Journal of Computer and System Sciences **58**, 83–99 (1999)
Article ID jcss.1998.1600, available online at http://www.idealibrary.com on IDEAL®

Combining Fuzzy Information from Multiple Systems*

Ronald Fagin†

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

Received July 4, 1996; revised June 22, 1998

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. In contrast, in this paper 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 database size). 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.

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 nondatabase 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[1] [NBE + 93] ("Query By Image Content"). QBIC can search for images by various visual characteristics such as color 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.[2]

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 "monotonicity" and "strictness." In Section 4, we present an algorithm (with several variations) for evaluating the conjunction of atomic queries. In Section 5, we define the performance cost of algorithms and prove 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 prove 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 use our lower-bound machinery to prove that a certain query 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 what we can do when a subsystem of interest has different semantics from the overall Garlic semantics. In Section 9, we discuss how additional assumptions might lead to more efficient

* An extended abstract of this paper appears in "Proc. Fifteenth ACM Symp. on Principles of Database Systems," Montreal, 1996, pp. 216–226.

† URL: http://www.almaden.ibm.com/cs/people/fagin/

[1] QBIC is a trademark of IBM Corporation.

[2] Alissa Pritchard and Christoph Braendli did the implementation, each in a different version of Garlic.

0022-0000/99 $30.00

algorithms. In Section 10, we consider related work. In Section 11, 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, then the result would be a sorted list with the reddest object first, the next reddest object second, etc.

In contrast, the result of a query to a relational database is simply a set.[3] This leads to a mismatch: the result of some queries is a sorted list, and that of other queries 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 $Artist = \text{“}Beatles\text{”}$ gives us a set, whereas the query $AlbumColor = \text{“}red\text{”}$ gives us a sorted list.[4] How do we combine a traditional database query and a multimedia query? For example, consider the query

$$(Artist = \text{“}Beatles\text{”}) \wedge (AlbumColor = \text{“}red\text{”}).$$

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

$$(Color = \text{“}red\text{”}) \wedge (Shape = \text{“}round\text{”}).$$

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 the query $Artist = \text{“}Beatles\text{”}$ or the query $AlbumColor = \text{“}red\text{”}$, 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 $Artist = \text{“}Beatles\text{”}$, 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 $AlbumColor = \text{“}red\text{”}$, 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 “aggregation functions” that assign a grade to a fuzzy conjunction, as a function of the grades assigned to the conjuncts. In this section, we present some of these aggregation functions. Some of this section is based on Zimmermann’s textbook [Zi96]. We introduce the notion of “strictness,” which we have not found in the literature, but which is probably there somewhere! We are interested in this notion of strictness (and of the well-known notion of “monotonicity”) since (a) almost all (and possibly all) aggregation functions in the

[3] Of course, in a relational database, the result to a query may be sorted in some way for convenience in presentation, such as sorting department members by salary, but logically speaking, the result is still simply a set, with a crisply defined collection of members.

[4] We are writing the query in the form $AlbumColor = \text{“}red\text{”}$ 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 [Io89, NBE + 93, SO95, SC96] that compute the closeness of 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.

literature that are intended to deal with conjunction are monotone and strict, and (b) these two properties are sufficient for the sake of our theorems. In particular, our theorems apply to a large class of possible choices of aggregation functions for the fuzzy conjunction.

We consider first the standard rules of fuzzy logic, as defined by Zadeh [Za65]. 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.

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 aggregation 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 due to an important theorem of Bellman and Giertz [BG73], extended and simplified by Yager [Ya82], Voxman and Goetschel [VG83], Dubois and Prade [DP84], and Wimmers [Wi98a]. 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, $\mu_{A \wedge A}(x) = \mu_A(x)$. As another example, $\mu_{A \wedge (B \vee C)}(x) = \mu_{(A \wedge B) \vee (A \wedge C)}(x)$. 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 aggregation function min for conjunction is *monotone*, in the sense that if $\mu_A(x) \leqslant \mu_A(x')$, and $\mu_B(x) \leqslant \mu_B(x')$, then $\mu_{A \wedge B}(x) \leqslant \mu_{A \wedge B}(x')$. Similarly, the aggregation 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' under the query A is at least as big as that of object x, and the grade of object x' under the query B is at least as big as that of object x, then the grade of object x' under the query $A \wedge B$ is at least as big as that of object x.

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 aggregation functions for conjunction and disjunction with these properties. (Bellman and Giertz's original theorem required more assumptions.)

THEOREM 3.1. *The unique aggregation functions for evaluating $\wedge$ and $\vee$ that preserve logical equivalence of queries involving only conjunction and disfunction and that are monotone in their arguments are min and max.*

Before we mention other possible aggregation functions for evaluating the conjunction and disjunction that have appeared in the literature, we introduce some notation.

Let us define an *m-ary aggregation function* to be a function from $[0, 1]^m$ to $[0, 1]$. For the sake of generality, we will consider m-ary aggregation 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.

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

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

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

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

For example, if t is min, then $F_t(A_1, ..., A_m)$ is equivalent in the standard fuzzy semantics to $A_1 \vee \cdots \vee A_m$, and if t is max, then $F_t(A_1, ..., A_m)$ is equivalent in the standard fuzzy semantics to $A_1 \vee \cdots \vee A_m$.

We now consider an important class of 2-ary aggregation functions. A *triangular norm* [SS63, DP80] is a 2-ary aggregation function t that satisfies the following properties:

$\wedge$-Conservation: $t(0, 0) = 0$; $t(x, 1) = t(1, x) = x$.

Monotonicity: $t(x_1, x_2) \leqslant t(x_1', x_2')$ if $x_1 \leqslant x_1'$ and $x_2 \leqslant x_2'$.

Commutativity: $t(x_1, x_2) = t(x_2, x_1)$.

Associativity: $t(t(x_1, x_2), x_3) = t(x_1, t(x_2, x_3))$.

It is reasonable to expect an aggregation function for conjunction to satisfy each of the properties of a triangular norm. We call the first condition "$\wedge$-conservation," since it implies in particular a conservative extension of the standard propositional semantics for conjunction, as we discussed in the case of min.

A *triangular co-norm* [DP85] is a 2-ary aggregation function s that satisfies the following properties:

$\vee$-Conservation: $s(1, 1) = 1$; $s(x, 0) = s(0, x) = x$.

Monotonicity: $s(x_1, x_2) \leqslant s(x_1', x_2')$ if $x_1 \leqslant x_1'$ and $x_2 \leqslant x_2'$.

Commutativity: $s(x_1, x_2) = s(x_2, x_1)$.

Associativity: $s(s(x_1, x_2), x_3) = s(x_1, s(x_2, x_3))$.

It is reasonable to expect an aggregation function for disjunction to satisfy each of the properties of a triangular co-norm.

Triangular norms and triangular co-norms are duals, in the sense that if t is a triangular norm, then the function s defined by $s(x_1, x_2) = 1 - t(1 - x_1, 1 - x_2)$ is a triangular co-norm [Al85]. Bonnissone and Decker [BD86] show that for suitable negation aggregation functions n (such as the standard $n(x) = 1 - x$), the natural generalization of DeMorgan's Laws hold between a triangular norm t and its co-norm s:

$$s(x_1, x_2) = n(t(n(x_1), n(x_2)))$$
$$t(x_1, x_2) = n(s(n(x_1), n(x_2))).$$

Below are some examples of triangular norms and their corresponding co-norms [BD86, Mi89]:

Minimum: $t(x_1, x_2) = \min\{x_1, x_2\}$.
Maximum: $s(x_1, x_2) = \max\{x_1, x_2\}$.

Drastic product:

$$t(x_1, x_2) = \begin{cases} \min\{x_1, x_2\} & \text{if } \max\{x_1, x_2\} = 1 \\ 0 & \text{otherwise} \end{cases}.$$

Drastic sum:

$$s(x_1, x_2) = \begin{cases} \max\{x_1, x_2\} & \text{if } \min\{x_1, x_2\} = 1 \\ 1 & \text{otherwise} \end{cases}.$$

Bounded difference: $t(x_1, x_2) = \max\{0, x_1 + x_2 - 1\}$.
Bounded sum: $s(x_1, x_2) = \min\{1, x_1 + x_2\}$.

Einstein product: $t(x_1, x_2) = (x_1 \cdot x_2) / (2 - (x_1 + x_2 - x_1 \cdot x_2))$.
Einstein sum: $s(x_1, x_2) = (x_1 + x_2)/(1 + x_1 \cdot x_2)$.

Algebraic product: $t(x_1, x_2) = x_1 \cdot x_2$.
Algebraic sum: $s(x_1, x_2) = x_1 + x_2 - x_1 \cdot x_2$.

