Microprocessing and Microprogramming 32 (1991) 755–764
North-Holland

ALGORITHMS FOR INFERENCE CONTROL

Achilles Kameas, Panagiotis Fitsilis, George Pavlides

Computer Technology Institute
and
Dept. of Computer Engineering and Informatics
University of Patras
P. O. Box 1122, 26110 Patras, GREECE
e-mail: fitsilis@grpatvx1.bitnet

ABSTRACT

In this paper a method to deal with the inference problem is presented. The database is represented with a directed bipartite graph. At each clearance level, different portions of the graph are accessible. Then, inference is defined as the existence of a 'virtual' cycle of information flow, at any classification level. The method we present consists of algorithms that detect and eliminate these cycles. Finally an algorithm that deals with compound queries is presented.

1. INTRODUCTION

The term *inference*, in information theory, refers to the process of proving or deriving conclusions based on some given facts and rules of derivation. In the context of computer security, the *inference problem* usually refers to one obtaining information he is not authorized to access directly. This is accomplished with the combination of one's *external knowledge* with information legally revealed by the system. This problem frequently arises in database systems with multilevel classifications. In these systems, users gain access to information with higher classification by posing expertly designed queries. The inference problem has not yet been solved effectively. This is due to the inability of controlling the human factor that is involved when inference is caused. That is, one cannot be sure of the extend of the user's external knowledge, and therefore, inference controls tend to be overprotective. On the other hand, any measure against inference that does not operate on the worst case principle is certain to leak information to any expert user's queries.
In this paper we do not claim solution to the inference problem as a whole. We only present some algorithms that can be used *in practice* to deal with simple and not so simple cases of inference, such as compound queries. We adopted the representation of a directed bipartite graph mostly to show that the concept that underlies inference is the continuity of information flow between entities and relations. This concept is represented with the notion of 'virtual' cycles that show up when we move down one classification level. We claim that to stop inference one must break these cycles, thus breaking the continuous flow of information. Then, the database is partitioned into clusters that can be accessed safely. We present two algorithms that detect and eliminate these 'virtual' cycles. We then show how to construct the query answer tree in order to answer a query safely.
If the query cannot be answered as a whole, only a part of it (that does not cause inference) will be answered. To prevent the combined use of safe queries to extract classified information, we present an algorithm that uses the notion of 'history' (it examines all the previously posed queries before answering the last one).

2. RELATED WORK

Much work has already been done on the subject, beginning with Goguen and Meseguer [1], who define a *non interference* condition which states that at any classification level, all effects coming from users at higher classification levels must be invisible.Su and Ozsoyoglou [2] presented two algorithms for classifying a database with respect

to the FDs and MVDs that hold between attributes. Their work applies to the level of consistently classifying a database, while the work of Hinke [3] and Morgenstern [4] refers to the level of reclassifying a database. Morgenstern introduced the INFER function to quantify the inference caused by a set of attributes at a certain classification level. Hinke's work is more general and defines inference as the existence of more than one paths with different aggregate classifications between two attributes. These two papers will be discussed in more detail later.

2.1. Query answer trees

As mentioned in [5], an E-R diagram can be transformed into a graph. Let D be the set of attributes, let S be the set of entities and let R be the set of relations. The *database graph* of an E-R scheme is a graph <V, E> where

$$V=D \cup S \cup R,\ W=S \cup R \text{ and } E \subseteq W \times V$$

If w belongs to W and d is an attribute of w, then edge (w,d) belongs to E. If entity S participates in relationship v, then edge (s, v) belongs to E. No other edges belong to E. The database graph of an E-R scheme differs from an E-R diagram in that the attributes are explicitly represented by a node in the graph. In addition, in a database graph one may use a directed cost function to represent the cardinality of the relations.

Furthermore, a way to represent queries with query graphs, in the relational model, is presented. In this paper, a query graph is a connected subgraph of the database graph.

A *query graph* cannot represent relational operations other than the natural join. A query is a *simple query* if it can be expressed by a single query graph. It is compound if it can berepresented as a relational expression in terms of simple queries. Finally, a cycle-free query is called a *query tree*, since it must be connected.

2.2. Inference detection using query answer paths

Hinke [3] has extended the notion of database graph to what he calls the *semantic relationship graph*. This is a directed graph G=<V, E>, with arrows indicating the direction of information flow, as well as the cardinality of the relationships they represent. The vertices V of G represent a set of data elements. Any two vertices V_1 and V_2 are connected by two edges: E_{12} which points from V_1 to V_2 and E_{21} which points from V_2 to V_1.

