sensor IDs and temperatures (within $\pm 1^o C$) of all sensors whose temperature is above $30^o C$." In this case, using the $l(o) = h - l$ definition, we have a laxity requirement of $l^{max}_q = 1$.

Summary.— Our proposed metrics encompass two different aspects of quality: the set-based aspect pertaining to our uncertainty with respect to objects meeting the selection predicate or not, and the value-based aspect which involves the uncertainty of objects that appear in the answer. Our proposed metrics have a reasonably intuitive meaning, making it easy for users to specify them and to interpret results with such quality guarantees. However, they are not the only possible metrics that could be used. They suffice however for illustrating the feasibility of exploiting the performance-accuracy tradeoff in the given setting.

2.3 Achieving the Quality Guarantees

The QaQ operator does not know $\mathcal{E}$. Objects read from the input $\mathcal{T}$ are placed into two sets, $\mathcal{Y}, \mathcal{N}$ if they are classified as YES and NO respectively. Naturally, $\mathcal{N}$ does not need to be maintained; these objects are simply discarded. The remainder of $\mathcal{T}$ is $\mathcal{M} = \mathcal{T} - \mathcal{Y} - \mathcal{N}$ and consists of objects that remain as MAYBE. This consists both of a set of objects that the operator has not yet seen, $\mathcal{M}_{ns}$, which are MAYBE by virtue of not having been examined, as well as a set of objects $\mathcal{M}_s$ that the operator has seen, were MAYBE, and were not probed. Thus $\mathcal{M} = \mathcal{M}_s \cup \mathcal{M}_{ns}$.

The answer set $\mathcal{A}$ will consist of some of the objects of $\mathcal{Y} \cup \mathcal{M}_s$; that is $\mathcal{A} \subseteq \mathcal{Y} \cup \mathcal{M}_s$. It will be built progressively, by (conceptually) appending objects to $\mathcal{A}$, and (practically) piping them to the output and presenting them to the user. Query evaluation will end when $\mathcal{A}$ is such that all quality

[2] Under the usual definition of intersection $\mathcal{A} \cap \mathcal{E} = \{x | x \in \mathcal{A} \wedge x \in \mathcal{E}\}$, an imprecise object in $\mathcal{A}$ whose precise version is in $\mathcal{E}$ would not be in $\mathcal{A} \cap \mathcal{E}$. This is just a consequence of the fact that we have defined $\mathcal{E}$ to include only precise objects, hence the need for a slightly modified definition.

guarantees are met. Remember that for an on-line algorithm, we cannot touch an object once it has been output, nor can we defer handling it by placing it into temporary storage.

The quality guarantees, if the answer is $\mathcal{A}$ are:

$$p \geq p^G = \frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{A}|} \tag{8}$$

$$r \geq r^G = \frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{Y}| + |\mathcal{M}_{ns}| + |\mathcal{M}_s - \mathcal{A}|} \tag{9}$$

$$l^{max} = \max_{x \in \mathcal{A}} l(x) \tag{10}$$

For p, we observe that it ranges in the interval $[\frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{A}|}, 1]$ depending on how many output objects (in number ranging from $|\mathcal{A} \cap \mathcal{Y}|$ to $|\mathcal{A}|$) actually satisfy λ; hence the guarantee given in (8). For r, we observe that it ranges in the interval $[\frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{Y}|+|\mathcal{M}_{ns}|+|\mathcal{M}_s - \mathcal{A}|}, \frac{|\mathcal{A} \cap \mathcal{Y}|+|\mathcal{A} \cap \mathcal{M}_s|}{|\mathcal{Y}|+|\mathcal{A} \cap \mathcal{M}_s|}]$. We know that the exact set contains at least $|\mathcal{Y}|$ objects. In the worst case, which gives the lower bound (9), all the $|\mathcal{M}_{ns}|$ un-seen objects are YES, as are all the seen MAYBE objects that were not forwarded (ignored), $|\mathcal{M}_s - \mathcal{A}|$ in number. Note that we don't repeat the special cases (for denominator equal to 0) in the expressions (8), (9), (10). These are the same as in previously seen (3), (4), (7).

Example.— Suppose that $|\mathcal{T}|$=1000 and we have seen 200 objects. Hence, $|\mathcal{M}_{ns}| = 800$. Of the ones we have seen, 100 were YES, and 50 were MAYBE; the remaining 50 were NO. We forwarded 80 of the YES ones, ignoring the remaining 20. We probed 20 of the 50 MAYBE ones, and 10 returned YES, which we forwarded, and 10 returned NO. Of the remaining $|\mathcal{M}_s| = 30$ which were not probed, we forwarded 20 and ignored $|\mathcal{M}_s - \mathcal{A}| = 10$. Thus, the total number of YES objects that were seen were the 100 that were read as YES plus the 10 that became YES after a probe. Hence, in total $|\mathcal{Y}| = 110$. Of these we forwarded all except the 20 that we ignored; hence $|\mathcal{A} \cap \mathcal{Y}| = 90$. The total answer consists of these 90 plus the 20 MAYBE ones that we also forwarded. Hence, in total $|\mathcal{A}| = 110$.