Hamacher product: $t(x_1, x_2) = (x_1 \cdot x_2)/(x_1 + x_2 - x_1 \cdot x_2)$.
Hamacher sum: $s(x_1, x_2) = (x_1 + x_2 - 2x_1 \cdot x_2)/(1 - x_1 \cdot x_2)$.

As we mentioned earlier, an m-ary conjunction can of course be obtained from a 2-ary conjunction by iterating. For example, if 2-ary conjunction is defined by the 2-ary aggregation function t, then 3-ary conjunction can be defined by $t(t(x_1, x_2), x_3)$. The following two properties hold for every m-ary aggregation function that is obtained by iterating a triangular norm:

Monotonicity: $t(x_1, ..., x_m) \leqslant t(x'_1, ..., x'_m)$ if $x_i \leqslant x'_i$ for every i.

Strictness: $t(x_1, ..., x_m) = 1$ iff $x_i = 1$ for every i.

Thus, an aggregation function is strict if it takes on the maximal value of 1 precisely if each argument takes on this maximal value. Strictness follows from the fact [DP80] that every triangular norm is bounded below by the drastic product and above by the min.

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

We note that there are aggregation functions for conjunction that have been considered in the literature that are not triangular norms. For example, Thole *et al.* [TZZ79] found various weighted and unweighted arithmetic and geometric means to perform empirically quite well. Such aggregation functions are not triangular norms: in fact, the arithmetic mean does not conserve the standard propositional semantics, since with arguments 0 and 1 it takes the value 1/2, rather than 0. These functions do satisfy monotonicity and strictness, and so our upper and lower bounds hold even in this case. Thus, if a system were to use, say, the arithmetic mean as an aggregation function for evaluating the conjunction, then our lower and upper bounds tell us how efficiently we can expect to evaluate the conjunction under natural assumptions.

We discuss examples and algorithms in the next section.

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

$$(Artist = \text{“}Beatles\text{”}) \wedge (AlbumColor = \text{“}red\text{”}),$$

which we discussed in our running example. In this example, the first conjunct $Artist = \text{“}Beatles\text{”}$ is a traditional database query, and the second conjunct $AlbumColor = \text{“}red\text{”}$ 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 good way to evaluate this query would be first to determine all objects that satisfy the first conjunct (call this set of objects S), and then to obtain grades from QBIC (using random access) for the second conjunct for all objects in S.[5] 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

$$(Color = \text{"}red\text{"}) \wedge (Shape = \text{"}round\text{"}).$$

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 (which we may refer to as the "top 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, $\mu_Q(y) \geqslant \mu_Q(z)$ 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.

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

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 [FW97], 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 $1 \leqslant i \leqslant m$. Assume that $F_t(A_1, ..., A_m)$ is a monotone query, that x and z are objects with $x \in \bigcap_i X^i$, and $\mu_{F_t(A_1, ..., A_m)}(z) > \mu_{F_t(A_1, ..., A_m)}(x)$. Then $z \in \bigcup_i X^i$.*

Proof. For ease in notation, let us write $F_t(A_1, ..., A_m)$ as Q. If it were the case that $\mu_{A_j}(x) \geqslant \mu_{A_j}(z)$ for every j, then by monotonicity of t, we would have that $t(\mu_{A_1}(x), ..., \mu_{A_m}(x)) \geqslant t(\mu_{A_1}(z), ..., \mu_{A_m}(z))$, that is, $\mu_Q(x) \geqslant \mu_Q(z)$, which contradicts our assumption that $\mu_Q(z) > \mu_Q(x)$. So $\mu_{A_j}(x) < \mu_{A_j}(z)$ for some j. Now $x \in \bigcap_i X^i \subseteq X^j$. Therefore, since X^j is upwards closed with respect to A_j, it follows that $z \in X^j \subseteq \bigcup_i X^i$, as desired. ∎

We now give an algorithm (called algorithm $\mathscr{A}_0$) that returns the top k answers for a monotone query $F_t(A_1, ..., A_m)$,

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 $\bigcap_i X^i$ 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 $\bigcup_i X^i$. The algorithm consists of three phases: sorted access, random access, and computation.

We first present the algorithm somewhat informally.

Sorted access phase: 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.

Random access phase: For each object x that has been seen, do random access to each subsystem j to find $\mu_{A_j}(x)$.

Computation phase: Compute the grade $\mu_Q(x) = t(\mu_{A_1}(x), ..., \mu_{A_m}(x))$ 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 $\{(x, \mu_Q(x)) \mid x \in Y\}$.

We now describe the sorted access phase and the random access phase a little more formally, and introduce notation that will be useful later.

In the sorted access phase, for each τ let us denote by G^i_τ the graded set consisting of the first τ pairs $(x, \mu_{A_i}(x))$ in the output of subsystem i. Let $X^i_\tau = \{x \mid (x, \mu_{A_i}(x)) \in G^i_\tau\}$, the projection of G^i_τ onto the first component. Thus, X^i_τ consists of the first τ objects in the output of subsystem i. In the sorted access phase, we wait until there are at least k matches; that is, we find T such that $L = \bigcap_{i=1}^m X^i_T$ contains at least k members.

In the random access phase, when we say that an object x "has been seen," we mean that $x \in \bigcup_{i=1}^m X^i_T$. In the random access phase, for each such object x we do random access to each subsystem j to find $\mu_{A_j}(x)$. Of course, if $x \in X^j_T$, then $\mu_{A_j}(x)$ has already been determined, so random access is not needed for this object x in this subsystem j.

We now prove correctness of this algorithm.

THEOREM 4.2. *For every monotone query, algorithm $\mathscr{A}_0$ correctly returns the top k answers.*

Proof. Let Q be a monotone query. Let N be the total number of objects x. Therefore, X^i_N contains all N objects for each i. Hence, $\bigcap_{i=1}^m X^i_N$ contains all N objects. Now by assumption, $k \leqslant N$. Therefore, $\bigcap_{i=1}^m X^i_N$ contains at least k objects. So T is well defined in the sorted access phase of the algorithm (as the least τ such that $\bigcap_{i=1}^m X^i_\tau$ 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 $\mu_Q(y) \geqslant \mu_Q(z)$. 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 $\bigcup_{i=1}^m X^i_T$ with at least k members, (b) Y consists of the k members of $\bigcup_{i=1}^m X^i_T$ with the highest grades, and (c) $y \in Y$, it follows that for some $x \in L$, we have $\mu_Q(x) \leqslant \mu_Q(y)$. Hence, $\mu_Q(x) < \mu_Q(z)$. Clearly, X^i_T is upwards closed with respect to A_i, for $1 \leqslant i \leqslant m$. Since also $x \in L = \bigcap_i X^i_T$, it follows from Proposition 4.1 that $z \in \bigcup_{i=1}^m X^i_T$. 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 $\mathscr{A}_0$ 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 $T_i \leqslant T$ for each i such that $\bigcap_{i=1}^m X^i_{T_i}$ contains k members. We could then replace all occurrences of $\bigcap_{i=1}^m X^i_T$ in algorithm $\mathscr{A}_0$ by $\bigcup_{i=1}^m X^i_{T_i}$, which could lead to fewer random accesses. Ait-Bouziad and Kassel [AK98] give another such improvement.

For particular aggregation functions t, we can modify algorithm $\mathscr{A}_0$ even further to improve its performance. For example, consider the important special case of the standard fuzzy conjunction $A_1 \wedge \cdots \wedge A_m$, 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 $1 \leqslant i \leqslant m$. Assume that t is min. Let i_0 be a value of i and x_0 a value of x that minimizes $\mu_{A_i}(x)$ over all subsystems i and all x in $\bigcap_{i=1}^m X^i$. Assume that x and z are objects with $x \in \bigcap_i X^i$, and $\mu_{F_t(A_1, ..., A_m)}(z) > \mu_{F_t(A_1, ..., A_m)}(x)$. Then $z \in X^{i_0}$.*

Proof. For ease in notation, let us write $F_t(A_1, ..., A_m)$ as Q. Since t is min, the fact that $\mu_Q(z) > \mu_Q(x)$ says that $\min\{\mu_{A_1}(z), ..., \mu_{A_m}(z)\} > \min\{\mu_{A_1}(x), ..., \mu_{A_m}(x)\}$. By definition of i_0 and x_0, it follows that $\min\{\mu_{A_1}(x), ..., \mu_{A_m}(x)\} \geqslant \mu_{A_{i_0}}(x_0)$. So $\min\{\mu_{A_1}(z), ..., \mu_{A_m}(z)\} > \mu_{A_{i_0}}(x_0)$, 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 $\mathscr{A}_0$, when t is min. The idea is as follows. Let Q denote the query $F_t(A_1, ..., A_m)$, when t is the min. Let i_0 and x_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 $\mathscr{A}_0$, 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 $\mu_{A_{i_0}}(x) \geqslant g_0$. We use the word "candidates," since these turn out to be the only objects we need to consider for the top k objects. Define algorithm $\mathscr{A}'_0$ to be the result of replacing all occurrences of $\bigcup_{i=1}^m X_T^i$ in algorithm $\mathscr{A}_0$ by the set of candidates. Thus, algorithm $\mathscr{A}'_0$ is defined by taking the sorted access phase to be the same as the sorted access phase of algorithm $\mathscr{A}_0$, and taking the remaining two phases as follows:

Random access phase: 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 $\mu_{A_{i_0}}(x) \geqslant g_0$. For each candidate x, do random access to each subsystem $j \neq i_0$ to find $\mu_{A_j}(x)$.