The existence of an edge E_{ij} indicates that knowledge of an element from the set V_i, permits one to know one or more elements from the set V_j. A path from V_1 to V_m is defined as a set of contiguously connected edges $E_1, E_2, \ldots, E_m$; there can be any number of edges in the path.

An inference is said to exist if for some edge E_{ij} connecting V_i to V_j, and having a classification equal to security level A, there is a path from V_i to V_j, which does not include edge E_{ij} and which has aggregate classification level B<A. This path is called an *inference path.*

In this paper, we use both the notions of semantic relationship graphs and query answer graphs. We assume that a query corresponds to a subset of the database graph. To answer a query, one has to find a path that contains all the attributes referenced in the query. Probably, this path will contain relations and entities irrelevant to those in the query, that must be used as intermediate links between them. We will use the whole query subgraph, and not the minimal tree, because this is where inference comes up. Following Hinke's definition, we assume that inference arises whenever there are more than one query answer paths, with different aggregate classifications.

2.3. Quantifying inference

Denning [6] has shown that the amount of information about y that can be inferred from x is measured by the *relative equivocation* that represents the reduction in uncertainty about y when x is known, and which is given by the following rule:

$$H_y = \sum_{xy} p(x, y) \log_2 \frac{1}{p_y(x)}$$

It assumes values ranging from zero, if no information can be inferred about y once x is known, to one when x discloses full information about y. Morgenstern [4] defined the inference function INFER(x->y) to be this relative equivocation whenever its value exceeds some threshold e. This threshold is used to give a handle on quantizing a tolerance of minute information flows. It may be set to zero, if one cannot afford to have any inference at all. The mathematical definition of this function is given below:

$$\text{INFER}(x \to y) = \begin{cases} \dfrac{H(y)-H_x(y)}{H(y)}, & \text{iff } \dfrac{H(y)-H_x(y)}{H(y)} > e \\ 0, & \text{otherwise} \end{cases}$$

If x discloses no information about y, then $H_X(y)=H(y)$ and INFER(x->y) =0. If x discloses full information about y (its exact value), then $H_X(y)=0$ and INFER(x->y) =1.

3. CONTROLLING INFERENCE IN MULTI-LEVEL CLASSIFICATION SCHEMES

In a multilevel database we assume that there are data classified at various levels and users with different clearances. A user can access all data that have classification level lower than or equal to his clearance.
Unfortunately, this often is not the case. An expert user can gain access to data classified at higher levels than his clearance by posing well-designed queries that return data he can access, and then by relating these data with his knowledge of relations between them. This was defined above to be the inference problem.
In all the previous work done on the subject, a database is represented with a graph. We will adopt a similar representation in our work with the sole difference being the use of a bipartite graph. We assume that the nodes in the graph correspond to tables (entities and relations) and that the edges show the authorized flow of information between them. Furthermore we assume that classification levels have already been assigned to the database components (by some process which is not of our interest), though these levels may not be consistent. Su and Ozsoyoglou have described a method of assigning classification levels that are based on dependencies between attributes. They prove the problem to be NP-complete, but give efficient algorithms to solve it. For a query to be answered, all the entities and relations that participate in the construction of the table to be returned, must be connected by means of paths in the database graph. Thus the *query answer table* is a subgraph of the database graph that is visible at the user's clearance level.
Then, in general, we define inference to be the existence of *virtual cycles* in the query answer graph; these cycles can be formed from the query answer graph with the use of subgraphs that are visible at the next higher clearance level, as will be explained later. This definition is equivalent to the one given by Hinke, in which inference is caused if more than one paths, with different aggregate classification, exist between two entities in the graph. Then, one can obtain classified information by following the paths with lower than his clearance classification levels.
To deal with inference we propose the breakage of these virtual cycles. This can be done in many ways, depending on the level of classification we are using (i.e. when using attribute level classification, one can reclassify an edge in the path). This will result disjoint groups of nodes that are securely accessible. In addition, there are many possible criteria one can use to determine the edge that causes the greatest damage.

3.1. Representation of the problem

We shall start with some definitions, to make our work-space clear. The reader must keep in mind that we are working on a database with attribute-level classification.

