Combining Fuzzy Information from Multiple Systems

(Extended Abstract)

Ronald Fagin
IBM Almaden Research Center
650 Harry Road
San Jose, California 95120-6099
email: fagin@almaden.ibm.com

Abstract

In a traditional database system, the result of a query is a set of values (those values that satisfy the query). In other data servers, such as a system with queries based on image content, or many text retrieval systems, the result of a query is a sorted list. For example, in the case of a system with queries based on image content, the query might ask for objects that are a particular shade of red, and the result of the query would be a sorted list of objects in the database, sorted by how well the color of the object matches that given in the query. A multimedia system must somehow synthesize both types of queries (those whose result is a set, and those whose result is a sorted list) in a consistent manner. In this paper we discuss the solution adopted by Garlic, a multimedia information system being developed at the IBM Almaden Research Center. This solution is based on "graded" (or "fuzzy") sets.

Issues of efficient query evaluation in a multimedia system are very different from those in a traditional database system. This is because the multimedia system receives answers to subqueries from various subsystems, which can be accessed only in limited ways. For the important class of queries that are conjunctions of atomic queries (where each atomic query might be evaluated by a different subsystem), the naive algorithm must retrieve a number of elements that is linear in the database size. By contrast, here an algorithm is given, which has been implemented in Garlic, such that if the conjuncts are independent, then with arbitrarily high probability, the total number of elements retrieved in evaluating the query is sublinear in the database size (in the case of two conjuncts, it is of the order of the square root of the size of the database). It is also shown that for such queries, the algorithm is optimal. The matching upper and lower bounds are robust, in the sense that they hold under almost any reasonable rule (including the standard min rule of fuzzy logic) for evaluating the conjunction. Finally, we find a query that is provably hard, in the sense that the naive linear algorithm is essentially optimal.

PODS '96, Montreal Quebec Canada
© 1996 ACM 0-89791-781-2/96/06..$3.50

1 Introduction

Garlic [CHS+95, CHN+95] is a multimedia information system being developed at the IBM Almaden Research Center. It is designed to be capable of integrating data that resides in different database systems as well as a variety of non-database data servers. A single Garlic query can access data in a number of different subsystems. An example of a nontraditional subsystem that Garlic will access is QBIC [NBE+93] ("Query By Image Content"). QBIC can search for images by various visual characteristics such as color, shape, and texture.

In this paper, we discuss the semantics of Garlic queries. This semantics resolves the mismatch that occurs because the result of a QBIC query is a sorted list (of items that match the query best), whereas the result of a relational database query is a set. Our semantics uses "graded" (or "fuzzy") sets [Za65]. Issues of efficient query evaluation in such a system are very different from those in a traditional database system. As a first step in dealing with these fascinating new issues, an optimal algorithm for evaluating an important class of Garlic queries is presented. This algorithm has been implemented in Garlic.[1]

In Section 2, we discuss the problem of the mismatch in semantics in more detail, and give our simple solution. In Section 3, we consider various operators in the literature for conjunction and disjunction, and focus on those properties of interest to us for the conjunction, namely "strictness" and "monotonicity". In Section 4, we present algorithms for evaluating the conjunction of atomic queries. In Section 5, we define the performance cost of algorithms, and give a result that says that the performance cost of our algorithm is small (in particular, sublinear), under natural assumptions. This upper bound depends on conjunction being monotone. In Section 6, we give a lower bound, which implies that the cost of our algorithm is optimal up to a constant factor. This lower bound depends on conjunction being strict. In Section 7, we give an example of a query

[1] Alissa Pritchard did the implementation.

that is hard (in the sense that every algorithm for this query must retrieve a linear number of objects in the database). In Section 8, we discuss the effect of various assumptions on the existence of efficient algorithms. In Section 9, we give our conclusions.

2 Semantics

In response to a query, QBIC returns a sorted list of the top, say, 10, items in its database that match the query the best. For example, if the query asks for red objects (where "red" might be selected from a color wheel), then the result would be a sorted list with the reddest object first, the next reddest object second, etc.

By contrast, the result of a query to a relational database is simply a set. This leads to a mismatch: the result of some queries is a sorted list, and for other queries, it is a set. How do we combine such queries in Boolean combinations? As an example, let us consider an application of a store that sells compact disks. A typical traditional database query might ask for the names of all albums where the artist is the Beatles. The result is a set of names of albums. A multimedia query might ask for all album covers with a particular shade of red. Here the result is a sorted list of album covers. We see the mismatch in this example: the query *Artist='Beatles'* gives us a set, whereas the query *AlbumColor='red'* gives us a sorted list.[2] How do we combine a traditional database query and a multimedia query? For example, consider the query

(Artist='Beatles') $\wedge$ *(AlbumColor='red').*

What is the result of this query? In this case, we probably want a sorted list, that contains only albums by the Beatles, where the list is sorted by goodness of match in color. What about more complicated queries? For example, what should the result be if we replaced $\wedge$ by $\vee$ in the previous query? Is the answer a set, a sorted list, or some combination? How about if we combine two multimedia queries? An example is given by the query

(Sound='loud') $\vee$ *(AlbumColor='red').*

Our solution is in terms of graded sets. A graded set is a set of pairs (x, g), where x is an object (such as a tuple), and g (the grade) is a real number in the interval $[0, 1]$. It is sometimes convenient to think of a graded set as corresponding to a sorted list, where the objects are sorted by their grades. Thus, a graded set is a generalization of both a set and a sorted list.

Although our graded-set semantics is applicable very generally, we shall make certain simplifying assumptions for the rest of the paper. This will make the discussion and the statement of the results easier. Furthermore, these simplifying assumptions enable us to avoid messy implementation-specific details (such as object-oriented system versus relational database system, and the choice of query language). It is easy to see that our semantics is very robust, and does not depend on any of these assumptions. On the other hand, our results, which we view only as a first step, do depend on our assumptions.

We assume that all of the data in all of the subsystems that we are considering (that are accessed by Garlic) deal with the attributes of a specific set of objects of some fixed type. In the running example involving compact disks that we have been considering, each query, such as *Artist='Beatles'* or *AlbumColor='red'*, deals with the attributes of compact disks. As in these examples, we take *atomic queries* to be of the form $X = t$, where X is the name of an attribute, and t is a target. *Queries* are Boolean combinations of atomic queries.

For each atomic query, a grade is assigned to each object. The grade represents the extent to which that object fulfills that atomic query, where the larger the grade is, the better the match. In particular, a grade of 1 represents a perfect match. For traditional database queries, such as *Artist='Beatles'*, the grade for each object is either 0 or 1, where 0 means that the query is false about the object, and 1 means that the query is true about the object. For other queries, such as a QBIC query corresponding to *AlbumColor='red'*, grades may be intermediate values between 0 and 1.

There is now a question of how to assign grades when the query is not necessarily atomic, but possibly a Boolean combination of atomic queries. We consider this issue in the next section.

3 Dealing with Boolean combinations

A number of different rules for evaluating Boolean combinations of atomic formulas in fuzzy logic have appeared in the literature. In particular, there are a number of reasonable "scoring functions" that assign a grade to a fuzzy conjunction, as a function of the grades assigned to the conjuncts.