Computation phase: Compute the grade $\mu_Q(x) = \min\{\mu_{A_1}(x), ..., \mu_{A_m}(x)\}$ 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 $\mathscr{A}'_0$ has better performance than $\mathscr{A}_0$, since we do random access only for the candidates, each of which is a member of $X_T^{i_0}$, rather than for all of $\bigcup_{i=1}^m X_T^i$. The next theorem shows that algorithm $\mathscr{A}'_0$ gives the correct answer when t is min.

THEOREM 4.4. *In the case off standard fuzzy conjunction* (*where the aggregation function t is min*), *algorithm* $\mathscr{A}'_0$ *correctly returns the top k answers.*

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

For certain monotone aggregation functions t, we can define an algorithm that performs substantially better than algorithm $\mathscr{A}_0$ (whereas algorithm $\mathscr{A}'_0$ performs better than algorithm $\mathscr{A}_0$ by only a constant factor). 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 $A_1 \vee \cdots \vee A_m$. 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 $\mathscr{B}_0$) that returns the top k answers for the standard fuzzy disjunction $A_1 \vee \cdots \vee A_m$ of atomic queries $A_1, ..., A_m$. Algorithm $\mathscr{B}_0$ has only two phases: a sorted access phase and a computation phase.

Sorted access phase: 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.

Computation phase: For each $x \in \bigcup_{i=1}^m X_k^i$, let

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

Let Y be a set containing the k members x of $\bigcup_{i=1}^m X_k^i$ 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 aggregation function t is max*), *algorithm* $\mathscr{B}_0$ *correctly returns the top k answers.*

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

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 $\mathscr{A}_0$ when the aggregation function is monotone.

Our measure of cost corresponds intuitively to the cost to a middleware system such as Garlic of processing information passed to it from a database subsystem such as QBIC. The *sorted access cost* is 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 total number of objects obtained from the database under random access.

Let S be the sorted access cost, and let R be the random access cost. We take the *middleware cost* to be $c_1 S + c_2 R$, for some positive constants c_1 and c_2. The fact that c_1 and c_2 may be different reflects the fact that the cost to a middleware system of a sorted access and of a random access may be different. Later, we will sometimes find it convenient to work with $S + R$, the sum of the sorted access cost and the random access cost. This is the total number of elements retrieved by the middleware system. We may refer to this as the *unweighted middleware cost*, since it

corresponds to taking $c_1 = c_2 = 1$.[6] The middleware cost and the unweighted middleware cost are within constant multiples of each other, since

$$(\max\{c_1, c_2\}) \cdot (S + R) \geqslant c_1 S + c_2 R \geqslant (\min\{c_1, c_2\}) \cdot (S + R). \quad (1)$$

We note that the middleware cost as we have defined it is probably not a good reflection of the total system cost, since it does not accurately account for the costs inside of a "black box" like QBIC. Indeed, there are situations (such as in the case of a query optimizer) where we want a more comprehensive cost measure. Finding such cost measures is an interesting open problem. We believe that our algorithm $\mathscr{A}_0$ is sufficiently robust that it may well turn out to be optimal even under more comprehensive cost measures.

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. We shall prove that if the atomic queries $A_1, ..., A_m$ are independent, then with arbitrarily high probability, the cost of algorithm $\mathscr{A}_0$ for evaluating $F_t(A_1, ..., A_m)$ 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 $\mathscr{A}_0$ is of the order of the square root of the database size. We now define our terms, to make these statements more precise. We consider the following formal framework.

Let us assume that the database contains N objects, which, for ease in notation, we call 1, ..., N. We are considering a scenario involving m atomic queries $A_1, ..., A_m$ over the database. 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 $i = 1, ..., m$) a graded set, where the objects being graded are 1, ..., 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 $i = 1, ..., m$) a permutation of 1, ..., N. A scoring database $\mathscr{D}$ is *consistent with* skeleton $\mathscr{S}$ if for each i, the ith permutation in $\mathscr{S}$ gives a sorting of the ith graded set of $\mathscr{D}$ (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 middleware cost of algorithms that find the top k answers for $F_t(A_1, ..., A_m)$. 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 $\mathscr{A}$ is an algorithm, $\mathscr{D}$ is a scoring database, and $\mathscr{S}$ is a skeleton such that $\mathscr{D}$ is consistent with $\mathscr{S}$, we define $sortedcost(\mathscr{A}, \mathscr{D}, \mathscr{S})$ to be the sorted access cost of algorithm $\mathscr{A}$ when applied to scoring database $\mathscr{D}$ provided sorted access goes according to skeleton $\mathscr{S}$. We define $sortedcost(\mathscr{A}, \mathscr{S})$ to be the maximum of $sortedcost(\mathscr{A}, \mathscr{D}, \mathscr{S})$ over all scoring databases $\mathscr{D}$ that are consistent with $\mathscr{S}$.[7] Thus, the sorted access cost of an algorithm over a skeleton is the worst-case sorted access cost of the algorithm over all scoring databases consistent with the skeleton. Similarly, we define $cost(\mathscr{A}, \mathscr{S})$ and $cost(\mathscr{A}, \mathscr{D}, \mathscr{S})$ to be the corresponding middleware cost, and $sumcost(\mathscr{A}, \mathscr{S})$ and $sumcost(\mathscr{A}, \mathscr{D}, \mathscr{S})$ to be the corresponding unweighted middleware cost (which is the sum of the sorted access cost and the random access cost). We note for later use that it follows from (1) that

$$(\max\{c_1, c_2\}) \cdot sumcost \geqslant cost(\mathscr{A}, \mathscr{S}) \geqslant (\min\{c_1, c_2\}) \cdot sumcost(\mathscr{A}, \mathscr{S}). \quad (2)$$

Note that if a scoring database $\mathscr{D}$ is consistent with more than one skeleton, then the specification of algorithm $\mathscr{A}$ says that $\mathscr{A}$ gives the top k answers with input $\mathscr{D}$ 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 middleware 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 middleware cost for algorithm $\mathscr{A}_0$ 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

[6] In an earlier version of this paper, we took the unweighted middleware cost, rather than the middleware cost, to be our measure of cost, and we referred to it as the *database access cost*.

[7] An algorithm $\mathscr{A}$ might behave differently over two databases $\mathscr{D}$ and $\mathscr{D}'$ with the same skeleton $\mathscr{S}$. 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.

of possible skeletons (namely, $mN!$), and under an algorithm $\mathscr{A}$, each such skeleton $\mathscr{S}$ has middleware cost $cost(\mathscr{A}, \mathscr{S})$ as defined above. When we say that the atomic queries are independent and then consider probabilities of middleware costs, we mean that we are taking each such skeleton to have equal probability. This is equivalent to the assumption that each of the m sorted lists (one for each atomic query) contains the objects in random order (in other words, each permutation of 1, ..., N has equal probability), independent of the other lists. When we say that for our algorithm $\mathscr{A}_0$, "with arbitrarily high probability the middleware cost is $O(N^{(m-1)/m}k^{1/m})$," we mean that for every $\varepsilon > 0$, there is a constant c such that for every N,

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

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

Before we can prove a theorem on the cost of algorithm $\mathscr{A}_0$, we need a lemma. In this lemma, when we say that B_2 is a random set of ℓ_2 members of $\{1, ..., N\}$, we mean that all subsets of $\{1, ..., N\}$ of cardinality ℓ_2 are selected with equal probability. Before we state and prove the lemma, let us explain how it will be used. In the sorted access phase of algorithm $\mathscr{A}_0$, 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 $\bigcap_{i=1}^m X_T^i$ 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 $\bigcap_{i=1}^m X_\tau^i$, and
2. the probability that the size of $\bigcap_{i=1}^m X_\tau^i$ 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 $1 \leqslant j \leqslant m$, the expected size of $\bigcap_{i=1}^j X_\tau^i$, and the probability that the size of $\bigcap_{i=1}^j X_\tau^i$ is at most half 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 $\{1, ..., N\}$ with ℓ_2 randomly selected members of $\{1, ..., N\}$, and the probability that the size of this intersection is at most half the expected size. This 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 $\ell_1/N \leqslant 1/10$). We denote the size of B by $|B|$.

