WHY IS THIS USER ASKING SO MANY QUESTIONS? EXPLAINING SEQUENCES OF QUERIES

Aybar C. Acar Amihai Motro
aacar@gmu.edu ami@gmu.edu
Department of Information and Software Engineering
George Mason University

Abstract A sequence of queries submitted by a database user within a short period of time may have a single, illuminating explanation. In this paper we consider sequences of single-record queries, and attempt to guess what information their authors may be trying to accumulate. Query sequences may reflect clandestine intentions, where users attempt to *avoid direct queries* which may disclose their true interests, preferring instead to obtain the same information by means of sequences of smaller, less conspicuous, queries. Sequences of queries may also reflect attempts to *circumvent retrieval restrictions*, where users attempt to approximate information which is inaccessible, with sequences of legitimate requests (in the latter case, our explanations may lead database owners to either *tighten* access, or, conversely, to reorganize their interfaces to *facilitate* access). Because the true objective of a sequence may be clouded by the retrieval of spurious records, our approach considers all the possible aggregates that a user may accumulate with a sequence, and to *rank* them, search-engine style, according to their plausibility as retrieval objectives. Our method is probabilistic in nature and postulates that the likelihood that a set of records is the true objective of the user is inverse proportional to the likelihood that this set results from random selection. Our method is shown to have good performance even in the presence of noise (spurious records) as high as 40–50%.

Keywords: Query sequences, inference attack detection, data mining, statistical databases

1. Introduction

Often, the owner of a database may ask "what are users retrieving from this database?" The answer to this question appears to be straightforward: It is the collection of queries submitted to the database, readily available in the system logs. However, such an answer, while correct, might not be very informative, as it could be long and complicated (e.g., hundreds of SQL expressions). In many cases, a more abstract answer would be preferred.

This problem is best described by analogy to data mining. Data mining is the essential activity of discovering trends, rules and other abstractions in large repositories of data. By analogy, the issue here is mining trends, intentions, patterns of use and other abstractions in collections of queries. The accumulated queries may be available either intensionally, as collections of user requests (e.g., SQL statements), or extensionally, as collections of database answers (i.e., tables), or possibly both.

The problem is different from conventional data mining. When considering intensions, we would be mining for abstractions in "data" that are collections of short programs; i.e., query statements. When considering extensions, we would be mining in data that are collections of sets of elements; i.e., a sought-after discovery would be a characterization of a collection of *sets of elements*, rather than of a collection of *elements*. In a way, this task may be considered *second-order mining*.

In this paper we focus on a particular form of query mining that involves sequences of *single-record queries* (queries answered by a single database record) that are submitted by the same user (or by a small group of users) within a short period of time.

This problem is applicable to *information assurance*, as a sequence of queries may have a clandestine explanation:

1 It may be an attempt to *hide* the particular objective of the user. Even though a particular query may be permitted, a user may want to conceal his interest in the subject, preferring to obtain the same information by means of a sequence of smaller, less conspicuous, queries.

2 It may be an attempt to *circumvent* a retrieval restriction. A particular type of request might not be feasible; yet a combination of other requests may provide a means to approximate the same information.

The latter circumvention may also reflect a more benign situation, in which a user is forced through a sequence of small queries, because the database interface is inadequate for the purpose. Given explanations of their users' true objectives, information providers may then choose to reorganize their databases and user interfaces to facilitate such quests.

In either of the cases described, the user submits a sequence of queries in order to construct off-line an aggregate of records that constitutes an answer to a database query, a query which the user either is unable to submit or prefers not to submit. We shall refer to this aggregate as the *goal* of the user. After an aggregate is identified as a likely goal, its semantics still need to be captured in a description that can be communicated easily to the database owner. We shall refer to this description as an *explanation*. Essentially, this explanation should correspond to the true query that the user has in mind. Hence, an explanation is an intensional expression whose extension corresponds to the goal with high

accuracy [15]. Because this aspect of our problem has been investigated and standard solutions have been developed, in this paper we do not address this final annotation of the most plausible goals.

Users who attempt to conceal their true objective are likely to include spurious requests in their sequence. Users who try to circumvent a retrieval restriction are likely to submit some erroneous requests. Consequently, our approach to the problem is that the goal of the user could be any subset of the set of records that were obtained by means of the query sequence. Our method assigns each subset a likelihood of being the goal of the user, and then uses this likelihood to rank the different subsets, search-engine style, from the most likely to be the goal of the user to the least likely.

Our approach is based on probability and it postulates that the likelihood that a set of records is the true goal of the user is inverse proportional to the likelihood that this set results from random selection. Clearly, *general* database queries (those that retrieve sets of records that satisfy a condition) tend to be deliberate (non-random) selections. Hence, our method is likely to discover attempts to approximate general queries with sequences of single-record retrievals.