We consider first the standard rules of fuzzy logic, as defined by Zadeh [Za65]. These are the rules that are currently adopted by Garlic. If x is an object and Q is a query, let us denote by $\mu_Q(x)$ the grade of x under the query Q. If we assume that $\mu_Q(x)$ is defined for each atomic query Q and each object x, then it is possible to extend to queries that are Boolean combination of atomic queries via the following rules.

[2] We are writing the query in the form *AlbumColor='red'* for simplicity. In reality, it might be expressed by selecting a color from a color wheel, or by selecting an image I (that might be predominantly red) and asking for other images whose colors are "close to" that of image I. Systems such as QBIC have sophisticated color-matching algorithms [NBE+93] that compute the difference between the colors of two images. For example, an image that contains a lot of red and a little green might be considered moderately close in color to another image with a lot of pink and no green.

Conjunction rule: $\mu_{A \wedge B}(x) = \min\{\mu_A(x), \mu_B(x)\}$

Disjunction rule: $\mu_{A \vee B}(x) = \max\{\mu_A(x), \mu_B(x)\}$

Negation rule: $\mu_{\neg A}(x) = 1 - \mu_A(x)$

Thus, the standard conjunction rule for fuzzy logic is based on using min as the scoring function.

These rules are attractive for two reasons. First, they are a conservative extension of the standard propositional semantics. That is, if we restrict our attention to situations where $\mu_Q(x)$ is either 0 or 1 for each atomic query Q, then these rules reduce to the standard rules of propositional logic. The second reason is because of an important theorem of Bellman and Giertz [BG73], and simplified by Yager [Ya82] and Dubois and Prade [DP84]. We now discuss the Bellman-Giertz theorem.

The standard conjunction and disjunction rules of fuzzy logic have the nice property that if Q_1 and Q_2 are logically equivalent queries involving only conjunction and disjunction (not negation), then $\mu_{Q_1}(x) = \mu_{Q_2}(x)$ for every object x. For example, $\mu_{A \wedge A}(x) = \mu_A(x)$. This is desirable, since then an optimizer can replace a query by a logically equivalent query, and be guaranteed of getting the same answer.

Furthermore, the scoring function min for conjunction is *monotone*, in the sense that if $\mu_A(x) \leq \mu_A(x')$, and $\mu_B(x) \leq \mu_B(x')$, then $\mu_{A \wedge B}(x) \leq \mu_{A \wedge B}(x')$. Similarly, the scoring function max for disjunction is monotone. Monotonicity is certainly a reasonable property to demand, and models the user's intuition. Intuitively, if the grade of object x_1 under the query A (resp., under the query B) is at least as big as that of object x_2, the grade of object x_1 under the query $A \wedge B$ is at least as big as that of object x_2 under the query $A \wedge B$.

The next theorem, due to Yager [Ya82] and Dubois and Prade [DP84], is a variation of the Bellman-Giertz theorem that says that min and max are the unique scoring functions for conjunction and disjunction with these properties. (Bellman and Giertz's original theorem required more assumptions.)

Theorem 3.1 : *The unique scoring functions for evaluating $\wedge$ and $\vee$ that preserve logical equivalence and that are monotone in their arguments are min and max.*

Let us define an *m-ary scoring function* to be a function from $[0,1]^m$ to $[0,1]$. For the sake of generality, we will consider m-ary scoring functions for evaluating conjunctions of m atomic queries, although in practice an m-ary conjunction is almost always evaluated by using an associative 2-ary function that is iterated. Analogously to the binary case, we say that an m-ary scoring function t is *monotone* if $t(x_1, \ldots, x_m) \leq t(x'_1, \ldots, x'_m)$ when $x_i \leq x'_i$ for every i. As discussed before, monotonicity is a reasonable property to expect a scoring function to obey. Another such property is *strictness*: an m-ary scoring function t is *strict* if $t(x_1, \ldots, x_m) = 1$ iff $x_i = 1$ for every i. Thus, a scoring function is strict if it takes on the maximal value of 1 precisely if each argument takes on this maximal value. Scoring functions considered in the literature seem to be monotone and strict. In particular, scoring functions derived from "triangular norms" [SS63, DP80] are monotone and strict. Another important class of scoring functions include various weighted and unweighted arithmetic and geometric means, which Thole, Zimmerman, and Zysno [TZZ79] found to perform empirically quite well. These are also monotone and strict.

We can define an m-ary query (such as the conjunction of m formulas) in terms of an m-ary scoring function. The semantics of an m-ary query $F(A_1, \ldots, A_m)$ is given by defining $\mu_{F(A_1,\ldots,A_m)}$. For example, the standard fuzzy logic semantics of the conjunction $A_1 \wedge \ldots \wedge A_m$ is given by defining

$$\mu_{A_1 \wedge \ldots \wedge A_m}(x) = \min\{\mu_{A_1}(x), \ldots, \mu_{A_m}(x)\},$$

for each object x. Let t be an m-ary scoring function. We define the m-ary query $F_t(A_1, \ldots, A_m)$ by taking

$$\mu_{F_t(A_1,\ldots,A_m)}(x) = t(\mu_{A_1}(x), \ldots, \mu_{A_m}(x)).$$

For example, if t is min, then $F_t(A_1, \ldots, A_m)$ is equivalent in the standard fuzzy semantics to $A_1 \wedge \ldots \wedge A_m$. We call $F_t(A_1, \ldots, A_m)$ a *strict* (resp., *monotone*) *query* if t is strict (resp., monotone). The only properties of a query that are required in this paper for our theorems to hold are strictness and monotonicity. We need strictness for our lower bound on the efficiency of algorithms for evaluating queries under certain assumptions, and monotonicity for our upper bound.

4 Algorithms for query evaluation

A vital issue in any database management system is the efficiency of processing queries. In this section, we give an algorithm for evaluating monotone queries. Later, we show that under certain assumptions the algorithm is optimally efficient up to a constant factor.

Probably the most important queries are those that are conjunctions of atomic queries. For the sake of the current discussion, let us assume for now that conjunctions are being evaluated by the standard min rule. An example of a conjunction of atomic queries is the query

(Artist='Beatles') $\wedge$ (AlbumColor='red').

that we have discussed in our running example. In this example, the first conjunct *Artist='Beatles'* is a traditional database query, and the second conjunct

AlbumColor='red' would be addressed to a subsystem such as QBIC. Thus, two different subsystems (in this case, perhaps a relational database management system to deal with the first conjunct, along with QBIC to deal with the second conjunct) would be involved in answering the query. Garlic has to piece together information from both subsystems in order to answer the query. Under the reasonable assumption that there are not many objects that satisfy the first conjunct *Artist='Beatles'*, a reasonable way to evaluate this query would be to first determine all objects that satisfy the first conjunct (call this set of objects S), and then to obtain grades from QBIC for the second conjunct for all objects in S.[3] We can thereby obtain a grade for all objects for the full query. If the artist is not the Beatles, then the grade for the object is 0 (since the minimum of 0 and any grade is 0). If the artist is the Beatles, then the grade for the object is the grade obtained from QBIC in evaluating the second conjunct (since the minimum of 1 and any grade g is g). Note that, as we would expect, the result of the full query is a graded set where (a) the only objects whose grade is nonzero have the artist as the Beatles, and (b) among objects where the artist is the Beatles, those whose album cover are closest to red have the highest grades.