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

Proof. Let $B_1 = \{b_1, ..., b_{\ell_1}\}$. Let Z_i be a random variable whose value is 1 if $b_i \in B_2$, and 0 otherwise, for $1 \leqslant \ell \leqslant \ell_1$. The expected value $E[B_1 \cap B_2] = E[\sum_{i=1}^{\ell_1} Z_i] = \sum_{i=1}^{\ell_1} E[Z_i] = \sum_{i=1}^{\ell_1} \Pr[b_i \in B_2] = \sum_{i=1}^{\ell_1} \ell_2/N = \ell_1\ell_2/N = M$.

Let $a = \lfloor M/2 \rfloor$. In order to estimate the probability that the size of B is at most $M/2$, we define below four probabilistic processes involving flipping coins. We then show the following.

A. $\Pr[|B| \leqslant M/2] = \Pr[$There are at most a heads in process (1)$]$.

B. $\Pr[$There are at most a heads in process (1)$] = \Pr[$There are at most a heads in process (2)$]$.

C. $\Pr[$There are at most a heads in process (2)$] \leqslant \Pr[$There are at most a heads in process (3)$]$.

D. $\Pr[$There are at most a heads in process (3)$] \leqslant \Pr[$There are at most a heads in process (4)$]$.

E. $\Pr[$There are at most a heads in process (4)$] < e^{-M/10}$.

It is clear that statements (A), (B), (C), (D), and (E) taken together imply that

$$\Pr[|B| \leqslant M/2] < e^{-M/10},$$

as desired. We now define the following four probabilistic processes.

Process 1: ℓ_1 coins are flipped, one after the other. The probability that the first coin is heads is ℓ_2/N. Assuming inductively that so far there have been h heads and t tails, the probability that the next coin is heads is $\max\{(\ell_2 - h)/(N - h - t), 0\}$.

Process 2: ℓ_1 coins are flipped, one after the other. The probability that the first coin is heads is ℓ_2/N. Assuming inductively that so far there have been h heads and t tails, the probability that the next coin is heads is

$$\max\{(\ell_2 - h)/(N - h - t), (\ell_2 - a)/(N - a)\}.$$

(Note that unlike the situation in Process 1, we do not need to include 0 in the set that we take the max over, because $\ell_2 - a \geqslant \ell_2 - \frac{1}{2}(\ell_1/N)\,\ell_2 \geqslant 0$.)

Process 3: ℓ_1 coins are flipped, one after the other. For every flip, the probability that the coin is heads is $(\ell_2 - a)/(N - a)$.

Process 4: ℓ_1 coins are flipped, one after the other. For every flip, the probability that the coin is heads is $\frac{19}{20}\ell_2/N$.

Proof of (A). We can calculate $\Pr[|B| \leqslant M/2]$ by considering each of the ℓ_1 members of B_1 one by one; the probability that the jth member of B_1 is in B_2, given that among the first $j-1$ members of B_1 it happens that h members

are in B_2 and t are not, is $\max\{(\ell_2-h)/(N-h-t), 0\}$. We think of the event of the jth member of B_1 being a member of B_2 as corresponding to the jth coin in process (1) coming up heads. It is then straightforward to see that

$$\begin{aligned}&\Pr[\,|B| \leqslant M/2\,]\\ &\quad = \Pr[\text{ There are at most } M/2 \text{ heads in process (1)}]\\ &\quad = \Pr[\text{ There are at most } a \text{ heads in process (1)}].\end{aligned}$$

Proof of (*B*). We first show that if $h \leqslant a$, then $(\ell_2-a)/(N-a) \leqslant (\ell_2-h)/(N-h)$. Assume that $h \leqslant a$. When the numerator is less than or equal to the denominator, as it is in $(\ell_2-h)/(N-h)$, and we subtract the same nonnegative number (in this case, $a-h$) from the numerator and the denominator, the value of the fraction cannot increase (since the numerator decreases by at least as big a percentage as the denominator does). Therefore indeed, if $h \leqslant a$, then

$$(\ell_2-a)/(N-a) \leqslant (\ell_2-h)/(N-h).$$

Furthermore,

$$(\ell_2-h)/(N-h) \leqslant (\ell_2-h)/(N-h-t).$$

Therefore, if $h \leqslant a$, then

$$(\ell_2-a)/(N-a) \leqslant (\ell_2-h)/(N-h-t).$$

Hence, if $h \leqslant a$, then the probability

$$\max\{(\ell_2-h)/(N-h-t), 0\}$$

in process (1) equals the probability

$$\max\{(\ell_2-h)/(N-h-t), (\ell_2-a)/(N-a)\}$$

in process (2). It follows that if we write the combinatorial sum for process (1), where we sum over the probabilities of all possible ways that the number of heads is at most a, then we get the same answer as in the corresponding combinatorial sum for process (2). So

$$\begin{aligned}&\Pr[\text{ There are at most } a \text{ heads in process (1)}]\\ &\quad = \Pr[\text{ There are at most } a \text{ heads in process (2)}],\end{aligned}$$

as desired.

Proof of (*C*). This follows immediately from the fact that the probability

$$\max\{(\ell_2-h)/(N-h-t), (\ell_2-a)/(N-a)\}$$

in process (2) is greater than or equal to the probability

$$(\ell_2-a)/(N-a)$$

in process (3).

Proof of (*D*). We need only show that the probability $(\ell_2-a)/(N-a)$ of process (3) is at least as big as the probability $\frac{19}{20}\ell_2/N$ of process (4). Now $a = \lfloor M/2 \rfloor \leqslant M/2 = \frac{1}{2}(\ell_1/N)\,\ell_2 \leqslant \frac{1}{2}\,\frac{1}{10}\,\ell_2 = \ell_2/20$. Hence, $(\ell_2-a)/(N-a) \geqslant \frac{19}{20}\ell_2/(N-a) \geqslant \frac{19}{20}\ell_2/N$, as desired.

Proof of (*E*). We can estimate Pr[there are at most a heads in process (4)] with a Chernoff bound. By results of [AV79] (see also [HR90]), if the probability of heads is p, and there are ℓ_1 flips of the coin, then the probability that there are at most $(1-\varepsilon)\,n$ heads, where $n = \ell_1\,p$ is the expected number of heads, is at most $e^{-\varepsilon^2 n/2}$. Let $p = \frac{19}{20}\ell_2/N$. So the expected number $n = \ell_1\,p$ of heads is $\frac{19}{20}\ell_1\ell_2/N = \frac{19}{20}\,M$. We are interested in the probability that there are at most $a = \lfloor M/2 \rfloor$ heads. How big is ε, if $a = (1-\varepsilon)\,n$? From what we have shown, we have $\varepsilon = 1-(a/n) \geqslant 1-(\frac{1}{2}\,M)/(\frac{19}{20}\,M) = 1-\frac{20}{19}\,\frac{1}{2} = \frac{9}{19}$. Therefore, the Chernoff bound $e^{-\varepsilon^2 n/2}$ is at most $e^{-(9/19)^2\,(19/20)\,M/2} < e^{-M/10}$, as desired. ∎

We are almost ready to state and prove a theorem about the performance cost of algorithm $\mathscr{A}_0$. 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 (the number of objects in the database), even though we suspect that users are most interested in the case where k is a small constant (like 10). Note also that the middleware 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 middleware cost is $O(\sqrt{N})$.

Remark 5.2. A few comments are in order about the extreme case where $k = N$, since it must be dealt with specially in our lower bound proofs in the next section. In this case, we certainly know *a priori* the top k objects (the k objects with the highest 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 answers"), it is clearly necessary in general to access every entry in the database. Note that in this extreme case, the middleware cost $O(N^{(m-1)/m}k^{1/m})$ as claimed in Theorem 5.3 is simply $O(N)$, as we would expect.

In Theorem 5.3, we determine the middleware cost for algorithm $\mathscr{A}_0$. In this theorem, we do not need to assume that $F_t(A_1, ..., A_m)$ is monotone. Monotonicity arises in correctness, not performance: the algorithm $\mathscr{A}_0$ is guaranteed to be correct only when $F_t(A_1, ..., A_m)$ is monotone (Theorem 4.2).

THEOREM 5.3. *Assume that the m atomic queries are independent. The middleware cost for algorithm $\mathscr{A}_0$ is $O(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability.*

Proof. By our comments above, we need only prove the theorem in a model where each of the m sorted lists contains the objects in random order, independent of the other lists. Therefore, we make this assumption. Since the middleware cost is bounded above by a constant times the unweighted middleware cost (by the first inequality of (2)), we need only prove the statement in the theorem when we replace "middleware cost" by "unweighted middleware cost."