Thus, we have an answer set of size $|\mathcal{A}| = 110$ of which only 90 (which are YES) are objects that are guaranteed to be in the exact set. Hence, precision is at least $p^G = \frac{90}{110} = 0.81$. Potentially all the 800 unseen objects could be YES, in addition to the 110 YES already seen and the 20 that were ignored. Hence the recall could be as low as $r^G = \frac{90}{110+800+20} = 0.097$.

```
(1)  procedure QaQSelection
(2)  INPUT: set $\mathcal{T} \neq \emptyset$, requirements $p_q, r_q, l_q^{max}$.
(3)  OUTPUT: set $\mathcal{A}$, guarantees $p^G, r^G, l^{max}$.
(4)
(5)  $\mathcal{A} \leftarrow \emptyset$;   (* Answer set *)
(6)  $|\mathcal{M}_{ns}| \leftarrow |\mathcal{T}|$;   (* Number of objects yet to be seen *)
(7)  $|\mathcal{Y}| \leftarrow 0$;   (* Number of YES objects seen *)
(8)  $|\mathcal{A} \cap \mathcal{Y}| \leftarrow 0$;   (* Number of YES objects output *)
(9)  $|\mathcal{M}_s - \mathcal{A}| \leftarrow 0$;   (* Number of ignored MAYBE objects *)
(10) do
(11)   $o \leftarrow \mathcal{T}.nextObject()$; $|\mathcal{M}_{ns}|$--;   (* Read object *)
(12)   if $\lambda(o) =$ YES decide (* Subject to Theorem 3.1 *)
(13)     (i): $\alpha \leftarrow \{o\}$; $|\mathcal{Y}|$++; $|\mathcal{A} \cap \mathcal{Y}|$++;   (* Forward a YES *)
(14)     (ii): $\alpha \leftarrow \{\omega^o\}$; $|\mathcal{Y}|$++; $|\mathcal{A} \cap \mathcal{Y}|$++;   (* Probe a YES *)
(15)     (iii): $\alpha \leftarrow \{\}$; $|\mathcal{Y}|$++;   (* Ignore a YES *)
(16)   else if $\lambda(o) =$ MAYBE decide (* Subject to Theorem 3.1 *)
(17)     (i): $\alpha \leftarrow \{o\}$; (* Forward a MAYBE *)
(18)     (ii): if $\lambda(\omega^o) =$ YES goto (14);   (* Probed MAYBE is YES *)
(19)           else goto (22);   (* Probed MAYBE is NO *)
(20)     (iii): $\alpha \leftarrow \{\}$; $|\mathcal{M}_s - \mathcal{A}|$++;   (* Ignore a MAYBE *)
(21)   else if $\lambda(o) =$ NO
(22)     $\alpha \leftarrow \{\}$;
(23)   end-if;
(24)   $\mathcal{A} \leftarrow \mathcal{A} \cup \alpha$;   (* Update the answer set *)
(25)   update $p^G, r^G, l^{max}$ from Equations 8,9,10;
(25) while $r^G < r_q$
```

Figure 1. Handling of Objects by the QaQ Selection Operator

3 QaQ Selection Operator

We will now show how the QaQ selection operator will be evaluated. Pseudo-code for the generic algorithm that achieves this is seen in Figure 1. Input consists of the set $\mathcal{T}$ and the quality requirements, expressed via the bounds p_q, r_q, l_q^{max}. Output consists of the answer set $\mathcal{A}$ as well as the bounds that the operator guarantees for it. These bounds p^G, r^G, l^{max} are such that $p^G \geq p_q \wedge r^G \geq r_q \wedge l^{max} \leq l_q^{max}$ must finally hold.

The algorithm assumes that the method used to access the input set $\mathcal{T}$ is a linear scan. In the presence of an index we can effectively prune away part of $\mathcal{T}$ implicitly, i.e., without actually accessing those objects and evaluating λ over them. For lack of space we will not address this problem in the present, whose main concern is to investigate the quality-performance tradeoff for set-based answers.

The operator first initializes the set counts that are needed to calculate the quality bounds. The answer set is initially empty (line 5), and none of the objects have yet been seen, hence $|\mathcal{M}_{ns}|$ is equal to $|\mathcal{T}|$ (line 6). No YES objects have been seen yet, hence $|\mathcal{Y}| = 0$ (line 7). Of course, so is $|\mathcal{Y} \cap \mathcal{A}|$ and $|\mathcal{M}_s - \mathcal{A}|$ (lines 8,9).

The algorithm in the **do** loop (line 10) reads one object o at a time and evaluates $\lambda(o)$, decrementing $|\mathcal{M}_{ns}|$ since one more object has now been "seen" by the operator. It then classifies o as YES (line 12), MAYBE (line 16), or NO (line 21), and handles it accordingly. Finally, $\mathcal{A}$ is updated by adding the set α to it which contains either the object o, its probed precise version ω^o, or nothing (line 24). Then, the quality guarantees are updated (line 25) and if the recall bound has still not been met, the loop continues. Remember that we can stop the algorithm when $p^G \geq p_q \wedge r^G \geq r_q \wedge l^{max} \leq l_q^{max}$ has been met. As we will see in the sequel $p^G \geq p_q \wedge l^{max} \leq l_q^{max}$ will always have to hold. Hence the stopping criteria is reduced to achieving the recall bound.

Let's see how objects are actually handled, beginning with the simplest case of a No object. In that case, nothing needs to be done; the object is rejected ($\alpha = \{\}$). The laxity and precision bounds are unaffected, since these depend only on the answer set. But, from (9) we see that the denominator of r^G is reduced by 1 since $|\mathcal{M}_{ns}|$ has been reduced. Hence the recall bound r^G improves.

The next case involves a Yes object. This can be forwarded as is, probed and ω^o forwarded, or discarded as in each of the options (i-iii) of lines 13-15. If it is forwarded (i), then $|\mathcal{Y}|$ and $|\mathcal{A} \cap \mathcal{Y}|$ are incremented by 1 and from (8), we see that p^G increases. r^G from (9) also increases: the denominator is the same as $|\mathcal{M}_{ns}|$ is decremented, while $|\mathcal{Y}|$ is incremented, but the nominator $|\mathcal{A} \cap \mathcal{Y}|$ is incremented. Finally l^{max} increases iff $l(o) > l^{max}$. If the Yes object is probed (ii), then l^{max} will be the same, since $l(\omega^o) = 0$. Precision and recall will be changed as in (i): probing a Yes object does not benefit set-based quality and makes sense only if an object o has laxity $l(o) > l_q^{max}$. Finally, a Yes object could be ignored (iii). Why would we ever do such a thing? Remember that a Yes object could have high laxity. Outputting it increases precision/recall, but it may violate laxity l_q^{max}. Hence, it might be advantageous to ignore it, waiting for some object o' which similarly increases precision/recall but with $l(o') \leq l_q^{max}$.