Let us now consider a more challenging example of a conjunction of atomic queries, where more than one conjunct is "nontraditional". An example would be the query

(Color='red') $\wedge$ *(Shape='round').*

For the sake of this example, we assume that one subsystem deals with colors, and a completely different subsystem deals with shapes. Let A_1 denote the subquery *Color='red'*, and let A_2 denote the subquery *Shape='round'*. The grade of an object x under the query above is the minimum of the grade of x under the subquery A_1 from one subsystem and the grade of x under the subquery A_2 from the second subsystem. Therefore, Garlic must again combine results from two different subsystems. Assume that we are interested in obtaining the top k answers (such as $k = 10$). This means that we want to obtain k objects with the highest grades on this query (along with their grades). If there are ties, then we want to arbitrarily obtain k objects and their grades such that for each y among these k objects and each z not among these k objects, $\mu_Q(y) \geq \mu_Q(z)$ for this query Q. There is an obvious naive algorithm:

1. Have the subsystem dealing with color to output explicitly the graded set consisting of all pairs $(x, \mu_{A_1}(x))$ for every object x.
2. Have the subsystem dealing with shape to output explicitly the graded set consisting of all pairs $(x, \mu_{A_2}(x))$ for every object x.
3. Use this information to compute $\mu_{A_1 \wedge A_2}(x) = \min\{\mu_{A_1}(x), \mu_{A_2}(x)\}$ for every object x. For the k objects x with the top grades $\mu_{A_1 \wedge A_2}(x)$, output the object along with its grade.

Can we do any better? On the face of it, it is not clear how we can efficiently obtain the desired k answers (or even what "efficient" means!)

What can we assume about the interface between Garlic and a subsystem such as QBIC? In response to a subquery, such as *Color='red'*, we can assume that the subsystem will output the graded set consisting of all objects, one by one, along with their grades under the subquery, in sorted order based on grade, until Garlic tells the subsystem to stop. Then Garlic could later tell the subsystem to resume outputting the graded set where it left off. Alternatively, Garlic could ask the subsystem for, say, the top 10 objects in sorted order, along with their grades, then request the next 10, etc. We refer to such types of access as "sorted access".

There is another way that we could expect Garlic to interact with the subsystem. Garlic could ask the subsystem the grade (with respect to a query) of any given object. We refer to this as "random access".

Shortly, we shall give an algorithm that evaluates conjunctions of atomic queries, and returns the top k answers. In fact, the algorithm applies to any monotone query. We note, however, that in the case of max, which is certainly monotone, and which standard fuzzy disjunction is defined in terms of, there is a much more efficient algorithm, as we shall discuss at the end of this section. Finally, as is discussed in another paper [FW95], this algorithm applies also when the user can weight the relative importance of the conjuncts (for example, where the user decides that color is twice as important to him as shape), since such "weighted conjunctions" are also monotone.

We now give a proposition that is the key as to why our algorithm is correct. Let us say that a set X of objects is *upwards closed* with respect to a query Q if whenever x and y are objects with $x \in X$ and $\mu_Q(y) > \mu_Q(x)$, then $y \in X$. Thus, X is upwards closed with respect to Q if every object with a grade under Q that is strictly greater than that of a member of X is also in X.

Proposition 4.1: *Assume that X^i is upwards closed with respect to query A_i, for $1 \leq i \leq m$. Assume that $F_t(A_1, \ldots, A_m)$ is a monotone query, that x and z are objects with $x \in \cap_i X^i$, and $\mu_{F_t(A_1,\ldots,A_m)}(z) > \mu_{F_t(A_1,\ldots,A_m)}(x)$. Then $z \in \cup_i X^i$.*

Proof: For ease in notation, let us write $F_t(A_1, \ldots, A_m)$ as Q. If it were the case that $\mu_{A_j}(x) \geq \mu_{A_j}(z)$ for

[3] We are assuming that QBIC can do such "random accesses" (which, in fact, it can). We return to this issue shortly.

every j, then by monotonicity of t, we would have that $t(\mu_{A_1}(x), \ldots, \mu_{A_m}(x)) \geq t(\mu_{A_1}(z), \ldots, \mu_{A_m}(z))$, that is, $\mu_Q(x) \geq \mu_Q(z)$, which contradicts our assumption that $\mu_Q(z) > \mu_Q(x)$. So $\mu_{A_j}(x) < \mu_{A_j}(z)$ for some j. Now $x \in \cap_i X^i \subseteq X^j$. Therefore, since X^j is upwards closed with respect to A_j, it follows that $z \in X^j \subseteq \cup_i X^i$, as desired. ∎

We now give an algorithm (called algorithm $\mathcal{A}_0$) that returns the top k answers for a monotone query $F_t(A_1, \ldots, A_m)$, which we denote by Q. (We assume that there are at least k objects, so that "the top k answers" makes sense.) Assume that subsystem i evaluates the subquery A_i. Our algorithm is based on Proposition 4.1. The idea is that each subsystem i will generate a set X^i that is upwards closed with respect to A_i, such that $\cap_i X^i$ contains at least k objects. It then follows (as we will show) from Proposition 4.1 that k objects with the highest grades must be in $\cup_i X^i$. The algorithm consists of three phases: sorted access, random access, and computation.

1. For each i, give subsystem i the query A_i under sorted access. Thus, subsystem i begins to output, one by one in sorted order based on grade, the graded set consisting of all pairs $(x, \mu_{A_i}(x))$, where as before x is an object and $\mu_{A_i}(x)$ is the grade of x under query A_i. Wait until there are at least k "matches", that is, wait until there is a set L of at least k objects such that each subsystem has output all of the members of L. More formally, for each τ, denote by G_τ^i the graded set consisting of the first τ pairs $(x, \mu_{A_i}(x))$ in the output of subsystem i. Let $X_\tau^i = \{x \,|\, (x, \mu_{A_i}(x)) \in G_\tau^i\}$, the projection of G_τ^i onto the first component. Thus, X_τ^i consists of the first τ objects in the output of subsystem i. Wait until there are at least k matches, that is, find T such that $L = \cap_{i=1}^m X_T^i$ contains at least k members.
2. For each object x that has been seen, that is, for each $x \in \cup_{i=1}^m X_T^i$, do random access to each subsystem j to find $\mu_{A_j}(x)$. Of course, if $x \in X_T^j$, then $\mu_{A_j}(x)$ has already been determined, so random access is not needed for this object x in this subsystem j.
3. Compute the grade $\mu_Q(x) = t(\mu_{A_1}(x), \ldots, \mu_{A_m}(x))$ for each object x that has been seen. Let Y be a set containing the k objects that have been seen with highest grades (ties are broken arbitrarily). The output is then the graded set $\{(x, \mu_Q(x)) \,|\, x \in Y\}$.

We now prove correctness of this algorithm.