Definition 1. Let G= (V, E) be the database graph. Since this is a directed bipartite graph, we have:

$$V = EV \cup RV$$

(the set of nodes is the union of the set of entities and the set of relations), and

$$E=\{(v_i, v_j) \mid (v_i \in EV \wedge v_j \in RV) \vee (v_i \in EV \wedge v_i \in RV)\}$$

(no edge connects two entities or two relations). This graph is extended in G'=(V',E) where

$$V'=\{(v_i, c_i) \mid c_i \in CL\}$$

and CL is the ordered set (in decreasing order) {Top Secret, Secret, Confidential, Unclassified}; to each node is assigned a classification level. We note here that edges are not classified, since they only denote the flow of information, but do not carry any information at all.
Definition 2. Each entity node consists of a collection of attribute nodes. This set of attributes is organized as a graph that shows the functional and multivalued dependencies among them. A classification level is assigned to each attribute which holds for all contents of this attribute (Figure 1a).
This means that an attribute classified at level A is completely invisible (and therefore not accessible) at lower than A clearance levels. The structure of the entity nodes generally is not of our concern here.
Definition 3. Each entity is represented by its key attributes, which are classified at the Upper Lower Bound of all the other attributes in it. The entity itself is classified at the key level (Figure 1a).
This means that information flows from the key towards every attribute of the entity but never backwards. To ensure this, we demand that the key attributes are classified lower than all the other attributes in the entity;therefore, every user with clearance level at least equal to the classification level of the entity is able to access the key of the entity and at least one other attribute. This is in accordance with the integrity rules Denning has proposed [7].

There is a special status governing the behavior of the entity once a query is posed. Suppose that we have entity A[C] (K[C],A_1[C],A_2[S],A_3[TS]), where K is the key, A_1, A_2, A_3 are the other attributes, and classification levels are shown in brackets (the entity classification level is C). A user with clearace level TS can access all the attributes of this entity. A user with clearance level S cannot access attribute A_3[TS]. Therefore, the contents of the entity (i.e. the attributes it is composed of) may differ at different security levels. If this user poses a query that contains this TS attribute, the whole entity is temporarily reclassified at level TS, to show that this one user cannot access the subset of the attributes contained in his query.
Corollary 1. A query that refers to attributes at higher classification levels than the user's clearance is not answered at all, rather than returning only the allowable subset of attributes.
Definition 4. Each relation node consists of

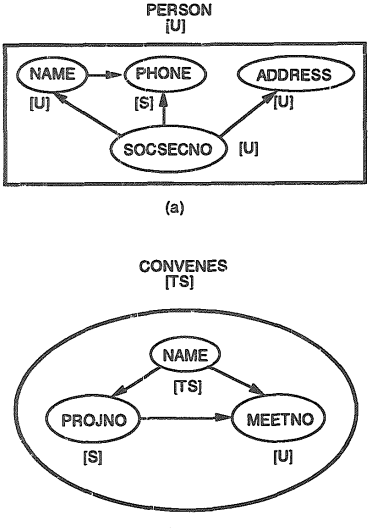


Figure 1.Representation of entities and relations

attribute nodes. These nodes are the key nodes of the entities [illegible] and any other possible relation attribute nodes form a complete group. A relation must at least be classified at the least Upper Bound of the entities it relates (Figure 1b). This guarantees that if a relation is accessible, then all entities it relates will also be accessible.Classification of a relation at a level that is higher than all the entities it relates is not prohibited. The graph consists of two kinds of nodes corresponding to entities and to relations between them (it is a bipartite graph), and of edges that show the flow of information between entities and relations. Note that neither two entities nor two relations can be directly connected via an edge. We represented relations with nodes to make clear that they hold information to be protected and do not only act as buffers of information flow. For example, when an entity is classified at Secret level, this means that every attribute is classified at least at Secret level, with the key attributes classified at Secret level.

Definition 5. A different portion of the graph is visible at each clearance level. The portion that is visible at some classification level is a superset of the portion visible at lower ones.

This is logical, since the higher the user's clearance, the larger the portion of the database he can access. Users with Top Secret clearance must virtually be able to access the whole database. Thus, Secret nodes are not visible in Unclassified level.

Corollary 2. When one moves from a higher level to a lower one, paths in the database break, as nodes become not visible and connections between entities disappear.

The problem of inference in this case arises when a user with clearance level A is able to deduct information contained in tables classified at a higher level B.

Definition 6. Suppose that relation R[B] which connected entities V_1 and V_2 becomes invisible as we move down one level, to level A. Then, the subgraph V_1RV_2 can be inference causing. In our case, this can be detected if the portion of the database graph that is visible at level A together with the above subgraph form a 'virtual' cycle. This means that there is an alternative path between these two entities with aggregate classification level lower than B.

The scope of this work is to examine the possibility of breaking this alternative path. Though this is costly most of the times because information is withholded, we believe that the elimination of inference that we propose can meet the cost.

3.2. Controlling inference

Definition 7. The virtual cycle that causes inference is broken with the elimination of one of its edges.