Finally, we deal with a Maybe object. This corresponds to the three cases (i-iii) of lines 17-20. If we forward (i) a Maybe object then the answer size $|\mathcal{A}|$ is incremented while $|\mathcal{A} \cap \mathcal{Y}|$ remains the same; hence p^G is reduced. r^G increases, since $|\mathcal{A} \cap \mathcal{Y}|$ remains the same, while $|\mathcal{M}_{ns}|$ in the denominator is decremented. We can easily prove that the increase in r^G produced by a Maybe object is less or equal to that produced by a Yes one. Laxity changes as in case (i) for Yes objects. If we probe (ii) a Maybe object then it becomes either Yes or No and handled as in those respective cases. Finally, if we ignore (iii) a Maybe object then precision and laxity remain unaffected, while recall also remains unaffected, since $|\mathcal{M}_{ns}|$ is decremented while the number of ignored objects $|\mathcal{M}_s - \mathcal{A}|$ is incremented.

We summarize these findings in Table 3. We note that exercizing some of the options (i-iii) for Yes and Maybe objects may not be feasible, since this may potentially cause the algorithm to reach a situation where no matter how it handles all the remaining objects from $\mathcal{M}_{ns}$ it will never be able to produce a final answer $\mathcal{A}$ which meets the QaQ requirements. Thus at a given point in the operator's evaluation, an option can be exercized under certain conditions which the following theorem enumerates. The theorem applies to all online algorithms for this problem with the same set of options of handling objects.

$\lambda(o)$	Action	p^G	r^G	l^{max}
No	Ignore	=	+	=
Yes	Forward	+	+	+/=
	Probe	+	+	=
	Ignore	=	=	=
Maybe	Forward	-	+	+/=
	Probe, $\lambda(\omega^o) =$ Yes	+	+	=
	Probe, $\lambda(\omega^o) =$ No	=	+	=
	Ignore	=	=	=

Table 1. How p^G, r^G, l^{max} change depending on the input object type and the QaQ operator's decision. +: increase, -: decrease, =: unaffected

Theorem 3.1 *Given $|\mathcal{A}|$, $|\mathcal{A} \cap \mathcal{Y}|$, $|\mathcal{Y}|$, $|\mathcal{M}_s - \mathcal{A}|$, l^{max} and a read object o: $\lambda(o) \neq$ No: (a) if $l(o) > l_q^{max}$, or (b) $\lambda(o) =$ Maybe and $\frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{A}|+1} < p_q$ then object o cannot be added to the answer set $\mathcal{A}$. If (c) $\frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{Y}|+|\mathcal{M}_s - \mathcal{A}|} < r_q$, then o cannot be ignored.*

Proof: (a) If $l(o) > l_q^{max}$ and o is forwarded then l^{max} will be updated to $l^{max} > l_q^{max}$ which violates the l_q^{max} requirement. l^{max} as defined in (10) can never decrease as more objects are added to $\mathcal{A}$. Hence, the l_q^{max} requirement will eventually be violated.

(b) If o is forwarded then the new precision guarantee p^G will be $\frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{A}|+1}$, from (8). But this means that $p^G < p_q$ according to the hypothesis. p^G gets updated only if new objects are added to $\mathcal{A}$. But since all the remaining objects may be No, the p_q requirement may be violated.

(c) If o is ignored, then r^G will remain the same, as in (9) and also see Table 3. From Table 3 we also see that no matter what kinds of objects are seen in the future, the operator can always take some action that increases r^G: it can forward all Yes objects and probe all Maybe ones, thus making sure that no objects in the remaining $\mathcal{M}_{ns}$ are missed. But, suppose that all remaining objects are No. In that case, $|\mathcal{M}_{ns}|$ will finally be reduced to 0 and the final r^G will be equal to $\frac{|\mathcal{A} \cap \mathcal{Y}|}{|\mathcal{Y}|+|\mathcal{M}_s - \mathcal{A}|}$. Thus from the hypothesis, finally $r^G < r_q$ and hence r_q may eventually be violated. ∎

The algorithm shown in Figure 1 performs some operations which have associated costs to them. In the following section we will examine the costs of these various operations. Subsequently, we will see how making the decisions of how to handle objects can affect the overall cost of evaluation, given a set of quality requirements.

Symbol	Description
c_r	Cost of reading *o* and evaluating $\lambda(o)$
c_p	Cost of probing *o* and evaluating $\lambda(\omega^o)$
c_{wi}	Cost of adding an imprecise object *o* to $\mathcal{A}$
c_{wp}	Cost of adding a precise object ω^o to $\mathcal{A}$
R	# of objects read
Y	# of objects read that were YES
M	# of objects read that were MAYBE
Y_f, Y_p	# of YES objects that were forwarded, probed
M_f, M_p	# of MAYBE objects that were forwarded, probed
M_{py}	# of probed MAYBE objects that became YES

Table 2. Cost Model

3.1 Cost Model

The parameters of the cost model are summarized in Table 2. We can split the QaQ's evaluation cost into:

- **Read Cost**.— Since R objects are read at a cost of c_r each, this is simply Rc_r. Note, that not all $|\mathcal{T}|$ objects need always be read. Sometimes, if r_q is low, we may only have to examine part of the input.
- **Probe Cost**.— Since $Y_p + M_p$ objects are probed at a cost of c_p each, this is $(Y_p + M_p)c_p$. Probing an object is usually more expensive than reading or writing it.
- **Write Cost**.— The output consists of the Y_f, M_f imprecise objects that were forwarded at a cost of c_{wi} each, plus the Y_p YES objects that were probed, and M_{py} MAYBE ones that were probed and returned YES, for a cost of c_{wp} each. In total, $(Y_f + M_f)c_{wi} + (Y_p + M_{py})c_{wp}$.

The full cost W of evaluation can be written:

$$W = Rc_r + (Y_p + M_p)c_p + (Y_f + M_f)c_{wi} + (Y_p + M_{py})c_{wp} \tag{11}$$

The goal is to minimize W, given $\mathcal{T}$ and p_q, r_q, l_q^{max}.