Theorem 4.2: *For every monotone query $F_t(A_1, \ldots, A_m)$, algorithm $\mathcal{A}_0$ correctly returns the top k answers.*

Proof: For ease in notation, let us write $F_t(A_1, \ldots, A_m)$ as Q. Let N be the total number of objects x. Therefore, X_N^i contains all N objects for each i. Hence, $\cap_{i=1}^m X_N^i$ contains all N objects. Now by assumption, $k \leq N$. Therefore, $\cap_{i=1}^m X_N^i$ contains at least k objects. So T is well-defined in the sorted access phase of the algorithm (as the least τ such that $\cap_{i=1}^m X_\tau^i$ contains at least k objects).

By definition, Y has k members. Under our definition of "the top k answers", we need only show that if z is an arbitrary object not in Y, then for every $y \in Y$ we have $\mu_Q(y) \geq \mu_Q(z)$. Assume that $z \notin Y$, and $\mu_Q(y) < \mu_Q(z)$ for some $y \in Y$; we shall derive a contradiction. Since (a) L is a subset of $\cup_{i=1}^m X_T^i$ with at least k members, (b) Y consists of the k members of $\cup_{i=1}^m X_T^i$ with the highest grades, and (c) $y \in Y$, it follows that for some $x \in L$, we have $\mu_Q(x) \leq \mu_Q(y)$. Hence, $\mu_Q(x) < \mu_Q(z)$. Clearly, X_T^i is upwards closed with respect to A_i, for $1 \leq i \leq m$. Since also $x \in L = \cap_i X_T^i$, it follows from Proposition 4.1 that $z \in \cup_{i=1}^m X_T^i$. But then, since $\mu_Q(y) < \mu_Q(z)$ for some $y \in Y$, it follows by definition of Y that $z \in Y$. But this is a contradiction. ∎

Note that the algorithm has the nice feature that after finding the top k answers, in order to find the next k best answers we can "continue where we left off".

There are various minor improvements we can make to algorithm $\mathcal{A}_0$ to improve its performance slightly. (The *performance* of an algorithm is formally defined in Section 5.) For example, instead of using a uniform value of T, we might find $T_i \leq T$ for each i such that $\cap_{i=1}^m X_{T_i}^i$ contains k members. We could then replace all occurrences of $\cup_{i=1}^m X_T^i$ in algorithm $\mathcal{A}_0$ by $\cup_{i=1}^m X_{T_i}^i$, which could lead to fewer random accesses.

For particular scoring functions t, we can modify algorithm $\mathcal{A}_0$ even further to improve its performance. For example, consider the important special case of the standard fuzzy conjunction $A_1 \wedge \ldots \wedge A_m$, where t is min. In this case, we can give a strengthening of Proposition 4.1, which, as we shall see, leads to a slightly more efficient algorithm.

Proposition 4.3: *Assume that X^i is upwards closed with respect to query A_i, for $1 \leq i \leq m$. Assume that t is min. Let x_0 be an object and i_0 a subsystem such that $\mu_{A_{i_0}}(x_0) = \min_{x \in \cap_{i=1}^m X^i} \min_i \mu_{A_i}(x)$. Assume that x and z are objects with $x \in \cap_i X^i$, and $\mu_{F_t(A_1,\ldots,A_m)}(z) > \mu_{F_t(A_1,\ldots,A_m)}(x)$. Then $z \in X^{i_0}$.*

Proof: For ease in notation, let us write $F_t(A_1, \ldots, A_m)$ as Q. Since t is min, the fact that $\mu_Q(z) > \mu_Q(x)$ says $\min\{\mu_{A_1}(z), \ldots, \mu_{A_m}(z)\} > \min\{\mu_{A_1}(x), \ldots, \mu_{A_m}(x)\}$. By definition of x_0 and i_0, it follows that

$$\min\{\mu_{A_1}(x), \ldots, \mu_{A_m}(x)\} \geq \mu_{A_{i_0}}(x_0).$$

So $\min\{\mu_{A_1}(z), \ldots, \mu_{A_m}(z)\} > \mu_{A_{i_0}}(x_0)$, and hence $\mu_{A_{i_0}}(z) > \mu_{A_{i_0}}(x_0)$. Since X^{i_0} is upwards closed, it follows that $z \in X^{i_0}$, as desired. ∎

We can use Proposition 4.3 to give a more efficient algorithm than Algorithm $\mathcal{A}_0$, when t is min. The idea is as follows. Let Q denote the query $F_t(A_1, \ldots, A_m)$, when t is the min. Let x_0 and i_0 be as in Proposition 4.3. Let $g_0 = \mu_Q(x_0)$. Intuitively, i_0 is a subsystem that has shown the smallest grade g_0 in the sorted access phase of algorithm $\mathcal{A}_0$, and x_0 is an object with this smallest grade g_0 in subsystem i_0. By the min rule, x_0 has overall grade g_0. Define the *candidates* to be the objects $x \in X_T^{i_0}$ with $\mu_{A_{i_0}}(x) \geq g_0$. We use the word "candidates", since these turn out to be the only candidates we need to consider for the set of objects with the top k answers. Define algorithm $\mathcal{A}_0'$ to be the result of replacing all occurrences of $\cup_{i=1}^m X_T^i$ in algorithm $\mathcal{A}_0$ by the set of candidates. Thus, algorithm $\mathcal{A}_0'$ is defined by taking the sorted access phase to be the same as the sorted access phase of algorithm $\mathcal{A}_0$, and taking the remaining two phases as follows:

2. Let x_0 be an object in L whose grade $\mu_Q(x_0)$ is the least of any member of L. Let i_0 be a subsystem such that $\mu_{A_{i_0}}(x_0) = \mu_Q(x_0)$. Let $g_0 = \mu_Q(x_0)$. The *candidates* are defined to be the objects $x \in X_T^{i_0}$ with $\mu_{A_{i_0}}(x) \geq g_0$. For each candidate x, do random access to each subsystem $j \neq i_0$ to find $\mu_{A_j}(x)$.

3. Compute the grade $\mu_Q(x) = \min\{\mu_{A_1}(x), \ldots, \mu_{A_m}(x)\}$ for each candidate x. Let Y be a set containing the k candidates with the highest grades (ties are broken arbitrarily). The output is then the graded set $\{(x, \mu_Q(x)) \mid x \in Y\}$.

Intuitively, algorithm $\mathcal{A}_0'$ has better performance than $\mathcal{A}_0$, since we do random access only for the candidates, each of which is a member of X^{i_0}, rather than doing random access for all of $\cup_{i=1}^m X_T^i$. The next theorem shows that algorithm $\mathcal{A}_0'$ gives the correct answer when t is min.

Theorem 4.4: *In the case of standard fuzzy conjunction (where the scoring function t is min), algorithm $\mathcal{A}_0'$ correctly returns the top k answers.*

Proof: Let Q be the standard fuzzy conjunction $A_1 \wedge \ldots \wedge A_m$. The proof is exactly the same as the proof of Theorem 4.2, except that instead of applying Proposition 4.1 to conclude that $z \in \cup_{i=1}^m X_T^i$, we instead apply Proposition 4.3 to conclude the stronger fact that $z \in X_T^{i_0}$. ∎

For certain monotone scoring functions t, we can define an algorithm that performs substantially better than algorithm $\mathcal{A}_0$. As an obvious example, let t be a constant function: then an arbitrary set of k objects (with their grades) can be taken to be the top k answers. Let us consider a more interesting and important example, where t is max, which corresponds to the standard fuzzy disjunction $A_1 \vee \ldots \vee A_m$. We will use this as an example later when we consider the limitations of our lower-bound results.