For each candidate goal, the proportion of spurious records that are retrieved by the query sequence is referred to as *noise*. Our method obtains good results with noise as high as 40–50%. The amount of statistical analysis required limits the length of query sequences that can be handled effectively. Our experiments show that a typical server can handle effectively sequences of up to 10–20 queries, depending on the demands of the application (a 10-query sequence may take about 1 second, whereas a 20-query sequence may take about 16 minutes).

The overall approach is formalized in Section 2. Section 3 outlines the validation methodology and analyses the experiments that have been performed. Section 4 provides the appropriate context for this research work by surveying related works. Section 5 summarizes the results and describes additional work that is being pursued.

2. Overall Approach

Assume a single database file D with fields $A_1, \ldots, A_l$ and a total of n records. Some of the fields of D are *accessible* (fields that users can query and retrieve), other fields are *hidden* (fields that are not available for querying and are not retrieved; indeed, their presence in the database may not be known to users).[1]

Let $Q_1, \ldots Q_k$ be a sequence of single-record queries submitted by a user against the file D; i.e., each query in the sequence retrieves a single record of

the file. Single-record queries are fairly common; for example, any database interface based on database keys generates single-record queries.

Let Q denote the set of records targeted by this query sequence. That is, Q is the set of records that satisfy some search goal known only to the author of the query sequence. Thus, the query sequence $Q_1, \ldots, Q_k$ is an attempt by its author to materialize Q. Our purpose here is to develop methods with which the database system can approximate Q with high accuracy.

Let G be the aggregate of records actually retrieved by this sequence, and let m ($m \leq k$) denote the cardinality of G. These m records can be assembled into $p = 2^m - 1$ aggregates. We refer to these aggregates as *candidate goals* and denote them $G_1, \ldots, G_p$. The candidate goals include every possible subset of the retrieved records. In trying to rank these candidate goals according to their likelihood of being the true objective of the user, we compare the distribution of the values in each field in the candidate goal with its distribution in the database. We postulate that *the likelihood that a set of records is the objective of the user is inverse proportional to the likelihood that it results from random selection.*

In defense of this postulate, we note that it is likely to endorse candidate goals that correspond to general selection queries (queries that retrieve sets of records that satisfy a condition), because answers to such queries tend to be non-random. For example, a query to a student database on $Major =$"English" and $Residence =$"Virginia" is likely to produce non-random sampling in these two fields. Consequently, our method is likely to discover attempts to approximate general selection queries with sequences of single-record retrievals. Conversely, candidate goals that resemble random samples of the database will be ranked low. Of course, goals whose characterizing attribute is not included in the database will be ranked low as well. In the above example, a query sequence that accumulates records of students who are older than 40 years will appear to be a random sample if the database does not include a field Age.

Measuring Randomness

We test each of the p candidate goals for the randomness of its fields. That is, in each subset of records we compare the distribution of values in each of the fields to the distribution of the corresponding field in the original file. The basic statistical issue here is to assess whether a given set of elements is a random sample from a larger population. The more a set of elements appears to be random, the lower its likelihood of being an objective of retrieval. We have experimented with several statistical tests and the test that we describe in the following has given us the best results.[2]

Our measure is based on the concept of joint probability distributions of random variables. In essence, it calculates how improbable is a given subset of records, assuming all selections are random.

Let A_j be an arbitrary field of D, and let $v_1, \ldots, v_d$ denote the different values that occur in this field. Let n_i denote the number of occurrences of v_i in the field A_j. Hence, $n = \Sigma_{i=1}^{d} n_i$.

Consider an arbitrary candidate goal G_i. Let m denote its number of records, and let m_i denote the number of occurrences of v_i in the same field A_j of G_i. Similarly, $m = \Sigma_{i=1}^{d} m_i$.

Consider the values in the field A_j of G_i as a *random sample* of size m from the field A_j of D. If we assume that each query in the sequence returns a new record, then the sampling is *without replacement*, and the probability of this sample is

$$p(G_i^{A_j}) = \frac{\binom{n_1}{m_1}\binom{n_2}{m_2}\cdots\binom{n_d}{m_d}}{\binom{n}{m}} \tag{1}$$

If we assume that each query is independent of previous queries (and can thus retrieve records that have already been retrieved), then each query has *multinomial* distribution,[3] and the probability of the sample is

$$p(G_i^{A_j}) = \frac{m!}{m_1!m_2!\cdots m_d!}\left(\frac{n_1}{n}\right)^{m_1}\left(\frac{n_2}{n}\right)^{m_2}\cdots\left(\frac{n_d}{n}\right)^{m_d} \tag{2}$$

The probability of a sample is known as the *Fisher's likelihood* and is commonly used to estimate the parameters of the population. Here, we use the same probability as an indicator of randomness. Our argument is that random sampling is likely to produce sets of records that are representative of the file, and would result in high $p(G_i^{A_j})$ values. Hence, low $p(G_i^{A_j})$ values are likely the result of non-random sampling.