Let $c \geqslant 2$ be a constant, and let $T = \lceil cN^{(m-1)/m}k^{1/m} \rceil$. There are two cases, depending on whether $T/N > 1/10$ or $T/N \leqslant 1/10$. Assume first that $T/N > 1/10$. The algorithm $\mathscr{A}_0$ certainly has unweighted middleware cost at most mN, since no object is accessed more than m times (once for every list). Since $T/N > 1/10$, it follows that with probability 1, the unweighted middleware cost of $\mathscr{A}_0$ is at most $mN < 10mT < 10m(cN^{(m-1)/m}k^{1/m} + 1)$, which is at most $(10m+1)cN^{(m-1)/m}k^{1/m}$ for c sufficiently large.

Therefore, for the rest of the proof, we can assume that $T/N \leqslant 1/10$. We now show that it is sufficient to show that $\Pr[\,|\bigcup_{i=1}^m X_T^i| \geqslant k\,]$ converges to 1 as c goes to infinity. We need only show that this implies that

$$\Pr_{\mathscr{S}}\,[\,sumcost(\mathscr{A}_0, \mathscr{S}) > (2cm^2)(N^{(m-1)/m}k^{1/m})\,] \qquad (3)$$

goes to 0 as c goes to infinity. Now (3) is bounded above by

$$\Pr_{\mathscr{S}}\,[\,sortedcost(\mathscr{A}_0, \mathscr{S}) > (2cm)(N^{(m-1)/m}k^{1/m})\,] \qquad (4)$$

since the unweighted middleware cost is at most m times the sorted access cost. Furthermore, (4) is bounded above by

$$\Pr_{\mathscr{S}}\,[\,sortedcost(\mathscr{A}_0, \mathscr{S}) > mT\,]; \qquad (5)$$

this follows fairly straightforwardly from the fact that if $\alpha > 1$, then $2\alpha > \lceil \alpha \rceil$. But (5) is bounded above by $\Pr[\,|\bigcap_{i=1}^m X_T^i| < k\,]$, since if the sorted access cost is greater than mT, then the sorted access cost is greater than T for each of the m lists, which implies that $|\bigcap_{i=1}^m X_T^i| < k$. So indeed, it is sufficient to show that $\Pr[\,|\bigcap_{i=1}^m X_T^i| \geqslant k\,]$ converges to 1 as c goes to infinity.

Define d_j to be $cN^{(m-j)/m}k^{j/m}$, for $1 \leqslant j \leqslant m$. In particular, $T = \lceil d_1 \rceil$, so each X_T^i has $\lceil d_1 \rceil$ members. Furthermore, $d_m = ck$. For $1 \leqslant j \leqslant m$, let W_j be the event that $|\bigcap_{i=1}^j X_T^i| < d_j$, we now show by induction on j that

$$\Pr[\,W_j\,] \leqslant \sum_{i=2}^{j} e^{-d_i/5}. \qquad (6)$$

The base case $j = 1$ is immediate, since the probability that $\lceil d_1 \rceil$ is less than d_1 is 0. For the inductive step, assume that $j < m$, and that (6) holds. Let ℓ_1 denote the size of $\bigcap_{i=1}^j X_T^i$. By assumption $T/N \leqslant 1/10$, so $\ell_1/N \leqslant T/N \leqslant 1/10$. Let $M = \ell_1 T/N$. By Lemma 5.1,

$$\Pr\left[\,\left|\bigcap_{i=1}^{j+1} X_T^i\right| < M/2\,\right] < e^{-M/10}. \qquad (7)$$

Let $M' = d_j(2N^{(m-1)/m}k^{1/m})/N = 2d_{j+1}$.

Let us now evaluate $\Pr[\,W_{j+1} \wedge \neg W_j\,]$. We will use the fact that

$$T \geqslant cN^{(m-1)/m}k^{1/m} \geqslant 2N^{(m-1)/m}k^{1/m}.$$

Now $\neg W_j$ is the event that $\ell_1 \geqslant d_j$. So if $\neg W_j$ holds, then

$$M = \ell_1 T/N \geqslant d_j T/N \geqslant d_2(2N^{(m-1)/m}k^{1/m})/N = M'. \qquad (8)$$

Now W_{j+1} is the event that $|\bigcap_{i=1}^{j+1} X_T^i| < d_{j+1} = M'/2$. So by (8), $W_{j+1} \wedge \neg W_j$ is a subset of the event that $|\bigcap_{i=1}^{j+1} X_T^i| < M/2$, which by (7) has probability less than $e^{-M/10}$. Since $\neg W_j$ implies by (8) that $M \geqslant M'$, it follows that

$$\Pr[\,W_{j+1} \wedge \neg W_j\,] < e^{-M'/10} = e^{-d_{j+1}/5}. \qquad (9)$$

Therefore,

$$\begin{aligned}\Pr[\,W_{j+1}\,] &= \Pr[\,W_{j+1} \wedge W_j\,] + \Pr[\,W_{j+1} \wedge \neg W_j\,] \\ &\leqslant \Pr[\,W_j\,] + \Pr[\,W_{j+1} \wedge \neg W_j\,] \\ &\leqslant \sum_{i=2}^{j+1} e^{-d_i/5},\end{aligned}$$

where the last inequality follows from (6) and (9). This completes the inductive step (and in fact shows that the inequality in (6) is strict for $j \geqslant 2$).

By taking $j = m$ in (6), it follows that

$$\Pr\left[\,\left|\bigcap_{i=1}^{m} X_T^i\right| < d_m = ck\,\right] \leqslant \sum_{i=2}^{m} e^{-d_i/5}. \qquad (10)$$

Since $c \geqslant 2$, we see from (10) that

$$\Pr\left[\,\left|\bigcap_{i=1}^{m} X_T^i\right| < k\,\right] \leqslant \sum_{i=2}^{m} e^{-d_i/5}. \qquad (11)$$

By taking c sufficiently large, the probability on the right-hand side of (11) can be made arbitrarily small. More precisely, for each $\varepsilon > 0$, there is c such that for every $N \geqslant 1$ and every $k \geqslant 1$, this probability is less than ε. Therefore, $\Pr[\,|\bigcap_{i=1}^m X_T^i| \geqslant k\,]$ converges to 1 as c goes to infinity. ∎

In the previous proof we see from (11) that the probability that more than $cN^{(m-1)/m}k^{1/m}$ objects are accessed by sorted access in each list is at most $\sum_{i=2}^{m} e^{-d_i/5}$. It is easy to see that all terms except the last term $e^{-ck/5}$ are negligible even for moderate-sized N (such as $N \geqslant 100$), because of the involvement of N in the exponent of these earlier terms. Thus, the dominant term is $e^{-ck/5}$. We note that Wimmers [Wi98b] has done a more refined analysis than ours in the case when $m = 2$ (that is, when there are exactly two atomic queries), and thereby obtained an improved upper bound on the probability that more than $c\sqrt{Nk}$ objects are accessed by sorted access in each list. His improved upper bound has dominant term e^{-c^2k}. Wimmers' upper bound is less than 2×10^{-8} if $c = 2$, and less than 4×10^{-27} if $c = 3$. Thus, the probability is less than 2×10^{-8} that more than $2\sqrt{Nk}$ objects are accessed by sorted access in each list, and less than 4×10^{-27} that more than $3\sqrt{Nk}$ objects are accessed by sorted access in each list.

From Theorems 4.2 and 5.3, we immediately obtain the following theorem.

THEOREM 5.4. *There is an algorithm for finding the top k answers to each monotone query $F_t(A_1, ..., A_m)$, where $A_1, ..., A_m$ are independent, with middleware cost $O(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability.*

Monotone queries are an important class. For example, any natural notion of conjunction should be monotone. Theorem 5.4 guarantees an efficient algorithm for monotone queries (when the atomic queries are independent).

6. LOWER BOUNDS

Theorem 5.4 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 of $\Omega(N^{(m-1)/m}k^{1/m})$ for strict queries. Thus, we show that in the case of strict queries, no correct algorithm $\mathscr{A}$ that finds the top k answers can do better. Our results say that for such an algorithm $\mathscr{A}$ and for each N and each $\theta \geqslant 0$,

$$\Pr_{\mathscr{S}}[cost(\mathscr{A}, \mathscr{S}) \leqslant (\min\{c_1, c_2\})\, \theta N^{(m-1)/m}k^{1/m}] \leqslant \theta^m,$$

where c_1, c_2 are as in the definition of the middleware cost. Hence, there is no function f of N with $f = o(N^{(m-1)/m}k^{1/m})$ such that if the atomic queries are independent, then with arbitrarily high probability the middleware cost for algorithm $\mathscr{A}$ is $O(f)$. Therefore, the middleware cost $O(N^{(m-1)/m}k^{1/m})$ of algorithm $\mathscr{A}_0$ is optimal.

Remark 6.1. To prove our lower bound of

$$\Omega(N^{(m-1)/m}k^{1/m}), \tag{12}$$

we need to assume that the aggregation function t is strict. Note that max is *not* strict. In fact, in the case of max, the lower bound (12) fails. Algorithm $\mathscr{B}_0$ of Theorem 4.5 has middleware cost only mk, independent of the size N of the database!