Many problems arise with this method. The criteria that are used to select the edge to be removed is a serious one. Another one is the precise definition of 'edge elimination'. Finally, the problem of accessing the resulting partitions in order to answer a posed query must be solved.

Our algorithm is run once and for all to mark the possible inference sources for each classification level. Then, each time a query is posed, we try to construct a query answer tree that does not contain any of these sources. If this is not possible, we break the query answer tree by eliminating a set of edges that breaks all inference paths. Here, we must apply another algorithm to do the job. Finally, we answer a number of subqueries with a minimun loss of information with respect to the initial query. The elimination of some edge is done by making non-accessible some attributes in the relation on this edge. It results in the partitioning of the databse into clusters. Then, we answer only those queries that do not range over more than one cluster.

Example

To make these clear, we give an example. Suppose that we have the database scheme:
E1: PROJECT(PROJNO, head) [S]
E2: COMPANY(COMPNO, cname, address, country) [U]

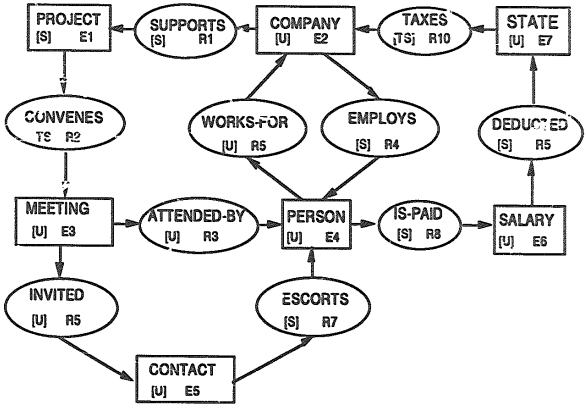


Figure 2. Extended Database Graph

E3: MEETING(MEETNO, room) [U]
E4: PERSON(SOCSECNO, name, address, phone) [U]
E5: CONTACT(NAME, phone) [U]
E6: SALARY(AMOUNT, deductions, rank) [U]
E7: STATE(NAME, address) [U]
R1: SUPPORTS: COMPANY->>PROJECT [S]
R2: CONVENES: PROJECT->>MEETING [TS]
R3: ATTENDED-BY: MEETING->>PERSON [U]
R4: EMPLOYS: COMPANY->>PERSON [S]
R5: WORKS-FOR: PERSON->COMPANY [U]
R6: INVITED: MEETING->>CONTACT [U]
R7: ESCORTS: CONTACT->>PERSON [S]
R8: IS-PAID: PERSON->SALARY [S]
R9: DEDUCTED: SALARY->STATE [S]
R10: TAXES: STATE->COMPANY [TS]

where the attributes in capitals are the entity keys and arrows indicate the flow of information. We assume that entities and relations are represented as tables, and that relation tables hold the key attributes of the related entities. We also assume a three-level classification scheme, with Unclassified<Secret<Top Secret. The database graph that results from the above scheme is shown in Figure 2.

The portion of the graph that is visible at each classification level is depicted in Figure 3, a-c. Note how the elimination of R10 breaks the path between E7 and E2, as one moves from level Top Secret to Secret, in Figure 2, a-b. This is a point of possible inference, since one can infer the contents of R10 by following the alternative path E2R5E4R8E6R9E7.

3.2.1.Algorithm that detects inference cycles

We present here a two-phase algorithm, that can be used for the detection of an inference cycle. Note that our algorithm works at the Top Secret level; therefore it has an overall view of the whole database. This cycle-detecting algorithm works as follows:

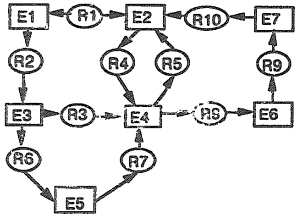
In phase one, which is executed at the database design stage, the input is graph G' and the output is a graph G_i and a set of subgraphs IS_i for every classification level c_i belongs to CL. G_i is the subgraph visible at level c_i, where IS_i is the set of inference sources in this level. We take the following two steps:

1. Move to level $c_{i-1}<c_i$. Let $G_{i-1}= (V_{i-1}, E_{i-1})$ be the portion of the initial graph that is visible at this level. Then

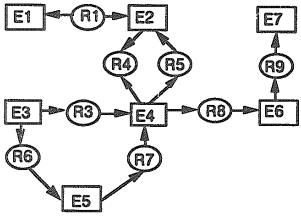
$V_{i-1} = V_i - \{(v_k, c_k)\}$, and

$E_{i-1}= E_i-\{(v_k, v_j) \mid (v_k, v_j) \in E_i \wedge \neg((v_k, v_j \notin V_{i-1}) \vee ((v_k \notin RV_{i-1} \wedge v_j \in V_{i-1}) \vee (v_j \notin RV_{i-1} \wedge v_k \in V_{i-1})))\}$

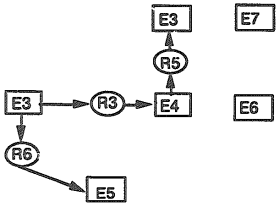
Definition 8. As we move to a lower clearance level, only the attributes with classification level at most equal to the current clearance level remain visible (Figure 4a). An entity node finally becomes invisible when the current clearance level is lower than the classification level of its key attributes (and by definition lower than the entity classification level).