We now give an algorithm (called algorithm $\mathcal{B}_0$) that returns the top k answers for the standard fuzzy disjunction $A_1 \vee \ldots \vee A_m$ of atomic queries $A_1, \ldots, A_m$.

1. For each i, use sorted access to subsystem i to find the set X_k^i containing the top k answers to the query A_i.

2. For each $x \in \cup_{i=1}^m X_k^i$, let

$$h(x) = \max_i \{\mu_{A_i}(x) \mid x \in X^i\}.$$

Let Y be a set containing the k members x of $\cup_{i=1}^m X_k^i$ with the highest values of $h(x)$ (ties are broken arbitrarily). The output is then the graded set $\{(x, h(x)) \mid x \in Y\}$.

The next theorem is straightforward.

Theorem 4.5: *In the case of standard fuzzy disjunction (where the scoring function t is max), algorithm $\mathcal{B}_0$ correctly returns the top k answers.*

As we shall discuss later (in particular, after we define "performance cost"), the algorithm $\mathcal{B}_0$ has substantially better performance than algorithm $\mathcal{A}_0$.

5 Performance cost

In this section, we consider the performance cost of algorithms for evaluating queries. In particular, we focus on the cost of algorithm $\mathcal{A}_0$ when the scoring function is monotone.

Our measure of cost corresponds intuitively to the amount of information that an algorithm obtains from the database. The *sorted access cost* is the the total number of objects obtained from the database under sorted access. For example, if there are only two lists (corresponding, in the case of conjunction, to a query with two conjuncts), and some algorithm requests altogether the top 100 objects from the first list and the top 20 objects from the second list, then the sorted access cost for this algorithm is 120. Similarly, the *random access cost* is the the total number of objects obtained from the database under random access. The *database access cost* is the sum of the sorted access cost and the random access cost.

Using our notion of the database access cost as a cost measure is somewhat controversial. After all, a single sorted access is probably much more expensive than a single random access. However, our results are fairly robust with respect to a choice of cost measure. For example, let us consider the case of greatest interest in this paper, where the scoring function is monotone and strict. It follows from our lower bounds that except

for algorithms with an extremely large random access cost (at least equal to the number of objects in the database), no algorithm can have sorted access cost less than a constant times the database access cost of our algorithm $\mathcal{A}_0$. This shows that the cost performance of $\mathcal{A}_0$ is optimal up to a constant factor even under arbitrary cost measures where we charge more for each sorted access than for each random access. Note that under such cost measures, the database access cost of algorithm $\mathcal{A}_0$ is the same up to a constant factor as that under our original cost measure, and in particular is sublinear.

Of course, there are situations (such as in the case of a query optimizer) where we want a more realistic cost measure than our definition of the database access cost. The important point is that our lower and upper bounds are sufficiently robust that they probably apply even with this more realistic cost measure.

We will make probabilistic statements about the performance cost of algorithms, and so we will need to define a probabilistic model. Let N be the number of objects in the database. Our results say that if the atomic queries $A_1, \ldots, A_m$ are independent, then with arbitrarily high probability, the cost of algorithm $\mathcal{A}_0$ for evaluating $F_t(A_1, \ldots, A_m)$ is $O(N^{(m-1)/m}k^{1/m})$, which is sublinear (in contrast to the naive algorithm we described near the beginning of Section 4, which is linear). In particular, if $m = 2$ (so that there are exactly two atomic queries A_1 and A_2), then the cost of algorithm $\mathcal{A}_0$ is of the order of the square root of the database size. We now define our terms, to make these statements more precise.

Let us assume that the database contains N objects, which, for ease in notation, we call $1, \ldots, N$. Let a_i be the sorted list generated by subsystem i in response to the subquery A_i, for $i = 1, \ldots, m$. Thus, a_i is a permutation of $1, \ldots, N$. We write $a_i(j)$ to represent the jth member of a_i. Define a *primitive statement* to be a sentence of the form $a_i(j) = a_{i'}(j')$, where $i \neq i'$. We may refer to this primitive statement as a *primitive $\{i, i'\}$-statement*, where we explicitly mention the two subsystems (i and i') that are involved.

We now define what it means when we say that "the atomic queries are independent". There are two conditions. The first condition says intuitively that there is no interaction between primitive $\{i, i'\}$-statements and primitive $\{\ell, \ell'\}$-statements unless $\{i, i'\} = \{\ell, \ell'\}$. For each choice of i, i' where $i < i'$, let $\varphi_{\{i,i'\}}$ be a (standard propositional) conjunction of primitive $\{i, i'\}$-statements. Thus, $\varphi_{\{i,i'\}}$ is of the form

$$(a_i(j_1) = a_{i'}(j'_1)) \wedge \cdots \wedge (a_i(j_r) = a_{i'}(j'_r)).$$

The first condition (where $\Pr[\cdot]$ represents the probability) is:

$$\Pr[\bigwedge_{i<i'} \varphi_{\{i,i'\}}] = \prod_{i<i'} \Pr[\varphi_{\{i,i'\}}]$$

That is, the probability of the conjunction is the product of the probabilities.

The second condition says that the probabilities do not depend on the names of the objects. It says that for each permutation π of $1, \ldots, N$, we have:

$$\begin{aligned} & \Pr[(a_i(j_1) = a_{i'}(j'_1)) \wedge \cdots \wedge (a_i(j_r) = a_{i'}(j'_r))] \\ = \; & \Pr[(a_i(\pi(j_1)) = a_{i'}(\pi(j'_1))) \wedge \cdots \\ & \wedge (a_i(\pi(j_r)) = a_{i'}(\pi(j'_r)))]. \end{aligned}$$

It is straightforward to verify that taken together, our two conditions that define the notion of "the atomic queries are independent" are equivalent to saying that the probabilities of conjunctions of primitive statements behave as if each sorted list a_i contains the objects in random order, independent of the other lists, such that within each a_i, each permutation of $1, \ldots, N$ has equal probability.

Before we can prove a theorem on the cost of algorithm $\mathcal{A}_0$, we need a lemma. In this lemma, when we say that B_2 is a random set of ℓ_2 members of $\{1, \ldots, N\}$, we mean that all subsets of $\{1, \ldots, N\}$ of cardinality ℓ_2 are selected with equal probability. Before we state the lemma, let us explain how it will be used. In the sorted access phase of algorithm $\mathcal{A}_0$, sorted access to each subsystem takes place until there are at least k matches. That is, the sorted access phase continues until each subsystem has output T values under sorted access where T has the property that $\cap_{i=1}^m X_T^i$ contains at least k members. Therefore, in our analysis we are interested in determining, as a function of N, τ:

1. the expected size M of $\cap_{i=1}^m X_\tau^i$, and
2. the probability that the size of $\cap_{i=1}^m X_\tau^i$ is much smaller than this expected size M (in particular, is at most $M/2$).