Another aggregation function that is not strict is the median. Again, our lower bound fails in this case. For example, assume that $m = 3$, so that we wish to consider the query that evaluates median $(\mu_{A_1}(x), \mu_{A_2}(x), \mu_{A_3}(x))$ for each object x. We now give an algorithm that finds the top k answers to this query. The algorithm is based on the fact (which we leave to the reader to verify) that

$$\text{median}(a_1, a_2, a_3) = \max\{\min\{a_1, a_2\}, \min\{a_1, a_3\}, \min\{a_2, a_3\}\}. \tag{13}$$

We describe the algorithm informally as follows.

1. Find the top k answers for the query that evaluates $\min\{\mu_{A_1}(x), \mu_{A_2}(x)\}$ for each object x, by using algorithm $\mathscr{A}_0$. Let $X_{1,2}$ be a set containing k objects with the highest scores.
2. Find the top k answers for the query that evaluates $\min\{\mu_{A_1}(x), \mu_{A_3}(x)\}$ for each object x, by using algorithm $\mathscr{A}_0$. Let $X_{1,3}$ be a set containing k objects with the highest scores.
3. Find the top k answers for the query that evaluates $\min\{\mu_{A_2}(x), \mu_{A_3}(x)\}$ for each object x, by using algorithm $\mathscr{A}_0$. Let $X_{2,3}$ be a set containing k objects with the highest scores.
4. Output the k objects in $X_{1,2} \cup X_{1,3} \cup X_{2,3}$ with the highest median scores, along with these scores.

It is not hard to see that (13) implies the correctness of the algorithm. This algorithm has middleware cost $O(\sqrt{Nk})$, with arbitrarily high probability, and so the lower bound (12) with $m = 3$ fails.

An interesting example of an aggregation function that is not strict and that is based on an aggregation function that arises in "real life" occurs when assigning scores in (artistic) gymnastics. There are a number of judges, each of whom assigns a score; the top and bottom scores are eliminated, and the remaining scores are averaged.[8] The corresponding aggregation function is not strict. If there are three judges, then this aggregation function is simply the median, so the lower bound (12) with $m = 3$ fails.

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 unweighted middleware cost is at least

[8] In this paper, we have defined aggregation function to have range $[0, 1]$. For scoring in actual gymnastics, however, this is not the case: a perfect score in gymnastics is 10. A more complete discussion of the scoring rules for gymnastics is currently available at http://www.usa-gymnastics.org/gymnastics/scoring/artistic.html.

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 6.2. *Assume that t is strict. Let $\mathscr{S}$ be a skeleton on N objects, and let $\mathscr{A}$ be an arbitrary algorithm that finds the top k answers to $F_t(A_1, ..., A_m)$. Assume that $sumcost(\mathscr{A}, \mathscr{S}) < N$, and $T \geqslant sortedcost(\mathscr{A}, \mathscr{S})$. Let X^i_T denote the top T objects in list i according to skeleton $\mathscr{S}$. Then $\bigcap_{i=1}^m X^i_T$ contains at least k members.*

Proof. The case $k = N$ is a special case, as we discussed in Remark 5.2. In this case, the unweighted middleware cost is $mN \geqslant N$, and so the hypothesis $sumcost(\mathscr{A}, \mathscr{S}) < N$ fails.

Therefore, assume that $k < N$. Assume that $sumcost(\mathscr{A}, \mathscr{S}) < N$, and $T \geqslant sortedcost(\mathscr{A}, \mathscr{S})$, and that $\bigcap_{i=1}^m X^i_T$ contains less than k members; we shall derive a contradiction. Specifically, we shall construct a skeleton $\mathscr{S}'$ and a scoring database $\mathscr{D}'$ such that $\mathscr{D}'$ is consistent with $\mathscr{S}'$, and when the algorithm $\mathscr{A}$ is run against scoring database $\mathscr{D}'$ and sees skeleton $\mathscr{S}'$, it gives the wrong answer.

We first define a scoring database $\mathscr{D}$ that is consistent with skeleton $\mathscr{S}$ as follows: for each list i (with $1 \leqslant i \leqslant m$), the grades in list i of members of X^i_T are all 1, and the grades in list i of the remaining members of list i are all 0.

Let us say that the grade of object x in list i is *determined* if the grade of x in list i is obtained by algorithm $\mathscr{A}$ when applied to scoring database $\mathscr{D}$ where sorted access goes according to skeleton $\mathscr{S}$. Otherwise, we say that the grade of object x in list i is *undetermined*. We say that x is *untouched* if the grade of object x in list i is undetermined for every list i. Intuitively, x is untouched if the algorithm $\mathscr{A}$ does not determine any information about any of x's grades. Since by assumption $sumcost(\mathscr{A}, \mathscr{S}) < N$, there is some object x_0 that is untouched.

Define scoring database $\mathscr{D}'$ to be the same as scoring database $\mathscr{D}$, except that in $\mathscr{D}'$, the grade of x_0 is 1 in every list. Since t is strict, x_0 and the members of $\bigcap_{i=1}^m X^i_T$ all have grade 1 under the query $F_t(A_1, ..., A_m)$ in $\mathscr{D}'$, and no other objects in $\mathscr{D}'$ have grade 1. Since by assumption $\bigcap_{i=1}^m X^i_T$ contains less than k members, necessarily x_0 is one of the top k objects in scoring database $\mathscr{D}'$ according to the query.

Let $\mathscr{S}'$ be a skeleton such that $\mathscr{D}'$ is consistent with $\mathscr{S}'$ and such that the top T objects in each list according to skeleton $\mathscr{S}'$ is the same as the top T objects in each list according to skeleton $\mathscr{S}$ (we could let x_0 be the $(T+1)$th member of each list). By correctness of the algorithm $\mathscr{A}$, it should give the top k answers when run against scoring database $\mathscr{D}'$ when the algorithm sees skeleton $\mathscr{S}'$, and in particular must give x_0 with its grade of 1 as one of the top k objects. But when the algorithm is run against scoring database $\mathscr{D}'$ and the algorithm sees skeleton $\mathscr{S}'$, then the algorithm will see exactly the same information as when it is run against scoring database $\mathscr{D}$ and sees skeleton $\mathscr{S}$. In this latter case, we know that x_0 and its grade are not given as one of the top k answers, since the algorithm does not even see x_0 and its grades in any list. So the same is true when the algorithm is run against scoring database $\mathscr{D}'$ and sees skeleton $\mathscr{S}'$. This is a contradiction. ∎

Remark 6.3. To help show the subtleties involved in Lemma 6.2, we note that in contrast to Lemma 6.2, it is possible that $sumcost(\mathscr{A}, \mathscr{D}, \mathscr{S}) < N$ and $T > sortedcost(\mathscr{A}, \mathscr{D}, \mathscr{S})$ (as opposed to $sumcost(\mathscr{A}, \mathscr{S}) < N$ and $T > sortedcost(\mathscr{A}, \mathscr{S})$) even though $\bigcap_{i=1}^m X^i_T$ contains less than k members. Thus, Lemma 6.2 tells us that if

$$sumcost(\mathscr{A}, \mathscr{S}) = \max\{sumcost(\mathscr{A}, \mathscr{D}, \mathscr{S}) \mid \mathscr{D} \text{ is consistent with } \mathscr{S}\} < N$$

and

$$sortedcost(\mathscr{A}, \mathscr{S}) = \max\{sortedcost(\mathscr{A}, \mathscr{D}, \mathscr{S}) \mid \mathscr{D} \text{ is consistent with } \mathscr{S}\} \leqslant T$$

then $\bigcap_{i=1}^m X^i_T$ contains at least k members. It is still possible that here could be a specific scoring database $\mathscr{D}$ consistent with skeleton $\mathscr{S}$ such that $sumcost(\mathscr{A}, \mathscr{D}, \mathscr{S}) < N$ and $sortedcost(\mathscr{A}, \mathscr{D}, \mathscr{S}) \leqslant T$ even though $\bigcap_{i=1}^m X^i_T$ contains less than k members. In fact, this could happen even when t is min. For example, assume that the top object in the first list is x, and that x has grade 0.9 in every list. A single sorted access to the first list tells us that no object can have (overall) grade greater than 0.9, and random access to the other lists tells us that x has grade 0.9. Therefore, we have determined that x is the top answer (and if another sorted access to the first list tells us that the next object in the first list has grade 0.8, then we would even know that x is the unique top object, although we are not requiring this information). Thus, in this situation we are guaranteed that the top object is found with a single sorted access (and, we might add, only a small number of random accesses). This could happen even if $\bigcap_{i=1}^m X^i_T$ is empty for some fairly large T.