Since $p(G_i^{A_j})$ is taken to indicate the level of randomness, we rank the different candidate goals based on their $1 - p(G_i^{A_j})$ scores: high values indicate high likelihood that the values in this field are the result of deliberate (non-random) selection.

Fusing Multiple Rankings

The measure described above can be used to rank the candidate goals with respect to each individual field. Consequently, a candidate goal G_1 may rank high with respect to field A_1 (its A_1 values suggest a deliberate selection of records), but low with respect to field A_2 (its A_2 values appear to be representative of the file values). How should these ranks be combined? More specifically, assume a candidate goal G_1 with scores of 0.9 and 0.2 and 0.1 for fields A_1, A_2 and A_3, respectively, and a candidate goal G_2 with scores of

0.4. 0.4 and 0.4 for the same fields. Which of these is more likely to be the objective of the user?

Similar ranking fusion problems occur in Internet meta-search engines, which forward the same query to different search engines (each with different ranking algorithms) and combine the resultant rankings. Perhaps the most straightforward way of achieving this is to rank each document according to the sum of its ranks in the individual rankings. In analogy, each candidate goal is ranked with respect to the *sum* of the ranks it achieves for each individual field. Let $R_{i,j}$ denote the rank of candidate goal i ($1 \leq i \leq p$) with respect to field j ($1 \leq j \leq l$). Then the overall rank of this candidate goal is $\sum_{j=1}^{l} R_{i,j}$. Another ranking fusion possibility is to assign each candidate goal the *maximal* rank it achieved in its individual fields: $\max_{j=1}^{l} R_{i,j}$.

3. Validation

Methodology

Our purpose is to determine the set of records that the author of a sequence of queries is attempting to accumulate,[4] and our method is to *rank* the possible sets according to their perceived plausibility. Testing any method that claims to achieve this purpose requires inviting sequences of queries against a test database, and then comparing the professed targets of these sequences with the results generated by our method. The results should take into account the level of noise present in the sequence (the discrepancy between the professed target and the complete set of records retrieved by the sequence). Our validation methodology corresponds largely to such a test, except that much of it is simulated.

The first challenge is to simulate a query sequence that "attacks" a specific retrieval goal. We define a user's retrieval goal by means of a selection condition that involves several of the fields of the file, and we retrieve the database records that satisfy the goal. Let Q denote this set of records, and let m denote its cardinality. To generate a query sequence with noise level q ($0 \leq q \leq 1$), we randomly sample $q \cdot m$ records from the set Q and $(1-q) \cdot m$ records from the rest of the file. Denote the set of sampled records G. We then form a sequence of m single-record queries, each targeting a different record of G (these queries simply specify key values). This sequence is taken as an "attack" on the retrieval goal Q. We now perform the statistical analysis and ranking, as described in Section 2. The result is a ranking of the $2^m - 1$ candidate goals.

We now describe how we measure the success of this method. It must be noted that when comparing the set of records accumulated by a sequence, G, with the user's retrieval goal, Q, one observes two discrepancies: records in $G - Q$ and records in $Q - G$. Records in $G - Q$ are the spurious requests (noise), which have already been discussed. Records in $Q - G$ are records in the

user's retrieval goal that have not been retrieved. Since we limited the search for explanations to subsets of G, these records are not being considered in measuring the success of our method. Hence, our method should be considered fully successful if it identifies the *practicable goal* $Q \cap G$. It is the purpose of the eventual phase of annotating goals with explanations to "compensate" for this omission. That is, the optimal explanation for a candidate goal G_i should be a concise expression whose extension E_i "fits" G_i optimally, minimizing both discrepancies $E_i - G_i$ and $G_i - E_i$. This expression could encompass records that have not been retrieved by the query sequence.

In summary, our method is considered *fully* successful if its top-ranked goal is *identical* to the practicable goal. Otherwise, we judge its *level* of success with the goal's *similarity* to the practicable goal. Our definition of similarity is the *overlap measure* which quantifies the similarity of two sets with the proportion of the cardinalities of their intersection and their union:

$$\Omega(G_i) = \frac{|(G_i \cap (Q \cap G))|}{|(G_i \cup (Q \cap G))|} = \frac{|(G_i \cap Q)|}{|(G_i \cup (Q \cap G))|} \tag{3}$$

The value of the overlap measure is between 0 and 1; it is 0 when the sets are disjoint and 1 when they are identical. The measure may be considered a combination of the dual *recall* and *precision* measures known from classical information retrieval.