We compute these quantities in an inductive fashion, by determining, for each j with $1 \leq j \leq m$, the expected size of $\cap_{i=1}^j X_\tau^i$, and the probability that the size of $\cap_{i=1}^j X_\tau^i$ is at most half of the expected size. In order to carry out this induction, we must know, as a function of N, ℓ_1, ℓ_2, the expected size of the intersection of ℓ_1 members of $\{1, \ldots, N\}$ with ℓ_2 randomly selected members of $\{1, \ldots, N\}$, and the probability that the size of this intersection is at most half of the expected size. That is what the following lemma does, under the assumption that ℓ_1 is not too big (in the lemma, for convenience we simply assume that $\ell_1/N \leq 1/10$). We denote the size of B by $|B|$.

Lemma 5.1 : *Let B_1 be a set of ℓ_1 members of $\{1,\ldots,N\}$, and let B_2 be a random set of ℓ_2 members of $\{1,\ldots,N\}$. Let $M = \ell_1\ell_2/N$. The expected size of $B = B_1 \cap B_2$ is M. Assume that $\ell_1/N \leq 1/10$. Then* $\Pr[|B| \leq M/2] < e^{-M/10}$.

The proof of Lemma 5.1 is given in the full paper [Fa95].

The next theorem discusses the cost of algorithm $\mathcal{A}_0$ in evaluating $F_t(A_1,\ldots,A_m)$. The theorem says that if the atomic queries $A_1,\ldots,A_m$ are independent, then with arbitrarily high probability the database access cost for algorithm $\mathcal{A}_0$ is $O(N^{(m-1)/m}k^{1/m})$, where N is the database size. We have to make sense of what we mean by the "probability" and what we mean by "with arbitrarily high probability". We consider the following formal framework.

We are considering a scenario where there are m atomic queries $A_1,\ldots,A_m$ over a database with N objects, which we are taking to be $1,\ldots,N$. For the purposes of this paper, it is convenient to focus on the graded sets associated with each atomic query. Therefore, we define a *scoring database* to be a function associating with each i (for $i = 1,\ldots,m$) a graded set, where the objects being graded are $1,\ldots,N$. Intuitively, the ith graded set in the scoring database is the graded set corresponding to the result of applying atomic query A_i to the original database. We may speak of random access (resp., sorted access) to the ith graded set in the scoring database, which corresponds to random access (resp., sorted access) to the original database under atomic query A_i. We define a *skeleton (on N objects)* to be a function associating with each i (for $i = 1,\ldots,m$) a permutation of $1,\ldots,N$. A scoring database $\mathcal{D}$ is *consistent with* skeleton $\mathcal{S}$ if for each i, the ith permutation in $\mathcal{S}$ gives a sorting of the ith graded set of $\mathcal{D}$ (in descending order of grade). A scoring database can be consistent with more than one skeleton if there are ties, that is, if for some i two distinct objects have the same grade in the ith graded set.

We are interested in the database access cost of algorithms that find the top k answers for $F_t(A_1,\ldots,A_m)$. For simplicity, we shall consider algorithms as being run against the scoring database (as opposed to being run against the original database), since the scoring database captures all that is relevant. Our algorithms are allowed only to do sorted access and random access to the scoring database. Because of ties, the sorted access cost might depend on which skeleton was used during the course of the algorithm. That is, if objects x and y have the same grade in list i, then it is possible that either x or y appears first during a sorted access to list i. If $\mathcal{A}$ is an algorithm, $\mathcal{D}$ is a scoring database, and $\mathcal{S}$ is a skeleton such that $\mathcal{D}$ is consistent with $\mathcal{S}$, we define $cost(\mathcal{A},\mathcal{D},\mathcal{S})$ to be the database access cost (the total number of sorted accesses and random accesses) of algorithm $\mathcal{A}$ when applied to scoring database $\mathcal{D}$ provided sorted access goes according to skeleton $\mathcal{S}$. We define $cost(\mathcal{A},\mathcal{S})$ to be the maximum of $cost(\mathcal{A},\mathcal{D},\mathcal{S})$ over all scoring databases $\mathcal{D}$ that are consistent with $\mathcal{S}$.[4] Thus, the cost of an algorithm over a skeleton is the worst-case cost of the algorithm over all scoring databases consistent with the skeleton. Similarly, we define $sortedcost(\mathcal{A},\mathcal{S})$, where we consider only the cost of sorted access. Note that if a scoring database $\mathcal{D}$ is consistent with more than one skeleton, then the specification of algorithm $\mathcal{A}$ says that $\mathcal{A}$ gives the top k answers with input $\mathcal{D}$ no matter which of these skeletons the algorithm "sees", that is, no matter which skeleton is used when the algorithm is run (although conceivably the database access cost might be different for different skeletons). The answers could also be different if there are ties, since in this case "the top k answers" could be one of several possibilities.

We now explain how we formalize the meaning of the statement that "if the atomic queries are independent, then with arbitrarily high probability the database access cost for algorithm $\mathcal{A}_0$ is $O(N^{(m-1)/m}k^{1/m})$". For a given N (database size) and m (number of lists), there are only a finite number of possible skeletons, and under an algorithm $\mathcal{A}$, each such skeleton $\mathcal{S}$ has database access cost $cost(\mathcal{A},\mathcal{S})$ as defined above. When we consider probabilities of database access costs, we are taking each such skeleton to have equal probability. (This corresponds to our assumption that atomic queries are independent.) When we say that for our algorithm $\mathcal{A}_0$, "with arbitrarily high probability the database access cost is $O(N^{(m-1)/m}k^{1/m})$", we mean that for every $\epsilon > 0$, there is a constant c such that for every N,

$$\Pr_{\mathcal{S}}[cost(\mathcal{A}_0,\mathcal{S}) > cN^{(m-1)/m}k^{1/m}] < \epsilon.$$

We write $\mathcal{S}$ under $\Pr[\cdot]$ to make it clear that the probability is taken over possible skeletons $\mathcal{S}$.

For the sake of making explicit the dependence of the cost on k (where the algorithm is obtaining the top k answers), we are thinking of k as a function of N, even though we suspect that users are most interested in the case where k is a small constant (like 10). Note also that the database access cost $O(N^{(m-1)/m}k^{1/m})$ is sublinear if $k = o(N)$, and in particular if k is a constant, which is the case of most interest. Note that when k is a constant and when $m = 2$ (which corresponds to two atomic queries), the database access cost is $O(\sqrt{N})$.

A few comments are in order about the extreme case where $k = N$. In this case, we certainly know *a priori* the top k objects (the k objects with the highest

[4] An algorithm $\mathcal{A}$ might behave differently over over two databases $\mathcal{D}$ and $\mathcal{D}'$ with the same skeleton $\mathcal{S}$. This is because the action of the algorithm might depend on the specific grades it sees. For example, an algorithm might take some special action when it sees a grade of 0.

grades): this is simply the set of all N objects. But to find the *grades* of the objects (which is required in our specification of finding "the top k values"), it is clearly necessary in general to access every entry in the database. Note that in this extreme case, the database access cost $O(N^{(m-1)/m}k^{1/m})$ as claimed in the theorem below is simply $O(N)$, as we would expect.

In the next theorem, we determine the database access cost for algorithm $\mathcal{A}_0$. In this theorem, we do not need to assume that $F_t(A_1, \ldots, A_m)$ is monotone. Monotonicity arises in correctness, not performance: the algorithm $\mathcal{A}_0$ is guaranteed to be correct only when $F_t(A_1, \ldots, A_m)$ is monotone (Theorem 4.2).