(a) Top Secret level



(b) Secret level



(c) Unclassified level

Figure 3. The different subgraphs visible at each clearance level

Definition 9. As we move to a lower clearance level, only those key attributes with classification level at most equal to the current clearance level remain visible (Figure 4b). Those with higher classification levels become invisible, together with all the edges that run into them. A relation node that consists of n key attribute nodes and k relation nodes finally becomes invisible if at least n-i of its key attributes become invisible. All edges that connect this relation are deleted, even if one adjacent node is still visible. In this way, no dangling edges are left.

2. It is clear that inference is possible only if a relation node becomes invisible but the related entities remain visible. Therefore, the Inference Set for level c_{i-1} contains subgraphs formed of such a relation node together with the edges that connect it to its adjacent entity nodes, only if there is an alternative path between these entity nodes, with lower aggregate classification.

Thus, for every relation node $v_r \in RV_i \wedge v_r \notin RV_{i-1}$ and entity nodes v_p, v_q belong to EV (this is a relation between two entities only), if $P_a=\{(v_p,v_{r1}), (v_{r1},v_{s1}),...,(v_{rn},v_q)\}$ is the alternative path that connects these nodes (v_r does not belong to P_a) with aggregate classification level $c_k \leq c_{i-1}$, the inference set (initially empty) will be:

$$IS_{i-1} = IS_{i-1} \cup (v_r, ((v_p, v_r), (v_r, v_q)))$$

A path's aggregate classification level is equal to the Least Upper Bound of the nodes it is composed of. This definition can be easily expanded for relations between more than two entities. There exist many known algorithms that test the existence of such an alternative path.

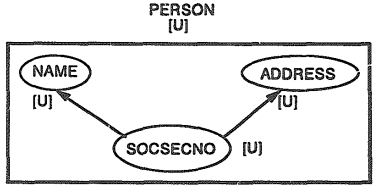
To remove one edge, we don't permanently delete any data. We simply use reclassification to make it invisible to the user. So we've marked the edge between an entity and a relation to be removed. Since we adopted a bipartite graph representation, we could reclassify at a higher level the attributes of this relation that also belong to this entity. In this way, only part of the relation is visible, and no connection can be made to the entity. Relations are used as *switches* that permit the flow of information if turned on, or restrict it, if turned off.

Eliminating edges results in the clustering of nodes. Thus the database is partitioned in clusters of nodes and queries posed over one cluster can be answered without fear of inference, but it is dangerous to answer queries that range over more than one cluster.

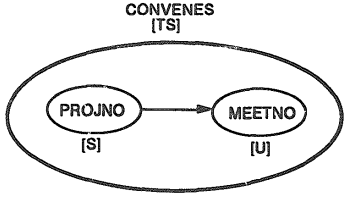
By the end of phase one, all the inference causing subgraphs are already marked. We move then in phase two, which is executed each time a query is posed. At this stage, our algorithm performs the following:

1. Each time a query q with clearance level cq is posed, it determines the corresponding query answer graph AG_q, and creates the first version of the query answer tree AT_q.
2. Then, for every $(v_r, \{(v_i, v_r), (v_r, v_j)\})$ which belongs to IS_{cq}, it checks if v_i belongs to AT_q and v_j belongs to AT_q.Note that neither (v_i,v_j) nor (v_r,v_j) belong to AT_q. We keep all these 'virtual' subgraphs in set ISQ (the *query inference set*).
3. If this is true for at least one subgraph, an algorithm that eliminates inference is called.

The subgraph that is visible at every clearance level may consist of one or more connected components.



(a)



(b)

Figure 4. Evolution of the contents of entity and relation nodes

Definition 10. A query answer tree may at most be equal to one of the connected components of the query clearance level. Thus, this connected component is the query answer subgraph AG_q. The query answer tree is a subgraph of this connected component and is formed as follows:
1. Every attribute that will appear in the result table is a leaf.
2. The minimal acyclic connected graph that contains these leaf nodes is the query answer tree.