3.2 Handling Objects

Now we will talk about how to perform the crucial step in the QaQ selection algorithm of Figure 1 of deciding how an object should be handled, and why this decision is important. We will first show why all possible ways of handling a YES or MAYBE object may actually be useful, by means of some extreme examples.

Suppose that it is required $p_q = 1$, $r_q = 1$, i.e., perfect set-based quality. It is clear that we have no choice but to probe each MAYBE object from the input. If we forward such an object then the answer won't be precise. If we ignore it then the answer won't be complete. Suppose that additionally $l_q^{max} = 10$, but for all YES objects in the input it is $l(o) > l_q^{max}$. If we forward these then l_q^{max} will be violated. If we ignore them, r_q will be violated. Hence, again we have to probe them. Thus, we establish that both MAYBE and YES objects sometimes have to be probed.

Now, suppose that we only require $r_q = 1$, i.e., perfect recall. In this case, we have to exhaust $\mathcal{T}$ and forward all YES and MAYBE objects we encounter. If we ignore one, then r_q is violated. If we probe one, then we are paying the extra cost of c_p unnecessarily, since that gives no help in reaching the goal of $r^G = r_q = 1$: all objects of $\mathcal{T}$ must be seen. Thus, in this case, the smartest strategy is to just forward objects, because ignoring them is a non-option (violating r_q) and probing them gives no benefit.

Finally, suppose that we require $p_q = 1$ and $r_q = 0.02$ and $l_q^{max} < 1$, i.e., we want to get only a few of the objects that satisfy the query, but we want these to be fairly precise. Now, suppose that at some stage in the evaluation of the algorithm it is $|\mathcal{Y}| = |\mathcal{A} \cap \mathcal{Y}| = 1$, $|\mathcal{M}_{ns}| = 99$, and $|\mathcal{M}_s - \mathcal{A}| = 0$. From (9) we have that $r^G = \frac{1}{100} = 0.01$. From Theorem 3.1 we see that if we ignore an object then r^G can be improved up to $\frac{1}{2}$. Hence, ignoring an object is a feasible option. Now, suppose that either a YES object with $l(o) > 1$ or a MAYBE object is read. If we probe the YES one, then r^G will increase to 0.02 and hence the QaQ may end. We can't forward it because of the laxity constraint. Why would we choose to ignore it instead of probing it? Suppose that there is another YES object with $l(o) < 1$ further down the input. If we wait until we encounter that, then r_q can be met without doing *any* probes.

If a MAYBE object is read, then again we cannot forward it because of the precision constraint. We can ignore it, using Theorem 3.1 again. If we probe it, it may return YES in which case r_q is again met. But, for the same reason as before, we may decide to ignore it, since there is a chance that we will meet a YES object in the remaining input that will make the probe unnecessary.

In summary, there are cases for all possible decisions for handling objects listed in the algorithm of Figure 1 in which they are either mandatory or preferred. Therefore, we must deal with all of them.

4 Optimization Framework

In this section we will first present, in Section 4.1 a way of visualizing objects that allows us to distinguish between objects that are to be handled in different ways. This will result in a set of parameters based on which the decision will be made at run-time. Subsequently, in Section 4.2 we will discuss how to set these parameters to optimal values in order to minimize the cost of evaluation W.

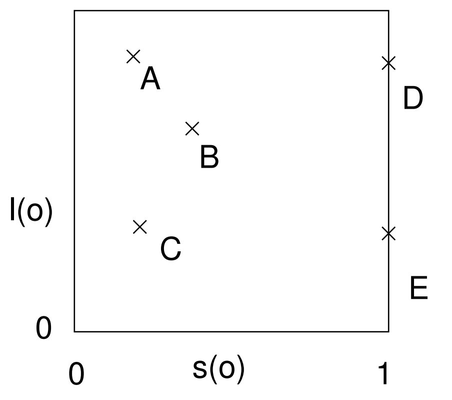

Figure 2. Objects on the $s(o), l(o)$ plane

4.1 The $s(o), l(o)$ Plane

To develop our optimization framework we will assume that we know the probability that a given MAYBE object o will return a YES after a probe. This *probability of success* is noted $s(o)$. It might be possible to estimate such a probability using some model of imprecision for ω^o. For example, suppose that $o = [l, h]$ represents some $\omega^o \in [l, h]$. If the query asks for all objects with value $\geq x$, then assuming that ω^o is a random variable uniformly distributed in $o = [l, h]$, then the probability that o will return YES is $s(o) = \frac{h-x}{h-l}$. If $s(o)$ cannot be estimated, we can set it to some value based on our prior belief, e.g., that $s(o) = 0.5$, meaning that a probe will return YES with the same probability that it will return NO.

We possess the laxity $l(o)$ for each object o. We can thus plot o as a point in the $s(o), l(o)$ plane. For YES objects, we can define $s(o) = 1$. In Figure 2 you see five objects: A, B, C are MAYBE and D, E are YES. D has a greater laxity than E. B has a higher probability of success than C. The entire set of YES and MAYBE objects that the QaQ operator has to deal with will be such a set of points in the plane. Therefore, the decision of how to handle them can be reduced to identifying regions in this plane that ought to be handled in some manner, e.g., probed. Then, when the algorithm reads an object, it will determine which region it belongs to and handle it accordingly. Remember that this decision must conform to Theorem 3.1 which takes precedence in order to assure correctness.

NO objects.— As we have already seen, NO objects, corresponding to region 1 of Figure 3 are ignored.

YES objects.— First, if a YES object o has $l(o) > l_q^{max}$ then it can never be forwarded. This corresponds to region 6 of Figure 3. Thus, it will either be probed or ignored. When such an object is encountered, it is probed with probability p_{py} and ignored with probability $1 - p_{py}$. Second, if a YES object o has $l(o) \leq l_q^{max}$ then it should never be probed.