Theorem 5.2: *Assume that the m atomic queries are independent. The database access cost for algorithm $\mathcal{A}_0$ is $O(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability.*

The proof of Theorem 5.2, which is based on Lemma 5.1, is given in the full paper [Fa95].

6 Lower bounds

Theorem 5.2 says that if the m atomic queries are independent, then the database access cost for algorithm $\mathcal{A}_0$ is $O(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability. Since algorithm $\mathcal{A}_0$ is correct for monotone queries (by Theorem 4.2), this gives an upper bound of $O(N^{(m-1)/m}k^{1/m})$ for monotone queries. In this section, we give a matching lower bound in the case of strict queries. Thus, we show that in the case of strict queries, no correct algorithm $\mathcal{A}$ that finds the top k answers can do better. Our results say that for such an algorithm $\mathcal{A}$ and for each N and each $\theta \geq 0$,

$$\Pr_{\mathcal{S}}[cost(\mathcal{A}, \mathcal{S}) \leq \theta N^{(m-1)/m}k^{1/m}] \leq \theta^m.$$

Hence, there is no function f with $f = o(N^{(m-1)/m}k^{1/m})$ such that if the atomic queries are independent, then with arbitrarily high probability the database access cost for algorithm $\mathcal{A}$ is $O(f)$. So the database access cost $O(N^{(m-1)/m}k^{1/m})$ of algorithm $\mathcal{A}_0$ is optimal.

To prove our lower bound, we need to assume that the scoring function t is strict. Note that max is *not* strict. In fact, in the case of max, the lower bound does not hold. Algorithm $\mathcal{B}_0$ of Theorem 4.5 has database access cost only mk, independent of the size N of the database!

The next lemma, which we use in the proof of our lower bounds, says that if t is strict, then except in an extreme situation where the database access cost is at least N (the number of objects in the database), the sorted access cost is closely related to the size of the intersection of the top objects in each list.

Lemma 6.1: *Assume that t is strict. Let $\mathcal{S}$ be a skeleton on N objects, and let $\mathcal{A}$ be an arbitrary algorithm that finds the top k answers to $F_t(A_1, \ldots, A_m)$. Assume that $cost(\mathcal{A}, \mathcal{S}) < N$, and $T \geq sortedcost(\mathcal{A}, \mathcal{S})$. Let X_T^i denote the top T objects in list i according to skeleton $\mathcal{S}$. Then $\cap_{i=1}^m X_T^i$ contains at least k members.*

The proof of Lemma 6.1 is given in the full paper [Fa95], along with a discussion as to why we must consider $cost(\mathcal{A}, \mathcal{S})$ and $sortedcost(\mathcal{A}, \mathcal{S})$ in Lemma 6.1 rather than $cost(\mathcal{A}, \mathcal{D}, \mathcal{S})$ and $sortedcost(\mathcal{A}, \mathcal{D}, \mathcal{S})$.

We now give our lower bound, which says intuitively that every correct algorithm has database access cost at least a constant times that of our algorithm $\mathcal{A}_0$.

Theorem 6.2: *Let N be given. Assume that t is strict. Let $\mathcal{A}$ be an arbitrary algorithm that finds the top k answers to $F_t(A_1, \ldots, A_m)$. If $A_1, \ldots, A_m$ are independent, then*

$$\Pr_{\mathcal{S}}[cost(\mathcal{A}, \mathcal{S}) \leq \theta N^{(m-1)/m}k^{1/m}] \leq \theta^m,$$

for every $\theta \geq 0$.

The proof of Theorem 6.2, which is based on Lemma 6.1, is given in the full paper [Fa95].

The next theorem puts our results together to obtain a matching upper and lower bound. It says that if t is monotone and strict, then the database access cost for finding the top k answers to $F_t(A_1, \ldots, A_m)$, where $A_1, \ldots, A_m$ are independent, is $\Theta(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability. As usual, Θ means that there is a matching lower and upper bound (up to a constant factor). In this case, it means that

1. There is an algorithm $\mathcal{A}_0$ for finding the top k answers to $F_t(A_1, \ldots, A_m)$, such that for every $\epsilon > 0$, there is a constant c_1 such that for every N,
$$\Pr_{\mathcal{S}}[cost(\mathcal{A}_0, \mathcal{S}) > c_1 N^{(m-1)/m}k^{1/m}] < \epsilon.$$
2. For every algorithm $\mathcal{A}$ for finding the top k answers to $F_t(A_1, \ldots, A_m)$ and for every $\epsilon > 0$, there is a constant c_2 such that for every N,
$$\Pr_{\mathcal{S}}[cost(\mathcal{A}, \mathcal{S}) < c_2 N^{(m-1)/m}k^{1/m}] < \epsilon.$$

Theorem 6.3: *The database access cost for finding the top k answers to a monotone, strict query $F_t(A_1, \ldots, A_m)$, where $A_1, \ldots, A_m$ are independent, is $\Theta(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability.*

Proof: This follows in a straightforward way from Theorems 4.2, 5.2 and 6.2. ∎

Intuitively, Theorem 6.3 tells us that we have matching upper and lower bounds for any natural notion of conjunction.

We close this section by giving a variation of Theorem 6.2 that focuses on the sorted access cost, and discuss how this shows the robustness of our definition of database access cost.

Theorem 6.4: *Let N be given. Assume that t is strict. Let $\mathcal{A}$ be an arbitrary algorithm that finds the top k answers to $F_t(A_1, \ldots, A_m)$, with database access cost less than N, for every database with N objects. If $A_1, \ldots, A_m$ are independent, then*

$$\Pr_{\mathcal{S}}[sortedcost(\mathcal{A}, \mathcal{S}) \leq \theta N^{(m-1)/m} k^{1/m}] \leq \theta^m,$$

for every $\theta \geq 0$.

Again, the proof is omitted.

Let us assume that t is monotone and strict. Theorem 6.4 tells us that except for algorithms with an extremely large random access cost (at least equal to the number of objects in the database), no correct algorithm can have a sorted access cost less than a constant times that of our algorithm $\mathcal{A}_0$. Since the random access cost of our algorithm $\mathcal{A}_0$ is at most a constant times the sorted access cost, this tells us that except for algorithms with an extremely large random access cost, no correct algorithm can have a sorted access cost less than a constant times the database access cost of our algorithm $\mathcal{A}_0$. As we noted in Section 5, this shows that the cost performance of $\mathcal{A}_0$ is optimal up to a constant factor even under arbitrary cost measures where we charge more for each sorted access than for each random access.

7 A provably hard query

We have given an algorithm for evaluating the conjunction of atomic queries that is efficient when the conjuncts are independent. What if the conjuncts are not independent?

As we see from both our algorithm $\mathcal{A}_0$ (upper bound) and from our lower bound machinery (in particular, Lemma 6.1), in order to obtain the top k answers it is necessary to retrieve roughly T objects from the database, where T is the least value such that $\cap_{i=1}^m X_T^i$ contains at least k members. Therefore, if the conjuncts are positively correlated, this can only help the efficiency. What if the conjuncts are negatively correlated?