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

THEOREM 6.4. *Let N be given. Assume that t is strict. Let $\mathscr{A}$ be an arbitrary algorithm that finds the top k answers to $F_t(A_1, ..., A_m)$. Let c_1, c_2 be as in the definition of the middleware cost. If $A_1, ..., A_m$ are independent, then*

$$\Pr_{\mathscr{S}}[cost(\mathscr{A}, \mathscr{S}) \leqslant (\min\{c_1, c_2\})\, \theta N^{(m-1)/m} k^{1/m}] \leqslant \theta^m,$$

for every $\theta \geqslant 0$.

Proof. It follows from the second inequality of (2) in Section 5 that if $cost(\mathscr{A}, \mathscr{S}) \leqslant (\min\{c_1, c_2\})\, \theta N^{(m-1)/m} k^{1/m}$,

then $sumcost(\mathscr{A}, \mathscr{S}) \leqslant \theta N^{(m-1)/m} k^{1/m}$. Therefore, it is sufficient to show that

$$\Pr_{\mathscr{S}} [sumcost(\mathscr{A}, \mathscr{S}) \leqslant \theta N^{(m-1)/m} k^{1/m}] \leqslant \theta^m.$$

Assume that $\theta < 1$, since otherwise the result is trivial. The case $k = N$ is a special case, as we discussed in Remark 5.2. In this case, the unweighted middleware cost is $mN \geqslant N$. But then $\theta N^{(m-1)/m} k^{1/m} = \theta N < N$. So

$$\Pr_{\mathscr{S}} [sumcost(\mathscr{A}, \mathscr{S}) \leqslant \theta N^{(m-1)/m} k^{1/m}] = 0.$$

Therefore, assume that $k < N$. Let $T = \lfloor \theta N^{(m-1)/m} k^{1/m} \rfloor$. Then $T \leqslant \theta N < N$. By Lemma 6.2, we know that if $sumcost(\mathscr{A}, \mathscr{S}) < N$, and $sortedcost(\mathscr{A}, \mathscr{S}) \leqslant T$, then $\bigcap_{i=1}^m X_T^i$ contains at least k members. In particular, if $sumcost(\mathscr{A}, \mathscr{S}) \leqslant T$, then $\bigcap_{i=1}^m X_T^i$ contains at least k members. It follows that

$$\Pr_{\mathscr{S}} [sumcost(\mathscr{A}, \mathscr{S}) \leqslant T] \leqslant \Pr \left[\left| \bigcap_{i=1}^m X_T^i \right| \geqslant k \right]. \quad (14)$$

Let us estimate the probability on the right-hand side of (14). By an argument very similar to that at the beginning of Lemma 5.1, the expected size of $\bigcap_{i=1}^m X_T^i$ is $T(T/N)^{m-1}$. But since $T \leqslant \theta N^{(m-1)/m} k^{1/m}$, it follows easily that $T(T/N)^{m-1} \leqslant \theta^m k$. Since the expected size of $\bigcap_{i=1}^m X_T^i$ is at most $\theta^m k$, we have that

$$\Pr \left[\left| \bigcap_{i=1}^m X_T^i \right| \geqslant k \right] \leqslant \theta^m. \quad (15)$$

From (14) and (15), it follows that

$$\Pr_{\mathscr{S}} [sumcost(\mathscr{A}, \mathscr{S}) \leqslant T] \leqslant \theta^m,$$

as desired. ∎

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 middleware cost for finding the top k answers to $F_1(A_1, ..., A_m)$, where $A_1, ..., A_m$ 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 $\mathscr{A}_0$ for finding the top k answers to $F_1(A_1, ..., A_m)$ such that for every $\varepsilon > 0$, there is a constant c such that for every N,

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

2. For every algorithm $\mathscr{A}$ for finding the top k answers to $F_1(A_1, ..., A_m)$ and for every $\varepsilon > 0$, there is a constant d such that for every N,

$$\Pr_{\mathscr{S}} [cost(\mathscr{A}, \mathscr{S}) < dN^{(m-1)/m} k^{1/m}] < \varepsilon.$$

THEOREM 6.5. *The middleware cost for finding the top k answers to a monotone, strict query $F_1(A_1, ..., A_m)$, where $A_1, ..., A_m$ 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, 5.3, and 6.4. ∎

Intuitively, Theorem 6.5 tells us that we have matching upper and lower bounds for many natural notions of conjunction, such as all triangular norms.

We close this section by giving a variation of Theorem 6.4 that focuses on the sorted access cost. This theorem shows that the sorted access cost of our algorithm $\mathscr{A}_0$ is essentially optimal. Such a result must have some restriction on the random access cost, since the naive algorithm mentioned in Section 4 can be implemented with 0 sorted access cost, by accessing every object under random access in each subsystem.

THEOREM 6.6. *Let N be given. Assume that t is strict. Let $\mathscr{A}$ be an arbitrary algorithm that finds the top k answers to $F_1(A_1, ..., A_m)$, with unweighted middleware cost less than N, for every database with N objects. If $A_1, ..., A_m$ are independent, then*

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

for every $\theta \geqslant 0$.

Proof. This follows by a small variation of the proof of Theorem 6.4. ∎

Let us assume that t is monotone and strict. Theorem 6.6 tells us that except for algorithms with an extremely large random access cost (linear in 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 $\mathscr{A}_0$. It might be interesting to consider whether a similar result holds for the random access cost. This would show that our algorithm $\mathscr{A}_0$ is essentially optimal both under sorted access cost and random access cost. This would be stronger than our result that algorithm $\mathscr{A}_0$ is optimal under an arbitrary positive linear combination of these costs, which we obtain from Theorem 6.5.

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 $\mathscr{A}_0$ (upper bound) and our lower bound machinery (in particular, Lemma 6.2), 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 $\bigcap_{i=1}^{m} X_T^i$ contains at least k members. 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. In particular, for the query $Q \wedge \neg Q$, we can restrict our attention to skeletons $\mathscr{S}$ consisting of a permutation π_Q of $1, ..., N$ (for the subquery Q) along with the "reverse" permutation $\pi_{\neg Q}$ (where $\pi_{\neg Q}(x) = \pi_Q(N+1-x)$) for the subquery $\neg Q$. Thus, the top object $\pi_Q(1)$ according to the permutation π_Q is the bottom object $\pi_{\neg Q}(N)$ according to the permutation $\pi_{\neg Q}$.

We now show that the middleware 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 $\mathscr{A}$ for finding the top answer to $Q \wedge \neg Q$, and a constant c, such that for every skeleton $\mathscr{S}$ and every N,

$$cost(\mathscr{A}, \mathscr{S}) < cN.$$

2. For every algorithm $\mathscr{A}$ for finding the top answer to $Q \wedge \neg Q$, there is a constant d such that for every skeleton $\mathscr{S}$ and every N,

$$cost(\mathscr{A}, \mathscr{S}) > dN.$$

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

Proof. The upper bound follows from the naive algorithm described near the beginning of Section 4, which is linear.

For the lower bound, it follows from the second inequality of (2) in Section 5 that it is sufficient to show that if $\mathscr{A}$ is a correct algorithm, $\mathscr{S}$ is a skeleton, and N is the number of objects, then $sumcost(\mathscr{A}, \mathscr{S}) \geqslant N/2$. Assume that $sumcost(\mathscr{A}, \mathscr{S}) < N/2$; we shall derive a contradiction.

We now follow the general outline of the proof of Lemma 6.2, with a few changes. As in the proof of Lemma 6.2, our goal is to construct a scoring database $\mathscr{D}'$ such that when the algorithm $\mathscr{A}$ is run against scoring database $\mathscr{D}'$, it gives the wrong answer.

We first define a scoring database $\mathscr{D}$ that is consistent with skeleton $\mathscr{S}$ as follows: the grades of the first $\lfloor N/2 \rfloor$ objects according to the first permutation in the skeleton (that is, using our earlier notation, the grades of the members of $X^1_{\lfloor N/2 \rfloor}$) are all distinct and strictly greater than $1/2$, and the grades of the remaining objects are all distinct and strictly less than $1/2$.

As in the proof of Lemma 6.2, we define what it means for an object x to be *untouched*, which means intuitively that the algorithm $\mathscr{A}$ does not see x's grades. Since by assumption $sumcost(\mathscr{A}, \mathscr{S}) < N/2 < N$, there is some object x_0 that is untouched.