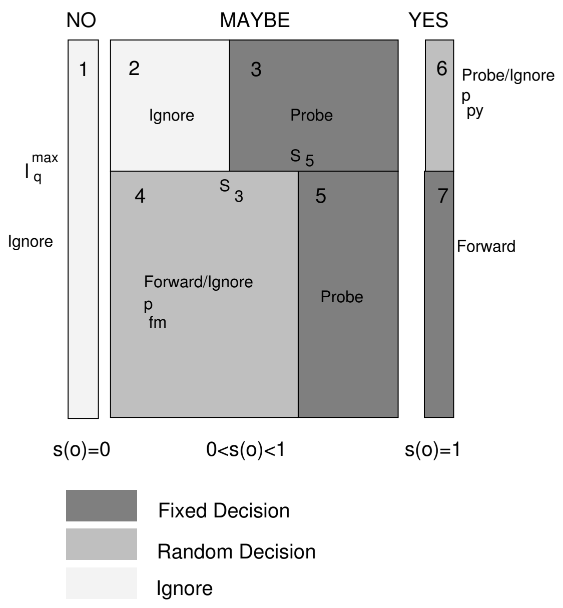

Figure 3. Handling of Objects

That corresponds to regions 7. Probing such an object is redundant, since it meets the laxity requirement, and its status as an answer to the query is YES. So, such an object must be forwarded; ignoring it has no benefit, unlike MAYBE objects, where ignoring them maintains the precision of the answer set.

MAYBE objects.— First, as with YES objects, we may never forward them if $l(o) > l_q^{max}$. This occurs in regions 2, 3. We may either probe or ignore them. Since probing objects with higher $s(o)$ is preferred, as these have a higher probability of returning YES, which increases the recall bound most, it will be the case that objects with highest $s(o)$ will be probed. Hence, the algorithm will probe objects with $s(o) > s_3$, where s_3 is a probing threshold, and it will ignore the rest. Second, if $l(o) < l_q^{max}$ we can either forward, probe or ignore them. For the same reason, we probe objects with highest $s(o)$, and thus decide to probe if $s(o) > s_5$. The rest, we can either ignore or forward. We forward them with probability p_{fm} and ignore them with probability $1 - p_{fm}$.

4.2 Optimal Parameter Setting

In the previous section, we identified the "topology", so to speak, of the decision regions that control how different objects are handled. These regions are determined by the status of an object as NO,YES, or MAYBE, the laxity tolerance l_q^{max}, and the parameters s_3, s_5, p_{py}, p_{fm}.

Now, we will see how *optimal* values for these parameters can be set. We notice that these will depend on (a) the *quality tolerances* of the QaQ: for example, if $r_q = 1$, we expect region 2 to be empty, and p_{fm}, the probability of forwarding a MAYBE object of region 4 to be 1, since ignoring a MAYBE object is not allowed, and (b) the *distribution of objects* on the $s(o), l(o)$ plane, since if we fix these parameters and change the distribution of objects, then the number of objects falling in each region of the plane will change.

There are numerous ways in which we may obtain an estimate of the density distribution of objects on the $s(o), l(o)$ plane, which we will note $g(s(o), l(o))$. For example, we may take a random sample of $\mathcal{T}$ prior to query evaluation and plot the objects of the sample on the plane. It is also conceivable that there might be a histogram of laxity values for the set $\mathcal{T}$ which can be used to estimate the marginal distribution $g(l(o))$. We will develop the parameter setting under the simple hypothesis that $g(s(o), l(o)) = \frac{1}{L}$, where $L = \max_{o \in \mathcal{T}} l(o)$ is the maximum laxity of objects in the input set. This uniformity assumption could be replaced with a more precise one that could be generated by the methods outlined above.

First, we note that the QaQ operator will encounter Y, M objects that are respectively YES and MAYBE. In region 6, there will be $\frac{L - l_q^{max}}{L} Y$ objects and in region 7, there will be $\frac{l_q^{max}}{L} Y$ objects. From region 6, we will probe $P_6 = p_{py} \frac{L - l_q^{max}}{L} Y$ objects. From region 7, we will forward $F_7 = \frac{l_q^{max}}{L} Y$ objects. In region 3, we will probe $P_3 = \frac{L - l_q^{max}}{L}(1 - s_3)M$ objects. In region 5, we will probe $P_5 = \frac{l_q^{max}}{L}(1 - s_5)M$ objects. In region 4, we will forward $F_4 = p_{fm} \frac{l_q^{max}}{L} s_5 M$ objects.

Thus, $Y_p = P_6$ and $Y_f = F_7$ for YES objects, and $M_p = P_3 + P_5$ and $M_f = F_4$ for MAYBE ones. Finally, let's see what is the expected result of the probes of regions 3, 5. In region 3, the average probe has a probability of success that is $\frac{s_3+1}{2}$, thus $M_{py3} = \frac{s_3+1}{2} P_3$ objects will return YES. In region 5, using the same calculation, it will be $M_{py5} = \frac{s_5+1}{2} P_5$. In total:

$$M_{py} = M_{py3} + M_{py5} \qquad (12)$$

4.2.1 Selectivity Estimation

Before we can finally formalize the optimization problem that we will be solving, we need one extra step, namely to estimate the proportion of YES and MAYBE objects that we are likely to encounter. This can be discovered via random sampling prior to query execution. By taking a random sample of $\mathcal{T}$ we measure the fraction of objects f_y that are YES and the fraction f_m that are MAYBE. Subsequently, we can estimate, that if we read R objects, it will be:

$$M = f_m R \qquad (13)$$

$$Y = f_y R \qquad (14)$$

4.2.2 The Full Optimization Problem

The goal is now to minimize W from Equation (11) under a set of constraints.