Definition 11. A path in the query answer tree is safe when all the "virtual" cycles are removed.

3.2.2. Algorithm that eliminates cycles

After the cycle detection phase we proceed with cycle breakage phase. In this phase, the input is a graph G_u=AGqUISQ.
The output is the query answering tree $T_q=(V_q,E_q)$, $G_u=(V_u, E_u)$ is the query answer graph with all the virtual inference causing subgraphs appended to it. We define INDEGREE(v) and OUTDEGREE(v) to be the number of incoming and outgoing edges respectively for node v. This four-step algorithm works as follows:
1. For every v that belongs to V_u if

INDEGREE(v)=OUTDEGREE(v), do:

$E_q = E_q \cup IN(v)$,

$E_u = E_u - IN(v) - OUT(v)$

$D_u = D_u \cup OUT(v)$

while, if OUTDEGREE(v)≥INDEGREE(v), do:

$E_q = E_q \cup OUT(v)$.

$E_u = E_u - IN(v)$

$D_u = D_u \cup IN(v)$

where IN(v) and OUT(v) are the sets of incoming and outgoing edges respectively for node v, and D_u is a 'waste' set, where we keep all the edges that are removed; later we will try to place some of them back in the graph.
2. $\forall (p, \{(u, p), (p, u)\}) \in ISQ$:

$E_q = E_q - \{(u, p), (p, v)\}$,

$V_q = V_q - \{p\}$ and

$D = D - \{(u, p), (p, v)\}$

3. For every (u, v) belongs to D, compute W_{uv}, where W_{uv} is some weight we assign to each edge. This weight will subsequently be used as a criterion to select the edges that will be placed back in the query answer graph.
4. For every (u, v) belongs to D, place the edge back in the graph, provided that the union of the resulting graph with ISQ forms an acyclic graph. Start from the edge with minimum W_{uv}. This step is necessary, since the algorithm that breaks cycles does not remove a minimum set of edges to achieve it. As a result, more edges than necessary are removed, and we place back those that don't do any harm.

Definition 12. A query is answered safely if all the paths in the query answer tree are safe.

Example (contd.)

We will now move on to show how the algorithm that detects cycles works with our example. At the TS level, the whole graph is visible (Figure 3a); therefore
V_{TS}={E1, E2, E3, E4, E5, E6, E7, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10} and
E_{TS}={(E1,R2), (R2,E3), (E3,R6), (R6,E5), (E5,R7), (R7,E4), (E3,R3), (R3,E4), (E4,R5), (R5,E2), (E2,R4), (R4,E4), (E2,R1), (R1,E1), (E4,R8), (R8, E6), (E6,R9), (R9,E7), (E7,R10), (R10,E2)}.
As we move down to level S, relations R2 and R10 become non visible, as they are classified TS (Figure 3b); therefore
V_S={E1, E2, E3, E4, E5, E6, E7, R1, R3, R4, R5, R6, R7, R8, R9} and
E_S={(E3,R6), (R6,E5), (E5,R7), (R7,E4), (E3,R3), (R3,E4), (E4,R5), (R5,E2), (E2,R4), (R4,E4), (E2,R1), (R1,E1), (E4,R8), (R8, E6), (E6,R9), (R9,E7)}.
But, as one can easily see, one can obtain the information contained in these TS tables just by following the alternative paths

P_1[S]=E3R3E4R5E2R1E1 and

P_2[S]=E3R6E5R7E4R5E2R1E1,

for node R2 and

P_3[S]=E2R4E4R8E6R9E7,

for node R10. Thus:
IS_S={(R2, {(E1,R2), (R2,E3)}, (R10, {(E7,R10), (R10,E2)})}.

The same procedure is followed for the next lower level (Unclassified). We now proceed with phase two, where we suppose that query q:

```
select PROJNO, SOCSECNO
from ATTENDED-BY
where {
      select SOCSECNO, COMPNO
      from WORKS-FOR
      where {
          select PROJNO, COMPNO
          from SUPPORTS
          where COMPNO='KITSOS'
          }
      }
```

is posed, with cq=S. The query answer tree AT_q is formed:

AT_q={(E3,R3), (R3,E4), (E4,R5), (R5,E2), (E2,R1), (R1,E1)}

while the query answer graph is G_S (the subgraph visible at this clearance level).

Clearly, this user is using the alternative path to obtain information for the MEETINGS that certain PROJECTS SUPPORTED BY COMPANY 'KITSOS' for which a known PERSON WORKS FOR and who ATTENDS them, CONVENES. The query answer tree that is formed at the S level contains nodes E1 and E3, while ISQ={R2, {(E1,R2), (R2, E3)}} (subgraph {R2, {(E1, R2), (R2, E3)} belongs to IS_S), so this query probably causes inference.