It is possible that our method will fail to place the practicable goal at the very top of its ranking, yet nonetheless this goal will be ranked high. As we assume that the database owner is to be presented with a set of the most plausible explanations, we shall consider such situations as partially successful. Therefore, *complete* success is when the candidate goals are ordered in descending order with respect to their similarity to the practicable goal (their Ω scores). Otherwise, the *rate* of success is calculated as the deviation of our method's ranking from this ideal ranking.

Table 1 shows a small example in which the total number of retrieved records is $m = 3$ and the number of candidate goals is 7. The true goal is $Q = \{a, b, c\}$, the records retrieved are $G = \{a, b, d\}$, and the practicable goal is $Q \cap G = \{a, b\}$. The level of noise is therefore 33%. The table lists the candidate goals, their similarity scores and the different rankings: The column *Ideal* is the ranking by similarity and the column *Method* is the ranking by our method.[5]

There are various alternatives for measuring the difference between the two rankings. As each ranking is a permutation of the integers $1, \ldots, p$, one possibility is to measure the *distance* between the two permutations. Another possibility is to compare the similarity scores of candidate goals that occupy the same position in the two rankings. The error in each position is measured by the square of the difference between the similarity scores, and then the er-

Goal	Ω *Score*	*Ranking (Ideal)*	*Ranking (Method)*
$G_1 = \{a, b, d\}$	0.67	2	1
$G_2 = \{a, b\}$	1.0	1	3
$G_3 = \{a, d\}$	0.33	5	2
$G_4 = \{b, d\}$	0.33	6	4
$G_5 = \{a\}$	0.5	3	7
$G_6 = \{b\}$	0.5	4	6
$G_7 = \{d\}$	0	7	5

Table 1. Rankings for a 3 query sequence.

ror is totaled for the entire set of candidate goals. When the two rankings are identical, the total error is 0.

Since our interest is primarily in the performance of our method with respect to the top part of its ranking (the *head* of the ranking), we choose to compare the *mean* similarity scores of the heads of the rankings (of course, the mean similarity scores of the entire rankings are identical). The *method's mean* is the average similarity to the practicable goal exhibited by the top ranked candidate goals, when they are ordered according to our method. The *ideal mean* is the average similarity exhibited by the top ranked candidate goals, when they are ordered according to their similarity to the practicable goal. Clearly, the ideal mean is the highest mean that a ranking could achieve. The *ratio* of our method's mean to the ideal mean is adopted as an indication of the success of this method.

Table 2 shows these mean similarity scores for the previous example, assuming that the head of the ranking is defined as the top 3 positions. The average similarity of the top ranked candidate goals to the practicable goal is 0.67. The highest possible average similarity of the top ranked candidate goals (in any ranking) to the practicable goal is 0.71. The success ratio is therefore is 0.92. In other words, at 33% noise level, the loss of average similarity at the top 3 positions is less than 8%.

Rank Position	Ω *Score (Ideal)*	Ω *Score (Method)*
1	1.0	0.67
2	0.67	0.33
3	0.5	1.0
Mean	0.72	0.67

Table 2. Mean similarity scores for the heads of the rankings.

Finally, one may also consider the method successful, if it places the practicable goal the among the top ranked candidate goals (i.e., at the head of the ranking). In the example, the method succeeds, because G_2, the practicable goal, is ranked third.

Experimentation and Analysis

The database of the experiment was a 13-field file with a total of 4,000 records, created using real world data [19]. Two different retrieval goals were tried: (1) a *complex* retrieval goal defined by a query with a conjunctive selection condition spanning 4 fields (3 equality comparisons and one range comparison), and (2) a *simple* retrieval goal defined by a query with a selection condition of a single equality comparison. We analyzed query sequences of length 10, and we experimented with 10 noise levels: from 0% to 90% with 10% increments. At each noise level, 10 different query sequences were attempted according to the methodology described earlier (with the exception of the 0% noise level, for which only one query sequence is possible). Altogether, for each retrieval goal, 91 query sequences were attempted. The number of candidate goals for sequences of length 10 is 1,023, and the head of a ranking was defined to consist of the top 10 candidate goals (less than 1% of the entire set). To fuse the rankings of independent fields (Section 2.3), both the *sum* and *max* methods were attempted. The results obtained with the *sum* method were significantly and uniformly superior to those obtained with the *max* method, leading us to adopt the *sum* fusion method. The results presented here are for this fusion method only.