1. Trivial constraints of positivity of all parameters.
2. $s_3 \leq 1$ and $s_5 \leq 1$. $p_{py} \leq 1$ and $p_{fm} \leq 1$.
3. We can read at most $|\mathcal{T}|$ objects. Hence, $R \leq |\mathcal{T}|$.
4. The number of YES and MAYBE objects (Y, M) are given in Equations (13), (14).
5. We can probe/forward at most M MAYBE objects. Hence, $M_p + M_f \leq M$.
6. We can probe/forward at most Y YES objects. Hence, $Y_p + Y_f \leq Y$.
7. The number of probes on MAYBE objects that return YES is M_{py}, see (12).
8. The result meets the precision constraint:
$$\frac{Y_p + Y_f + M_{py}}{Y_p + Y_f + M_{py} + M_f} \geq p_q \qquad (15)$$
9. The result meets the recall constraint:
$$\frac{Y_p + Y_f + M_{py}}{Y + M_{py} + |\mathcal{T}| - R + M - M_p - M_f} \geq r_q \qquad (16)$$

There are four free parameters to this optimization problem: s_3, s_5, p_{py}, p_{fm}. By solving this problem, their values that minimize W are identified. This instantiates the generic decision step in the QaQ algorithm. The non-linear optimization problem can be solved efficiently due to the small number of free parameters. In our modest 500MHz Pentium PC, the solution was found instantaneously, with < 100 iterations and evaluations of the objective function.

5 Performance Study

Our experiments consist of two parts. First, we implemented the optimization problem in AMPL [17], using the LOQO non-linear solver package [1], finding the optimal solution for various characterizations of the input and quality requirements. This helped build our intuition on the way parameters change under different settings. Subsequently, we implemented the QaQ selection operator and applied it on synthetic data that corresponded to the input characterizations previously seen. We tried to see how closely actual performance matched the theoretical optimum.

Since we introduced this framework, there are no published algorithms for the problem at hand. We compared the performance of our algorithm against two heuristics:

- **Stingy**.— The Stingy heuristic avoids paying any costs. Thus, it avoids probing any objects that exceed the laxity requirement l_q^{max} and ignores all MAYBE objects. In essence, it tries to answer the query using only the objects of region 7. However, due to Theorem 3.1, if these do not suffice, it will have to perform some probes. Using our set of parameters, Stingy always uses $s_3 = s_5 = 1$ and $p_{py} = 0, p_{fm} = 0$.

- **Greedy**.— The Greedy heuristic attempts to complete evaluation of the query as soon as possible. It thus tries to reach the recall goal by probing all YES objects of region 6, as well as all the objects of region 3. It ignores no objects, hence region 2 is non-existent. It forwards all MAYBE objects that meet the laxity bound. Greedy always uses $s_3 = 0, s_5 = 1$ and $p_{py} = 1, p_{fm} = 1$.

In all our experiments the probe cost c_p is $100\times$ the read/write costs $c_r = c_{wi} = c_{wp} = 1$. [3] Results with different cost parameters did not produce any surprising results: as long as probing (measured by c_p) is more expensive, optimal parameter settings are unaffected, although the resulting cost, which is linear in c_p is. Thus, including results with varying c_p is redundant and for lack of space we omit them. Also, note that the cost is also linear in $|\mathcal{T}|$. The framework determines the fraction of objects that are handled in different ways, but the actual number of these objects is proportional to $|\mathcal{T}|$ as is the cost from (11). We thus present the optimal solution costs normalized by dividing with $|\mathcal{T}|$, or $\frac{W}{|\mathcal{T}|}$. Finally, in all our experiments the maximum laxity of objects in $\mathcal{T}$ is taken to be $L = 100$.

5.1 Optimal Problem Solutions

We now report the optimal parameters for s_3, s_5, p_{py}, p_{fm} for various characterizations of the input (f_y, f_m) and quality (p_q, r_q, l_q^{max}) requirements, and try to build an intuitive understanding of how these vary.

Varying Laxity

In this and the next two experiments, we set $f_y = f_m = 0.2$, that is, the number of YES and MAYBE objects at the input are the same. We set $p_q = 0.9$ and $r_q = 0.5$, that is, we want at least half the objects that satisfy the query and we can tolerate at most 10% false positives. We investigate how parameters change as the laxity bound changes $l_q^{max} \in \{1, 20, 40, 60, 80, 99\}$

[3]Two orders of magnitude is a good approximation for the difference in latency between DRAM and disk [16], as well as between disk and network transfer. The difference may be even higher for low-bandwidth sensors distributed in wide geographical areas.

l_q^{max}	s_3	s_5	p_{py}	p_{fm}	$\frac{W}{\lvert\mathcal{T}\rvert}$
1	1	1	1	1	20.9
20	1	1	0.93	0.53	16.2
40	1	1	0.91	0.26	12.2
60	1	1	0.87	0.18	8.2
80	1	1	0.74	0.13	4.2
99	1	1	0	0.11	1.2

The key observation is the following: as the laxity bound increases, there is a greater availability of objects that can be forwarded without a probe to meet the recall requirement. Hence, the fraction of MAYBE objects that needs to be forwarded p_{fm} and the fraction of YES ones that needs to be probed p_{py} is reduced. $s_3 = s_5 = 1$ means that no probes of MAYBE objects are needed for this setup. The cost is (as expected) diminishing as the laxity bound becomes looser.

Varying Precision

We set $r_q = 0.5$ and $l_q^{max} = 50$ and vary the precision bound in $p_q \in \{0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$.

p_q	s_3	s_5	p_{py}	p_{fm}	$\frac{W}{\lvert\mathcal{T}\rvert}$
0.5	1	1	0.5	1	6.3
0.6	1	1	0.5	1	6.3
0.7	1	1	0.65	0.71	7.7
0.8	1	1	0.78	0.44	9.0
0.9	1	1	0.89	0.21	10.2
0.99	1	1	0.99	0.02	11.1

The first two parameters $s_3 = s_5 = 1$ signify that at this recall level, it is unnecessary to probe any MAYBE objects. The cost is increasing (as expected) as the precision bound becomes stricter. The most interesting feature is that we probe more YES objects (p_{py} increases) and forward less MAYBE ones (p_{fm} decreases) as more precision is required.