In this section, we consider the extreme case of negative correlation between queries, by considering queries $Q \wedge \neg Q$, for Q an atomic query. In standard propositional logic, such a query is unsatisfiable. But the situation is different if Q is "fully fuzzy" (that is, can take on any value in $[0, 1]$, not just 0 and 1).

Let us consider only the standard fuzzy semantics, where conjunction is evaluated by the min, and negation is evaluated by letting $\mu_{\neg A}(x) = 1 - \mu_A(x)$. Then $\mu_{Q \wedge \neg Q}(x) = 1/2$ when $\mu_Q(x) = 1/2$. Furthermore, it is easy to see that $1/2$ is the maximal possible value under $Q \wedge \neg Q$.

For convenience, we restrict our attention in this section to scoring databases where $\mu_Q(x) \neq \mu_Q(y)$ whenever x and y are distinct objects. This way, there are no ties.

We now give a theorem that says that the database access cost for finding the top answer to $Q \wedge \neg Q$ is $\Theta(N)$. In this case (where, unlike before, no probabilities are involved), this means that

1. There is an algorithm $\mathcal{A}$ for finding the top answer to $Q \wedge \neg Q$, and a constant c_1, such that for every skeleton $\mathcal{S}$ and every N,
$$cost(\mathcal{A}, \mathcal{S}) \leq c_1 N.$$
(This is trivial, since we can take $\mathcal{A}$ to be the naive algorithm described near the beginning of Section 4.)
2. For every algorithm $\mathcal{A}$ for finding the top answer to $Q \wedge \neg Q$, there is a constant c_2 such that for every skeleton $\mathcal{S}$ and every N,
$$cost(\mathcal{A}, \mathcal{S}) \geq c_2 N.$$

Theorem 7.1: *The database access cost for finding the top answer to the standard fuzzy conjunction $Q \wedge \neg Q$, where Q is fully fuzzy, is $\Theta(N)$.*

The proof of Theorem 7.1 is given in the full paper [Fa95].

Theorem 7.1 gives us a provably hard query: the query requires linear database access cost, the same cost as that incurred by the naive algorithm in evaluating the query.

8 Exploiting other information

We have discussed an algorithm $\mathcal{A}_0$ that works well in evaluating a monotone query $F_t(A_1, \ldots, A_m)$ when the atomic queries $A_1, \ldots, A_m$ are independent. Under additional assumptions, another algorithm may perform better. We now present an example, due to Jeff Ullman (personal communication). Assume that we are evaluating the standard fuzzy conjunction $A_1 \wedge A_2$ (where t is min). We now give an algorithm that finds the top answer (it is easy to see how to modify this algorithm to obtain the top k answers).

1. Give subsystem 2 the query A_2 under sorted access. Thus, subsystem 2 begins to output, one by one in sorted order based on grade, the graded set consisting of all pairs $(x, \mu_{A_2}(x))$.
2. As each pair $(x, \mu_{A_2}(x))$ is output from subsystem 2, do random access to subsystem 1 to obtain $\mu_{A_1}(x)$.
3. Stop as soon as an object x is found such that $\mu_{A_1}(x) \geq \mu_{A_2}(x)$.
4. For all of the objects x that have been seen, let x_0 be the object with the highest overall grade $g_0 = \min\{\mu_{A_1}(x_0), \mu_{A_2}(x_0)\}$. The output is then (x_0, g_0).

Correctness is easy to verify, since it is straightforward to see that no object that has not been seen can have overall grade greater than g_0. Assume that not only are the atomic queries A_1, A_2 independent, but also the grades of the objects under the query A_1 are uniformly distributed in $[0,1]$, and the maximum value γ of the grades of the objects under the query A_2 is less than 1 (note that we can find out the value of γ with one sorted access). Then the expected time to stop is after at most $1/(1-\gamma)$ objects have been seen, independent of the number N of objects in the database, since $\Pr[\mu_{A_1}(x) \geq \gamma] = 1 - \gamma$ for each object x.

Clearly other assumptions will lead us to consider other algorithms. It is an important problem to find other natural assumptions that lead to other efficient algorithms in cases of interest.

9 Conclusions

We have presented a semantics for Garlic, that allows us to combine information from different subsystems in a natural way. Furthermore, we have presented an algorithm that works efficiently on probably the most important class of queries, and given results that say that its performance cost is optimal. Both the upper bound and lower bound are quite robust, and hold for almost any reasonable rule for evaluating the conjunction.

Acknowledgements

The author is grateful to Laura Haas and Dragutin Petkovic for suggesting the problem, and for helpful discussions. The author is also grateful to Eli Upfal for valuable suggestions, and to Laura Haas and Ed Wimmers for detailed comments on a preliminary version of the paper.

References

[BG73] R. Bellman and M. Giertz, On the Analytic Formalism of the Theory of Fuzzy Sets, *Information Sciences* **5** (1973), pp. 149–156.

[CHS+95] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers, Towards Heterogeneous Multimedia Information Systems: the Garlic Approach, RIDE-DOM '95 (5th Int'l Workshop on Research Issues in Data Engineering: Distributed Object Management), 1995, pp. 124-131.

[CHN+95] W. F. Cody, L. M. Haas, W. Niblack, M. Arya, M. J. Carey, R. Fagin, M. Flickner, D. S. Lee, D. Petkovic, P. M. Schwarz, J. Thomas, M. T. Roth, J. H. Williams, and E. L. Wimmers, Querying Multimedia Data from Multiple Repositories by Content: the Garlic Project, *IFIP 2.6 3rd Working Conference on Visual Database Systems (VDB-3)*, 1995.

[DP80] D. Dubois and H. Prade, *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York, 1980.

[DP84] D. Dubois and H. Prade, Criteria Aggregation and Ranking of Alternatives in the Framework of Fuzzy Set Theory, in *Fuzzy Sets and Decision Analysis* (H. J. Zimmermann, L. A. Zadeh, and B. Gaines, Eds.), TIMS Studies in Management Sciences **20** (1984), pp. 209–240.

[Fa95] R. Fagin, Combining fuzzy information from multiple systems, IBM Research Report RJ 9980, October 1995.

[FW95] R. Fagin and E. Wimmers, User-Adjustable Weights in Fuzzy Formulas, to appear.

[NBE+93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker, The QBIC Project: Querying Images by Content Using Color, Texture and Shape, *SPIE Conference on Storage and Retrieval for Image and Video Databases* (1993), volume 1908, pp. 173–187.

[SS63] B. Schweizer and A. Sklar, Associative Functions and Abstract Semi-groups, *Publ. Math. Debrecen* **10** (1963), pp. 69–81.

[TZZ79] U. Thole, H.-J. Zimmerman, and P. Zysno, On the Suitability of Minimum and Product Operators for the Intersection of Fuzzy Sets, *Fuzzy Sets and Systems* **2** (1979), pp. 167–180.

[Ya82] R. R. Yager, Some Procedures for Selecting Fuzzy Set-Theoretic Operations, *International Journal General Systems* **8** (1982), pp. 115-124.

[Za65] L. A. Zadeh, Fuzzy sets, *Information and Control* **8** (1965), pp. 338–353.