Tables 3 and 4 summarize the results of these two experiments. Each row shows the average performance of 10 query sequences at the specified noise level (except for the first row which shows the performance of only one query sequence). *Mean Position* is the average position of the practicable goal in the ranking given by our method. For example, if the practicable goal was listed twice in position 1, 3 times in position 2, 3 times in position 3 and twice in position 4, the mean position would be 2.5. The final three columns measure the success of our method with the ratio of our method's head-of-ranking mean similarity to the ideal head-of-ranking mean similarity. The first of these columns is the mean of the mean similarity scores in our method's ranking; that is, the head-of-ranking mean similarity scores are averaged for all the tests at the same noise level. The next column is the head-of-ranking mean similarity in the ideal ranking (this score is identical for all tests at the same noise level). The final column is the mean rate of success.

The results in the complex query experiment are quite strong. For example, at noise levels up to 20%, the practicable goal was, on the average, in the top 3 positions, and at noise levels up to 40%, it was, on the average, in the top

8 positions. As expected, as noise increases, the values in the third column declined more rapidly than the values in the fourth column, resulting in ever-decreasing success ratio. Still, at 60% noise level, the ratio was over 0.8; that is, at this noise level, the loss of average similarity among the top ranked candidate goals was, on the average, less than 20%.

Noise Level	*Mean Position*	*Mean Ω (Method)*	*Mean Ω (Ideal)*	*Mean Success*
0%	1.00	0.8678	0.9100	0.9536
10%	2.50	0.8348	0.9011	0.9264
20%	3.00	0.8127	0.8903	0.9129
30%	5.50	0.7911	0.8768	0.9023
40%	7.60	0.7707	0.8595	0.8966
50%	12.90	0.7098	0.8367	0.8484
60%	26.30	0.6596	0.8050	0.8193
70%	39.50	0.5171	0.7583	0.6818
80%	66.70	0.4203	0.6833	0.6151
90%	92.60	0.1847	0.5500	0.3358

Table 3. Experiment with complex retrieval goal.

The simple query experiment was a bit less successful. At 20% and 40% noise levels, the mean position was within the top 6 and 9 positions, respectively (compared with 3 and 8, respectively, for the complex query). The success ratio dropped below 0.8 at a noise level of 50% (compared with 70% for the complex query). Figure 1 plots the success ratios of the complex and simple queries. This difference in performance is fairly simple to explain. As discussed earlier (Section 2.2), each comparison in the selection condition of the query is likely to affect the distribution of the values in a particular field. Thus, a query with a single comparison is likely to result in at least one field in which deliberate sampling is apparent, whereas a query with four comparisons is likely to result in at least four fields in which deliberate sampling is apparent. Therefore, complex queries provide our method with more "evidence" of deliberate sampling.

Overall, the results were strong enough to suggest that unless users spend the *majority* of their queries to retrieve information they do not want, the explanation for their true intentions can be found among a very small set of possible explanations.

Performance

So far, our main concern in validating our method has been its ability to detect successfully the true intentions of authors of query sequences. Another important concern is the *time* performance of the method. Our attempt to ex-

Noise Level	*Mean Position*	*Mean Ω (Method)*	*Mean Ω (Ideal)*	*Mean Success*
0%	1.00	0.8556	0.9100	0.9402
10%	3.70	0.7938	0.9011	0.8809
20%	5.60	0.7813	0.8903	0.8775
30%	8.10	0.7535	0.8768	0.8594
40%	8.70	0.7047	0.8595	0.8199
50%	12.20	0.6666	0.8367	0.7967
60%	33.30	0.5154	0.8050	0.6402
70%	45.40	0.4686	0.7583	0.6179
80%	76.30	0.3543	0.6333	0.5184
90%	276.80	0.1553	0.5500	0.2823

Table 4. Experiment with simple retrieval goal.

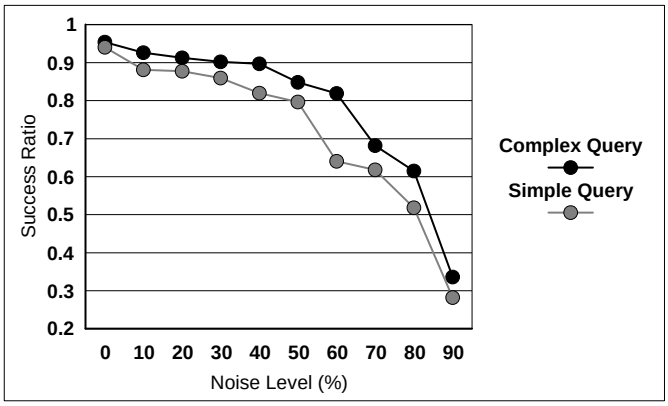

Figure 1. Comparison of the success rates of the complex and simple queries.

amine *all* the subsets of the records aggregated by a user (therefore allowing for an *arbitrarily high* level of noise), results in a process whose complexity is exponential in the length of the query sequence. On a typical computer server, the 10-query sequences of our experiments can be analyzed in about 1 second each. Sequences of 20 queries can take up to 16 minutes each. If we assume that input sequences have noise levels that do not exceed 50%, then we can limit our analysis to subsets that include at least 50% of the accumulated records. This reduces the number of candidate goals considerably, cutting the analysis times to about 0.6 second for a 10-query sequence, and 9 minutes for a 20-query sequence. A decision to avoid analyzing very small candidate goals may be justified by the fact that for very large levels of noise, the method is of limited benefit, anyway.