Varying Recall

In our final experiment of varying the user quality requirements, we maintain $p_q = 0.9$ and $l_q^{max} = 50$ and vary $r_q \in \{0.01, 0.1, 0.2, 0.4, 0.6, 0.8, 0.99\}$. We record one extra feature here, the fraction of read objects $\frac{R}{|\mathcal{T}|}$.

r_q	s_3	s_5	p_{py}	p_{fm}	$\frac{W}{\lvert\mathcal{T}\rvert}$	$\frac{R}{\lvert\mathcal{T}\rvert}$
0.01	1	1	0	0	0.1	0.09
0.1	1	1	0	0	0.69	0.63
0.2	1	1	0	0.08	1	0.9
0.4	1	1	0.53	0.17	6.5	1
0.6	0.87	0.87	1	0.29	13.8	1
0.8	0.5	0.5	1	0.61	21.4	1
0.99	0.03	0.33	1	1	27.8	1

Note how for small r_q it is not necessary to read the entire input, e.g., we get away with reading just 9% of the input for $r_q = 0.01$. Naturally, the cost of evaluation increases as r_q increases. Both probing (p_{py}) and forwarding (p_{fm}) become more important at the expense of ignoring objects: objects can be ignored less frequently. An interesting feature is that at the highest $r_q = 0.99$ we note that s_3, s_5 are different. This is because ignoring objects (region 2) is almost impossible, hence $s_3 = 0.03$, while forwarding them is still possible, hence $s_t = 0.33$, since the precision requirement $p_q = 0.9$ can be maintained.

Varying Selectivity

In the next two experiments, we keep $p_q = 0.9, r_q = 0.5, l_q^{max} = 50$. We vary the query selectivity $(fy, fm) \in \{(0.01, 0.01), (0.1, 0.1), (0.2, 0.2), (0.4, 0.4)\}$.[4]

(f_y, f_m)	s_3	s_5	p_{py}	p_{fm}	$\frac{W}{\lvert\mathcal{T}\rvert}$
(0.01, 0.01)	1	1	0.89	0.21	1.5
(0.1, 0.1)	1	1	0.89	0.21	5.6
(0.2, 0.2)	1	1	0.89	0.21	10.2
(0.4, 0.4)	1	1	0.89	0.21	19.3

The cost increases, as the answer set of the query increases, but the parameter choices remain unaffected.

Varying Input Uncertainty

Finally, we keep $f_y = 0.2$ and vary $f_m \in \{0.01, 0.1, 0.2, 0.4, 0.6\}$, i.e., we make the input progressively more "uncertain" with respect to the predicate λ.

f_m	s_3	s_5	p_{py}	p_{fm}	$\frac{W}{\lvert\mathcal{T}\rvert}$
0.01	1	1	0.02	1	1.4
0.1	1	1	0.42	0.32	5.4
0.2	1	1	0.89	0.21	10.2
0.4	0.78	0.78	1	0.2	20.3
0.6	0.67	0.67	1	0.2	40.0

The result is that we are forced to probe more and ignore less as the input becomes more uncertain. Before we conclude, we must interpret the meaning of the optimum costs presented so far. These do not indicate a lower bound on the cost that an algorithm can achieve. An algorithm can in practice get "lucky" and achieve even better performance, e.g., if its probes are successful, or if the input is ordered in a way that YES objects are encountered in advance of MAYBE and NO ones.

[4] $f_y + f_m$ has to be less than 1

5.2 QaQ Trial Runs

In the second section, we report actual experimental runs of the QaQ operator for the sets of input/quality requirement characterizations examined previously. We generate $|\mathcal{T}| = 10000$ objects, which are labeled YES, MAYBE and NO with probability $f_y, f_m, 1 - f_y - f_m$. For the MAYBE ones, we assign a uniformly drawn random $s(o) \sim U(0, 1)$; then, we assign a "probe result" for each MAYBE object: with probability $s(o)$ this is YES and $1 - s(o)$ it is NO. We assign a uniformly drawn random $l(o) \sim U(0, 100)$ to all objects. Finally, we note that we can't use the (unknown) data generating f_y, f_m parameters to estimate the parameters s_3, s_5, p_{py}, p_{fm} for the QaQ algorithm. These are estimated from a random sample of size 1%.

Varying Laxity

The results are shown below. They differ surprisingly little from the theoretical optimal cost estimates produced by our optimizer. QaQ performs the best for all laxity values, followed by Stingy, which benefits from the fact that in this case it is actually optimum to avoid probing MAYBE objects, which is Stingy's default policy. Greedy, on the other hand overprobes, and fares much worse.

l_q^{max}	1	20	40	60	80	99
QaQ	20.7	16.3	12.3	8.5	4.3	1.3
Stingy	23.3	18.3	13.9	9.7	4.6	1.3
Greedy	31.1	25.7	19.9	14.0	7.6	1.5

Varying Precision

We observe the significant tradeoff that QaQ achieves in contrast to both Stingy and Greedy whose evaluation cost tends to be more constant. This occurs because Stingy reaches the recall bound by using YES objects only; this results in the answer set being overly pure. QaQ on the other hand, allows more MAYBE objects to be forwarded and adapts the precision guarantee close to the required bound. Greedy again fares the worse, for the same reason. Its policy cannot adapt to the changing p_q requirement.

p_q	0.5	0.6	0.7	0.8	0.9	0.99
QaQ	6.3	6.3	8.0	9.2	10.2	11.3
Stingy	10.0	10.0	10.0	10.3	11.8	13.0
Greedy	16.7	16.7	16.7	16.7	16.7	16.7

Varying Recall

This experiment shows how Greedy wastes effort when the recall requirement is low. In that case, probing MAYBE objects is beneficial, since the YES ones suffice. In contrast, it comes into its own when the recall requirement is high;

in that case, its policy of trying to increase r_q the fastest actually performs better than the alternative strategies. But, across the spectrum, QaQ performs more consistently well.

r_q	0.01	0.1	0.2	0.4	0.6	0.8	0.99
QaQ	0.1	0.7	1	6.7	15.4	21.7	27.5
Stingy	0.1	0.7	1	7.6	15.5	22.1	27.5
Greedy	0.9	6.6	10.5	15.3	18.0	19.9	24.3

Varying Selectivity

Again QaQ performs the best, followed by Stingy and then Greedy. As noted previously, changing selectivity does not alter optimum parameter values; the main effect is in presenting to the operator a larger set of object candidates.