Define scoring database $\mathscr{D}'$ to be the same as scoring database $\mathscr{D}$, except that in $\mathscr{D}'$, the grade of x_0 is $1/2$ in both lists (this corresponds to the situation where $\mu_Q(x_0) = 1/2 = \mu_{\neg Q}(x_0)$). Since $sortedcost(\mathscr{A}, \mathscr{S}) \leqslant sumcost(\mathscr{A}, \mathscr{S}) < N/2$, it follows that in the course of running the algorithm $\mathscr{A}$ on database $\mathscr{D}$, no information has been seen that is inconsistent with x_0 having grade $1/2$.[9]

Since algorithm $\mathscr{A}$ is correct, this algorithm should determine that x_0 with its grade of $1/2$ is the top object according to the query $Q \wedge \neg Q$ in database $\mathscr{D}'$ (since no other object in $\mathscr{D}'$ has the top grade of $1/2$). But when the algorithm is run against scoring database $\mathscr{D}'$, the algorithm will see exactly the same information as when it is run against scoring database $\mathscr{D}$. In this latter case, we know that x_0 and its grade are not given as the top answer, since the algorithm does not even see x_0 and its grade. So the same is true when the algorithm is run against scoring database $\mathscr{D}'$. This is a contradiction. ∎

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

8. SUBSYSTEMS WITH A DIFFERENT SEMANTICS

Garlic is "on top of" various subsystems and may have no control over the semantics of these subsystems. This may lead to some confusion. For example, what if Garlic is asked

[9] Such an inconsistency would arise if under sorted access, two consecutive objects x_1 and x_2 were seen where x_1 has grade greater than $1/2$ and x_2 has grade less than $1/2$, since then there could be no object x_0 with grade $1/2$.

to evaluate a query $A_1 \wedge A_2$, where one subsystem, say QBIC, is responsible for evaluating both A_1 and A_2? Assume, as is the case currently, that QBIC has a different semantics for conjunction than Garlic. Therefore, if we simply ask QBIC to evaluate the conjunction, we might get different results than if Garlic asks QBIC to separately evaluate A_1 and A_2, with the answers being combined by the Garlic rules.

Perhaps the most natural way to account for this issue is to define two flavors of conjunction, which we could call *internal conjunction* and *external conjunction*.[10] (In fact, there might be one flavor of internal conjunction for every subsystem.) The user could request an internal conjunction for the sake of efficiency. If the user requests an external conjunction, then the external conjunction, which might involve many calls to the subsystem, must be used.

9. EXPLOITING OTHER INFORMATION

We have discussed an algorithm $\mathscr{A}_0$ that works well in evaluating a monotone query $F_1(A_1, ..., A_m)$ when the atomic queries $A_1, ..., A_m$ 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 1 the query A_1 under sorted access. Thus, subsystem 1 begins to output, one by one in sorted order based on grade, the graded set consisting of all pairs $(x, \mu_{A_1}(x))$.

2. As each pair $(x, \mu_{A_1}(x))$ is output from subsystem 1, do random access to subsystem 2 to obtain $\mu_{A_2}(x)$.

3. Stop if and when an object x is found such that $\mu_{A_2}(x) \geqslant \mu_{A_1}(x)$; if such an object x is never found, then continue until all objects have been seen.

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_2 are uniformly distributed in $[0, 1]$, and the maximum value of the grades of the objects under the query A_1 is, say, 0.9. Then the expected time to stop is after at most 10 objects have been seen, independent of the number N of objects in the database. Hence, in this case, the middleware cost is a constant.

[10] This idea was suggested by Ed Wimmers.

But the assumption that the grades of the objects under the query A_1 are bounded above by a constant (such as 0.9) less than 1 is fairly strong. It is interesting to consider the performance of Ullman's algorithm under the assumptions that the atomic queries A_1, A_2 are independent, and that the grades of the objects under both queries A_1 and A_2 are uniformly distributed in $[0, 1]$. We again assume that $k = 1$ (so that we are trying to find only the top answer). Ariel Landau (personal communication) has shown that in this case, the expected time to stop (and hence the middleware cost) is $\Theta(\sqrt{N})$. Thus, in this case the performance is no better than that of our algorithm $\mathscr{A}_0$, since the middleware cost of $\mathscr{A}_0$ under the sole assumption that the atomic queries A_1, A_2 are independent is $\Theta(\sqrt{N})$.

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

10. RELATED WORK

Chaudhuri and Gravano [CG96] consider ways to simulate algorithm $\mathscr{A}_0$ by using "filter conditions," which might say, for example, that the color score is at least 0.2. Wimmers *et al.* [WHTB98] carry out detailed studies on the performance of algorithm $\mathscr{A}_0$ and consider implementation issues (see the author's paper [Fa98] for a discussion).

11. 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 proven 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.

ACKNOWLEDGMENTS

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. Finally, the author thanks Moshe Vardi for suggesting the term "middleware cost."

REFERENCES

[Al85] C. Alsina, On a family of connectives for fuzzy sets, *Fuzzy Sets Syst.* **16** (1985), 231–235.

[AV79] D. Angluin and L. G. Valiant, Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. Syst. Sci.* **18** (1979), 155–193.

[AK98] A. Ait-Bouziad and H. Kassel, An imporved algorithm for retrieving fuzzy information from two systems, *Inform. Process. Lett*., to appear.

[BD86] P. P. Bonissone and K. S. Decker, Selecting uncertainty calculi and granularity: An experiment in trading-off precision and complexity, *in* "Uncertainty in Artificial Intelligence" (L. N. Kanal and J. F. Lemmer, Eds.), Amsterdam, 1986.

[BG73] R. Bellman and M. Giertz, On the analytic formalism of the theory of fuzzy sets, *Inform. Sci.* **5** (1973), 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, *in* "RIDE-DOM '95 (5th Int. Workshop on Research Issues in Data Engineering: Distributed Object Management)," pp. 124–131, 1995.

[CG96] S. Chaudhuri and L. Gravano, Optimizing queries over multimedia repositories, *in* "Proc. ACM SIGMOD Conference," pp. 91–102, 1996.

[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. Tork Roth, J. H. Williams, and E. L. Wimmers, Querying multimedia data from multiple repositories by content: The Garlic project, *in* "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. Zimmerman, L. A. Zadeh, and B. Gaines, Eds.), TIMS Studies in Management Sciences, Vol. 20, pp. 209–240, 1984.

[DP85] D. Dubois and H. Prade, A review of fuzzy set aggregation connectives, *Inform. Sci.* **36** (1985), 85–121.

[Fa98] R. Fagin, Fuzzy queries in multimedia database systems, *in* "Proc. ACM Symposium on Principles of Database Systems," pp. 1–10, 1998.

[FW97] R. Fagin and E. L. Wimmers, Incorporating user preferences in multimedia queries, *in* "Proc. 6th International Conference on Database Theory" (F. Afrati and Ph. Kolaitis, Eds.), Lecture Notes in Computer Science, Vol. 1186, pp. 247–261, Springer-Verlag, Berlin/New York, 1997. Full version under the title "A Formula for Incorporating Weights into Scoring Rules," to appear in *Theoret. Comput. Sci.*

[HR90] T. Hagerup and C. Rüb, A guided tour of Chernoff bounds, *Inform. Process. Lett.* **33** (1990), 305–308.

[Io89] M. Ioka, "A Method for Defining the Similarity of Images on the Basis of Color Information," Technical Report RT-0030, IBM Tokyo Research Lab, 1989.

[Mi89] M. Mizumoto, Pictorial representation of fuzzy connectives. I. Cases of *T*-norms, *T*-conorms and averaging operators, *Fuzzy Sets Syst.* **31** (1989), 217–242.

[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, *in* "SPIE Conference on Storage and Retrieval for Image and Video Databases," Vol. 1908, pp. 173–187, 1993; QBIC Web server is http://wwwqbic.almaden.ibm.com/.

[SS63] B. Schweizer and A. Sklar, Associative functions and abstract semi-groups, *Publ. Math. Debrecen* **10** (1963), 69–81.

[SC96] J. R. Smith and S.-F. Chang "Searching for Images and Videos on the World-Wide Web," Technical Report #459-96-25, Center for Telecommunications Research, Columbia University, 1996.

[SO95] M. Stricker and M. Orengo, Similarity of color images, *in* "SPIE Conference on Storage and Retrieval for Image and Video Databases III," Vol. 2420, pp. 381–392, 1995.

[TZZ79] U. Thole, H.-J. Zimmermann, and P. Zysno, On the suitability of minimum and product operators for the intersection of fuzzy sets, *Fuzzy Sets Syst.* **2** (1979), 167–180.

[VG83] W. Voxman and R. Goetschel, A note on the characterization of the max and min operators, *Inform. Sci.* **30** (1983), 5–10.

[Wi98a] E. L. Wimmers, Minimal Bellman–Giertz theorems, in preparation.

[Wi98a] E. L. Wimmers, A formula for getting the top elements, in preparation.

[WHTB98] E. L. Wimmers, L. M. Haas, M. Tork Roth, and C. Braendli, Using Fagin's algorithm for merging ranked results in multimedia middleware, in preparation.

[Ya82] R. R. Yager, Some procedures for selecting fuzzy set-theoretic operatons, *Internat. J. Gen. Syst.* **8** (1982), 115–124.

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

[Zi96] H.-J. Zimmermann, "Fuzzy Set Theory," 3rd ed., Kluwer Academic, Boston, 1996.