Another issue of performance may be suggested by the formulas used to calculate the probabilities $p(G_i^{A_j})$ (Section 2.2). When the file D is large (n is high), and the candidate goal G_i approaches in size to about half of this size (m is close to $n/2$), the numerators and denominators in Formula 1 can become excessively large. Nonetheless, if m is the range of 10–20, as suggested above, we get effective calculations for files with n as large as 1,000,000.

4. Background

The subject of this paper is the interpretation of sequences of single-record queries (i.e., sets of records accumulated by users). This subject has applications both in a *cooperative setting*, where the objective of the database system is to learn the query patterns of its users with the intention of facilitating the attainment of their eventual goals, and in a *controlled setting*, where the objective of the database system is to ascertain that users do not circumvent retrieval restrictions that protect specific portions of the database. We are unaware of previous work that is related to the former objective; the latter objective has been addressed in the areas of *data mining*, *statistical databases*, and *database security*. Below, we briefly position our work in these contexts.

The advent of data mining [14] and especially the development of increasingly effective and efficient methods of discovering associations and dependencies in vast amounts of data [2] have brought about considerations of security and privacy. Both these considerations arise from the fact that it is possible to ascertain confidential data by processing related but unrestricted information. Especially with large databases or data warehouses, the inference of confidential information, such as details regarding individuals, is a significant risk. As an abstract example, assume that B is a restricted field of the database but A is unrestricted. The discovery of an association rule of the kind $(A = a) \rightarrow (B = b)$ can be used to (1) determine the value of B whenever the value of A is known to be a, or (2) circumvent a restriction on retrieval of records by $(B = b)$, substituting it with retrieval on $(A = a)$. There have been several studies that have tried to strike a compromise between the legitimate need to mine data for general trends on one hand, and the protection of sensitive details on the other. One method is to introduce perturbations in individual data items without disturbing the general properties of the data as a whole [3]. Another method is to restrict disclosure of the results of data mining experiments according to thresholds on support and confidence [9].

For similar reasons, such controls have also been important in statistical databases. The main purpose again is to prevent disclosure of specific information, pertaining to individuals. An extensive survey of the methods used is given in [1]. The basic methods involve not returning results smaller than a given threshold or returning only aggregate results. One emphasis is the defeat

of *tracker* methods [11, 12], methods used by attackers to infer specific data (data about individuals) by manipulating carefully selected aggregates, while conforming to the restriction on the minimal size of queries.

In the related field of database security, two forms of attack on secure databases have gained wide attention: inference and aggregate attacks. An *inference attack* is an attempt to uncover classified information by combining knowledge of unclassified information with "outside knowledge" (e.g., association rules of the kind discussed earlier). An *aggregate attack* is an attempt to gain access to a classified aggregate of records by accumulating a sizeable number individual, unprotected records from this aggregate [4, 8, 16]. Practical applications center on the *prevention* of such attacks, whereas our work here may be regarded as *detection* of possible attacks. Although detection can lead to prevention, it is not the central aim of this study, which is to uncover possible leakage of information.

Methods that have been developed to detect inference and aggregate attacks can be classified as *schema level* and *data level* detections. Our approach here is of the latter type.

In its simplest form, schema level detection attempts to recognize the exploitation of functional dependencies in the schema of a database [13]. An exploitable "opening" that allows users to conclude classified information from unclassified information is referred to as an *inference channel*. One major approach links different elements in the schema to each other using semantic metadata specific to particular domains. This is then used by logical inference engines to decide whether involuntary disclosures are being made [10]. Other methods require less markup at the schema level but depend on expert information about the domain in the form of inference rules. For example, [5] illustrates the use of a monitor that keeps track of all a user's queries, constantly compares them to a set of predetermined inference channels, and denies any requests for objects that may satisfy the premise (antecedent) of an inference channel when combined with previously retrieved objects. A more recent method avoids the need to maintain complete query histories by keeping track of the number of building blocks already disclosed from the premise of an inference channel [18].

In general, schema level methods are efficient in operation and thus are suitable for real-time detection and prevention techniques. However, their proper operation mandates that *all* the possible inference channels be identified beforehand, and therefore requires exhaustive knowledge of the domain.