We call next the algorithm that breaks alternative paths, with input $G_u=G_S \cup ISQ$. We check every node in the query answer graph, starting from E1 and moving counterclockwise. First we deal with entity nodes and then with relation nodes.

For node E1, INDEGREE(E1)=OUTDEGREE(E1), therefore:

E_q={(E1,R2)},
E_u=E_u - {(R1,E1)} and
D_u={(R1,E1)}.

For node E3, INDEGREE(E3)<OUTDEGREE(E3), therefore:

E_q= E_q $\cup$ {(E3,R3), (E3,R6)},
E_u= E_u - {(R2,E3)} and
D_u= D_u $\cup$ {(R2,E3)}

For node E5, INDEGREE(E5)=OUTDEGREE(E5), therefore:

E_q= E_q $\cup$ {(E5,R7)},
E_u= E_u - {(R6,E5)} and
D_u= D_u $\cup$ {(R6,E5)}

For node E4, INDEGREE(E4)>OUTDEGREE(E4), therefore:

E_q= E_q $\cup$ {(R3,E4), (R7,E4), (R4,E4)},
E_u=E_u - {(R3,E4), (R7,E4), (R4,E4), (E4,R8), (E4,R5)} and
D_u= D_u $\cup$ {(E4,R8), (E4,R5)}

For node E6, INDEGREE(E6) =OUTDEGREE(E6), therefore:

E_q= E_q $\cup$ {(E6,R9)},
E_u= E_u - {(R8,E6)} and
D_u= D_u $\cup$ {(R8,E6)}

For node E7, INDEGREE(E7)>OUTDEGREE(E7), therefore:

E_q= E_q $\cup$ {(R9,E7)},
E_u= E_u - {(R9,E7)} and
D_u remains unchanged.

For node E2, INDEGREE(E2)<OUTDEGREE(E2), therefore:

E_q= E_q $\cup$ {(E2,R4), (E2,R1)},
E_u= E_u - {(R5,E2)} and
D_u= D_u $\cup$ {(R5,E2)}

We continue now with relation nodes, commencing with R2.

For node R2, INDEGREE(R2)>OUTDEGREE(R2), therefore:

E_q= E_q $\cup$ {(E1,R2)},
E_u= E_u - {(E1,R2)} and
D_u remains unchanged.

For node R6, INDEGREE(R6)>OUTDEGREE(R6), therefore:

E_q= E_q $\cup$ {(E3,R6)},
E_u= E_u - {(E3,R6)} and
D_u remains unchanged.

For node R3, INDEGREE(R3)>OUTDEGREE(R3), therefore:

E_q= E_q $\cup$ {(E3,R3)},
E_u= E_u - {(E3,R3)} and
D_u remains unchanged.

For node R7, INDEGREE(R7)>OUTDEGREE(R7), therefore:

E_q= E_q $\cup$ {(E5,R7)},
E_u= E_u - {(E5,R7)} and
D_u remains unchanged.

For node R9, INDEGREE(R9)>OUTDEGREE(R9), therefore:

E_q= E_q $\cup$ {(E6,R9)},
E_u= E_u - {(E6,R9)} and
D_u remains unchanged.

For node R4, INDEGREE(R4)>OUTDEGREE(R4), therefore:

E_q= E_q $\cup$ {(E2,R4)},
E_u= E_u - {(E2,R4)} and
D_u remains unchanged.

Finally, nodes R1, R5 and R8 are left without any incoming or outgoing edges, so all the edge sets remain unchanged. At the end of the algorithm's first step, what is left from the original query answer graph looks like Figure 5a. Now we remove node R2 from Vq, edge (E1,R2) from E_q and edge (R2, E3) from D_u, since in effect this subgraph is

not visible at this clearance level.
We will now assign weights to the edges in D_u, a process that in the real world must be done at the database design stage. The process we use is simple: Every edge that does not belong to the AT_q is assigned a 0 weight. Every edge that connects Unclassified nodes is assigned a 1 weight. Every edge that connects an Unclassified node with a Secret one is assigned a 2 weight. Finally every edge that connects two Secret nodes is assigned a 3 weight. The simple premise that underlies this process is that it is easier to reclassify a Secret node than an Unclassified one. D_u now is as follows:

D_u={(R1,E1)[3], (R5,E2)[1], (E4,R8)[0],(E4,R5)[1], (R6,E5)[1], (R8,E6)[0]},

with weights shown in brackets.
Now, we begin to put back edge after edge, commencing with the ones with the smallest weight, and checking that the graph remains acyclic. Note that edges with 0 weight are not used at all, since they are irrelevant to the query we are trying to answer. The edges are placed in the graph in the following order: (R5, E2), (E4, R5), (R6, E5)
Now, if we place edge (R1, E1), a cycle will be formed. Thus, the query answer tree is formed (Figure 5b). As one can see, reclassifying node E1 (and as a result relation R1) to a higher level, breaks the inference path. In general, one can observe that, in the extended database graph, if there are two nodes with classification level A in cascade, then no inference for these nodes can be caused at level B, A.