(f_y, f_m)	(0.01,0.01)	(0.1,0.1)	(0.2,0.2)	(0.4,0.4)
QaQ	1.5	6.1	10.6	19.5
Stingy	1.6	6.9	12.1	22.7
Greedy	1.9	10.5	17.9	27.4

Varying Input Uncertainty

The most interesting feature again is the performance of Greedy which starts off poorly for very precise inputs, since it tends to probe unnecessarily in such inputs where the YES objects suffice, but Greedy becomes better as the input becomes more uncertain, in which its aggressive policy tends to build the recall bound quickly.

f_m	0.01	0.1	0.2	0.4	0.6
QaQ	1.5	5.7	10.8	22.1	35.6
Stingy	1.6	5.7	12.2	23.8	37.4
Greedy	9.8	13.5	17.5	23.9	32.8

6 Related Work

Previous work in approximate query processing has been extensive in the database community [20, 8, 11, 9, 4, 2]. Much of this involves "best effort" approximation which operates over summaries of data [20, 9, 4, 2]. Thus, answers cannot be improved and no guarantees as to their quality are given. These approaches can be shown to work well for some datasets, but one should exercize caution when applying them to new types of data. [7] proposes probabilistic wavelet-based synopses which do provide answer accuracy guarantees for individual queries, although not to any degree of (user-specified) accuracy. In contrast, [8, 11] can improve answer results interactively, all the way to the exact answer. [8] uses random sampling and provides statistical guarantees of accuracy. Our work in this paper is intermediate in scope, since it provides *any* level of desired quality but does not support progressive answer refinement.

Our main motivation was in continuing our research [12] on exploiting the accuracy/performance tradeoff in the sensor database domain. In the architecture proposed in [12] approximate versions of time series are captured in the database system. In the present, we show how approximate values can be queried in a database, although our setting is more general than the real-valued examples treated there. The approximate replication framework was earlier seen in a simpler setting by [15, 10]. These papers considered mainly aggregation queries over numerical values, taking into account only the cost of probing, i.e., retrieving precise versions of imprecise values. [10] also considered the problem of selection for an object of a given rank.

Imprecise object representation in databases has been studied in the database literature. [14] proposed a model of imprecise and uncertain information, and recognized how this affects the ability to evaluate queries. In [14], measures of gauging the degree of uncertainty of the set of objects that may satisfy the query were presented, with the goal of defining an order in which these are presented to the user.

[13] proposed an extension of the relational model to represent indefinite and "maybe" information in the relational model. The concept of an I-table which is able to represent such information is introduced and a relational algebra over such tables is presented. That work is not applicable in our case, since it does not deal with the performance/accuracy tradeoff which can be effected when "maybe" information can be resolved via probe operations.

More recently, the problem of providing approximate results has been examined in the context of joins over data streams [6]. Unlike streams in which the performance issue arises from the inability to handle incoming (precise) tuples in real time, thus motivating *load shedding*, in our case we deal with traditional stored relations, with approximate results being provided due to the imprecise representation of objects of these relations at the query processing site.

A metric recently proposed in [5] for gauging the quality of non-aggregate queries uses the observation that a good result is one for which we are near certain either that it satisfies the query predicate or not. Similarly, a bad result is a most ambiguous one. The metric proposed in [5], which corresponds to $\frac{|s(o)-0.5|}{0.5}$ in our notation, attains its maximum for objects that are either almost certainly YES ($s(o) \rightarrow 1$) or NO ($s(o) \rightarrow 0$). This metric could potentially be incorporated in our optimization framework.

7 Conclusions and Future Work

In this paper we motivated the problem of answering selection queries approximately over imprecise data. We gave examples of situations were imprecise representation of data objects is meaningful, and showed that if queries are willing to tolerate some loss of accuracy in the results they obtain from the system, then it is possible to drastically reduce the cost of query evaluation. Our work used the precision/recall and laxity metrics as the measure of answer accuracy, although conceivably other metrics could have been used as well. Our main purpose was to show that (i) the accuracy/performance tradeoff could be achieved for set-based queries, and (ii) that the combined cost of data processing, and probing operations should be considered when selecting a way to evaluate the selection operator.

There are still many open directions of research. Extending this work to account for other relational operators, in particular joins, is part of our current work. We are also thinking about integrating the optimization framework with different ways of accessing the object collection (e.g., indexes); this will make it immediately applicable to many situations where such access methods are available.

Acknowledgements

Our work was supported by the NSF (Awards IIS-9996140, IIS-0086124, CCR-0220069, IIS-0083489) and by the USAF (Award F33615-01-C-1902).

References

[1] R.J. Vanderbrei, Princeton University, LOQO. http://www.orfe.princeton.edu/∼loqo/.

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD Conference*, 1999.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley.

[4] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB Conference*, 2000.

[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD Conference*, 2003.

[6] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD Conference*, 2003.

[7] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *SIGMOD Conference*, 2002.

[8] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *SIGMOD Conference*, 1997.

[9] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB Conference*, 1999.

[10] S. Khanna and W. C. Tan. On computing functions with uncertainty. In *PODS*, 2001.

[11] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD Conference*, 2001.

[12] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *ICDE Conference*, 2003.

[13] K.-C. Liu and R. Suderraman. Indefinite and maybe information in relational databases. *TODS*, 15(1), 1990.

[14] J. Morrissey. Imprecise information and uncertainty in information systems. *Transactions on Information Systems*, 8(2), 1990.

[15] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB Conference*, pages 144–155. Morgan Kaufmann, 2000.

[16] R. Rangaswami, Z. Dimitrijevic, E. Chang, and K. E. Schauser. Mems-based disk buffer for streaming media servers. In *ICDE Conference*, 2003.

[17] D. M. G. Robert Fourer and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming, 2nd ed.* Duxbury Press / Brooks/Cole Publishing Company, 2002.

[18] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD Conference*, 1995.

[19] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2), 2000.

[20] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD Conference*, 1999.