Data level inference detection, on the other hand, analyzes the values of fields in a file and the relationship of these values to values in other fields. This may lead to the discovery of previously unknown inference channels, channels that may be intrinsic to the application domain or the specific database at hand. It has been shown experimentally [20] that data level analysis is much more

comprehensive, in that inference channels that are not apparent in schema level analysis are actually found. The disadvantage of data level inference detection, again as stated in [20], is that it is computationally expensive and is therefore best done off-line.

In terms of effective implementation, to our knowledge, no complete data level inference detection system exists. There have been studies detailing the use of data mining techniques such as decision trees [6] and Bayesian networks [7] to discover inference channels in an automated way. These identified channels create a rule base with which the database owner may then restrict access to otherwise unclassified items lest they form precursors for an inference attack. This restriction can be absolute, or can rely on a process, such as those proposed by [5] and [18] above, that monitors the disclosure and restricts access only when a certain situation is reached. Using data mining methods requires extensive data preprocessing, however, and this may not be possible on an active database.

One method, described in [21], is similar to ours in that it applies probabilistic assessments to detect inference. The method proposes using rough-set theory [17] to label each object in the database with probabilities of causing an inference risk if disclosed. While this approach does not depend on predetermined rules, the inference rules identified are limited to binary relationships between field names and values.

We conclude by reiterating that our method does not assume any domain knowledge or prior identification of inference rules or channels. Additionally, it is not restricted by the relatively simple inference rules or channels assumed elsewhere. Since our method altogether disregards the conditions specified in the retrieval requests, it is not subject to the limitations of an "antecedent" (the premise of the inference channel), and its "consequent" (the target of the inference channel) can be a rather complex expression incorporating multiple basic comparisons.

5. Conclusion

We considered the issue of single-record query sequences that are submitted by a single user within a short period of time, and we attempted to discover the true intention of that user; that is, the subset of the record set accumulated by that user that is most likely the actual retrieval goal. This research has obvious applications for information providers, who may use the discoveries to *facilitate* access to their information, as well as in information assurance, where the discoveries may indicate more clandestine intentions, and may lead to better *protection* of the information.

Were it not for spurious requests in the sequence, the complete set of records accumulated by the sequence would correspond to the true retrieval goal. How-

ever, the possibility of spurious requests (whether the result of errors, unsuccessful guessing, or deliberate attempts at concealing intentions) dictates that any subset of the accumulated records must be considered a candidate for being the true retrieval goal.

Our method is purely statistical and does not assume any prior knowledge about specific retrieval targets embedded in the database; indeed, any collection of records that is the result of a selection query is a conceivable target. The output of our analysis is a ranking of the candidate goals, according to their likelihood of being the true retrieval target.

Our experiments show very good performance. At noise levels (percentage of spurious requests) of up to 50%, the true retrieval target may be expected to be found among the top 1% of the rankings. In other words, unless users spend the *majority* of their queries to retrieve information they do not want, the explanation for their true intentions can be found among a very small set of possible explanations (on the "first page"). Indeed, it is unlikely that any method could identify a coherent retrieval goal when the level of noise is extremely high.

Considering all the possible subsets of the accumulated record set implies that processing cost is exponential in the number of accumulated records (roughly, the length of the sequence). In practice, query sequences of length 10 require less than 1 second, performance which is acceptable for real-time analysis. The performance for longer sequences render the analysis more suitable for off-line application; for example, a sequence of length 20 requires about 9 minutes.

The work described here continues in several directions and we describe here three such directions.

1. *Performance*. One obvious objective is the improvement of performance. We are investigating methods that will avoid the exponential cost of exhaustive analysis of all possible subsets of records. Recall that the true retrieval goal is generally found in the top 1% of the rankings. Our interest is in heuristics that will get us faster to these record subsets.

2. *Real-time mode*. Conceivably, there are two different modes in which our method can be applied. For the most part, the discussion in this paper corresponds to the method's *off-line* mode, in which the query sequence is obtained from a log and is analyzed after the queries have been executed. The benefits of this mode are mostly informational. In the *real-time* mode, a small "sliding window" on the query sequence is to be observed and analyzed. The purpose would be to detect when a query sequence is "converging" into a plausible goal, and promptly alert the database system. In this mode, the limitation on the length of the query sequence would have much less significance.

3. *Robustness.* For successful application to information assurance, one must worry whether the method can be deceived. We assumed that deception would be in the form of spurious records, but our tacit assumption has been that these would be chosen randomly. Yet, with sufficient knowledge of

the records in the database, one could compose a sequence in which the spurious records would constitute a "decoy" target; for example, a sequence of 10 queries would retrieve two "clusters" of records: the larger cluster (say, 6 records) would consist a decoy target, whereas the true retrieval goal would be the smaller cluster. In such cases, our method is likely to rank the true target well below the decoy target.