3.3. Compound queries

The algorithms presented above may deal well with simple queries, but have no effect on compound queries.
Definition 13. A *compound query* is a 'virtual' query, in the sense that it is never posed as one. Instead, many simple queries are posed, the union of the answer sets of which is equal to that of the compound query.
It is clear that a compound query may well be causing inference, while the simple queries it consists of may be safe. So, if posed, the compound query would not have been answered, but the simple queries will indeed be answered. In this way, a malevolent user obtains information he is normally not allowed to access.
To deal with this situation, we propose an algorithm that uses the notion of history, that is, the answer to a query also depends on the previous queries the same user has posed. If some user continuously poses queries that cause inference, he should be marked as dangerous, and appropriate measures must be taken.

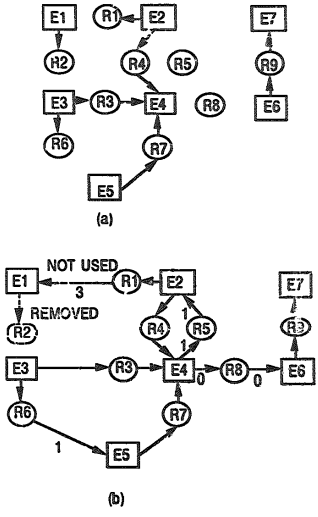


Figure 5. Construction of query answer tree.

3.3.1. Algorithm that deals with compound queries

With each user u, a graph HG_u is associated. This graph is formed by all the queries this user has posed up to date.
1. Each time a user u with clearance level cq poses a query q that can be answered safely, AT_q is saved. In effect, AT_q is appended to the graph HG_u.
2. Then, $\forall (v_r, \{(v_i, v_r), (v_r, v_j)\}) \in IS_{cq}$, it checks if $v_i \in HG_u \wedge v_j \in HG_u$. Note that neither (v_i, v_r) nor (v_r, v_j) belong to HG_u.
3. If one such subgraph is found, then an inference-causing compound query has been posed. The obvious way to deal with it is not to answer the last simple query. Note that this method can be expanded to deal with all the queries a user has posed up to date.

4. CONCLUSIONS AND FUTURE WORK

In this paper we examined the inference problem that arises in multilevel databases. We used an extended database graph model, and showed how to partition this graph to subgraphs that correspond to the various classification levels. Then we presented algorithms that detect and eliminate inference sources in order to answer a query safely. These algorithms act like filters that constrain query answer sets depending on the inference sources they contain. In the end we examined the complex case of compound queries, and presented a solution that is based on the notion of history.

We are currently examining different, more effective, criteria to be used in the removal of inference causing edges. In particular, we are working on the use of INFER function, that places a threshold on the inference causing information flow. This will eventually lead to the inclusion of more safe edges in the query answer graph, which means that more queries will be answered. Finally, we are examining the potential of the bipartite graph representation, to deduct more effective algorithms for inference detection and elimination.

ACKNOWLEDGEMENTS

We wish to greatly acknowledge the help of Prof. Christos Papadimitriou, whose invaluable advice saved us a great amount of time.

REFERENCES

[1] J.A.Goguen & J.Meseguer, Security policies and security models, IEEE symposium on security and privacy, 1982.

[2] Tsong-an Su & Gultekin Ozsoyoglou, Data dependencies and inference control in multilevel relational database systems, IEEE symposium on security and privacy, 1987.

[3] Thomas Hinke, Inference aggregation detection in database management systems, IEEE symposium on security and privacy, 1988.

[4] Matthew Morgenstern, Controlling logical inference in multilevel database systems, IEEE symposium on security and privacy, 1988.

[5] Joseph Wald & Paul Sorenson, Resolving the query inference problem using Steiner trees, ACM transactions on database systems, Sept.1984.

[6] Dorothy Denning, Cryptography and data security, Addison- Wesley, 1982.

[7] Dorothy Denning & Teresa Lunt, A multilevel relational data model, IEEE symposium on security and privacy, 1987.