Finally, the research reported here is part of a larger investigation of what we call *second-order data mining*: Finding trends, intentions, patterns of use and other abstractions in *collections* of database queries. Of the many additional research issues in this general area, we mention three.

1. *More general queries*. Our focus was on single-record selection queries. In the general case, one must consider queries that retrieve arbitrary numbers of records, as well as queries that involve joins, projections or aggregate functions.

2. *Additional characterizations*. The problem we addressed was to find a single explanation for a sequence of query. In the general case, one should consider other characterizations of a set of queries, including statistical conclusions, clustering, association rules and other abstractions.

3. *Intensional form*. In this paper we analyzed the set of records retrieved by a sequence (the *extension* of the queries). In the general case, it may be advantageous to consider the intensional form of queries as well (e.g., the SQL statements).

Notes

1. Our database model may be viewed as relational, though we use the generic terms *file*, *record* and *field* rather than *relation*, *tuple* and *attribute*.

2. In particular, it deals well with samples that could be rather small — just a few elements.

3. Multinomial distributions can also be assumed in the case of sampling without replacement, when the population is large and the sample is relatively small.

4. And subsequently annotate this set of records with a descriptive explanation.

5. The values in the *Method* column are just illustrative, as the entire file is not known in this example.

References

[1] N.R. Adam and J.C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of VLDB-94, the 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 439–450, 2000.

[4] V. Ashby, S. Jajodia, G. Smith, S. Wisseman, and D. Wichers. Inference and aggregation issues in secure database management systems. Technical Report 005 (Volume 1/5), National Computer Security Center, 1996.

[5] A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):900–919, 2000.

[6] L.W. Chang and I.S. Moskowitz. Parsimonious downgrading and decision trees applied to the inference problem. In *Proceedings of NSPW-98, Workshop on New Security Paradigms*, pages 82–89, 1998.

[7] L.W. Chang and I.S. Moskowitz. A Bayesian network schema for lessening database inference. In *Proceedings of CIMCA-01, International Conference on Computational Intelligence for Modelling, Control and Automation*, 2001.

[8] L.W. Chang and I.S. Moskowitz. A study of inference problems in distributed databases. In *DBSEC-2002, Research Directions in Data and Applications Security, IFIP WG 11.3 Sixteenth International Conference on Data and Applications Security*, IFIP Conference Proceedings, Volume 256, pages 191–204. Kluwer, 2003.

[9] E. Dasseni, V.S. Verykios, A.K. Elmagarmid, and E. Bertino. Hiding association rules by using confidence and support. Lecture Notes in Computer Science, Volume 2137, pages 369–383. Springer-Verlag, 2001.

[10] H.S. Delugach, T.H. Hinke., and A. Chandrasekhar. Applying conceptual graphs for inference detection using second path analysis. In *Proceedings of ICCS-93, International Conference on Conceptual Structures*, pages 188–197, 1993.

[11] D.E. Denning, P.J. Denning, and M.D. Schwartz. The tracker: a threat to statistical database security. *ACM Transactions on Database Systems*, 4(1):76–96, 1979.

[12] D.E. Denning and J. Schlrer. A fast procedure for finding a tracker in a statistical database. *ACM Transactions on Database Systems*, 5(1):88–102, 1980.

[13] J. Hale and S. Shenoi. Analyzing FD inference in relational databases. *Data and Knowledge Engineering*, 18:167–183, 1996.

[14] B. Kero, L. Russell, S. Tsur, and W.-M. Shen. An overview of database mining techniques. In *Proceedings of KDOOD/TDOOD-95, the DOOD'95 Post-Conference Workshops*, pages 1–8, 1995.

[15] A. Motro. Intensional answers to database queries. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):444–454, 1994.

[16] A. Motro, D.G. Marks, and S. Jajodia. Aggregation in relational databases: Controlled disclosure of sensitive information. In *Proceedings of ESORICS-94, Third European Symposium on Research in Computer Security*, Lecture Notes in Computer Science, Volume 875, pages 431–445. Springer-Verlag, 1994.

[17] L. Polkowski and A. Skowron. Rough sets: A perspective. In *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, pages 31–56. Physica-Verlag, 1998.

[18] J. Staddon. Dynamic inference control. In *Proceedings of the 8th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 94–100, 2003.

[19] P. van der Putten and M. van Someren (eds.). CoIL challenge 2000: The insurance company case. Technical Report 2000-09, Sentient Machine Research, Amsterdam and Leiden Institute of Advanced Computer Science, Leiden, 2000.

[20] R.W. Yip and K.N. Levitt. Data level inference detection in database systems. In *Proceedings of CSFW-98, The 11th Computer Security Foundations Workshop*, pages 179–189, 1998.

[21] K. Zhang. On rough sets and inference analysis. In *Proceedings of the First International Information Security Workshop*, pages 256–265, 1997.