Intelligent Database Retrieval By Visual Reasoning

Shi-Kuo Chang and Yi Deng
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

ABSTRACT

When a database becomes very large, user no longer has the precise knowledge of how the database is structured, nor does the user know clearly what should be retrieved. How to get at the data becomes a central problem for very large databases. Another major problem is the efficiency of database access. In this paper, we introduce the concept of personalized user profile for very large databases, and VisualNet model as a representation for such a user profile. A user profile is a small and dynamic structure, which contains the user’s knowledge of the database, serves as a starting point for the user’s access to the database, and supports intelligent database accesses. Techniques for automatic construction and maintenance of user profiles are discussed in the paper. A set of visual reasoning algorithms are presented to support intelligent and interactive user’s queries to the database, as well as to coordinate the search between the user profile and the global database.

I. Introduction

When a database becomes very large, information retrieval becomes a very costly task because of the overwhelming search space. Examples of such databases can be found in legal, medical, and geographical information systems. For instance, legal databases nowadays have data up to one trillion bytes.

As indicated in [3], when a database grows to such sizes, the user no longer knows what is in the database. Nor does the user know clearly what should be retrieved. How to get at the data becomes a central problem for very large databases.

One observation is that, for each individual user, the retrievals, in most cases, are usually concentrated in a specific and small portion in the database. Therefore, a possible way to get around the above problem is to build a user profile of the very large database, which only represents the space in the database the user has accessed. The database accesses, in most cases, can be restricted in the small space of the user’s profile. Great search efforts, therefore, can be saved. Conceptually, such a user profile contains both the user knowledge about the database, and the system’s knowledge about how the database is organized.

There are several requirements for such a user profile. First of all, the user profile should be built incrementally, i.e. for a first time user of the database, his profile to the database should be an empty set. The more the user accesses the database, the larger the profile. Second, the growth of a user profile should be transparent to the user. As a matter of fact, user should not even be aware of the existence of such a profile. What he faces looks like the whole database. Third, the user profile should be dynamic in the sense that information can be dynamically added to or removed from the user profile. Fourth, the size of the user profile should be as small as possible to achieve efficient search. Last but certainly not the least, the user profile should support intelligent database access operation, i.e. with reasoning abilities. In the following discussion, the overall database view is called global view.

Visual reasoning [3] on a database, is the process of reasoning and making inferences for database access, based upon visually presented clues. Conceptually, any database can be mapped into a space called visual space. The visual space is a multi-dimensional space, where each point represents an object (a record, a tuple, etc.) from the database. The mapping process is called data visualization. A visual clue is an example for a database access given by an user by means of icons. The basic visual reasoning mechanisms include ZOOM-IN, ZOOM-OUT, CROSS-PROJ.

Without losing generality, assume that the database is organized in certain loose hierarchy with a root on the top of the hierarchy. That is, each node in the hierarchy may have more than one parent, and parent-children relationship is defined based on attributes, i.e. a node A is a parent of node B only in terms of the attributes associated with the link, which connect A and B. In the hierarchy, data are stored on the leaves, while other parts of the hierarchy serve as index. In term of the concept of visual space, each internal node represents a region in the database, while the root represents the entire visual space of the database.

Before formally defining the VisualNet model, let’s examine, intuitively, the structure of a user profile. When a user accesses the database for the first time, his user profile is an empty set. What the user face is the whole database. From visual reasoning point of view, he faces a visual space corresponding to the entire database. Since the user’s profile is an empty set, the search for the desired objects of the user query is started from the root of the database indexing system. A subgraph is formed during the search process from the root to the desired objects. This subgraph is added as part of the user profile.

For a very large database, the number of levels from the top of the hierarchy to the bottom (real data) of the structure may be very large. In the subgraph corresponding to the user profile, many parts are simply linear links going through many levels without branches, as illustrated in figure 1(a). A search in such parts has to go through every level, which is certainly wasteful and unnecessary. As in figure 1(a), although the user does not need to access nodes "b" and "c" most of the time, the search to node "d" has to go through node "b" and "c" every time. Therefore such linear links should be collapsed as one link to provide efficient access, as shown in figure 1(b). If later on node "n" needed to be accessed, our structure should have the capability to allow the re-insertion of node "n".

Each time user access the database, the search will start from the user profile and the search may be expanded to the database view if desired objects can not be reached from the user profile. And each time of access, more information may be needed to be added to the user profile. After certain number of accesses, the user will have a relatively complete view of the database for his needs.

On the other hand, a user may only access certain portion of the database once or twice. It is obvious that the information about this kind of access should not be kept in the user profile, since otherwise, the size of the user profile may eventually grow out of control. This is the reason that the structure for the user profile should have the ability of dynamical shrinking.

0730-3157/90/0000/0459$01.00 © 1990 IEEE
Figure 1. An example of link collapse

Note that the foundation of this idea is the assumption that for a regular user, most of his database access is restricted to certain fixed small portions of a very large database. For instance, although there may be millions of volumes of books in our library, our accesses to the library are usually restricted to the books in computer science fields. Since the user profile is much smaller than the whole database, the search time to the user profile is much less than the time to search the whole database view.

The reasoning ability is necessary because when a database becomes very large, an user no longer has the meta-knowledge of the database. It becomes very difficult or impossible for an user to give precise queries to the database system. Therefore, the user's access to the database have to use the means of hints, examples, clues etc. The system should have the ability to help users form clues, hints, or examples; on the other side, the system should be able to find the user desired objects through these hints, clues or examples by means of intelligent search, or heuristics. This is exactly where reasoning cut into.

In this paper, we propose the VisualNet model, a G-net [5] based model, as a representation for user profile of very large databases. The VisualNet model will support all the requirements raised above at certain level; and moreover, reasoning activities about user query in very large databases are supported by VisualNet model. The paper is organized as follow: VisualNet model is introduced in section II. Section III will be devoted to discuss the operations which are performed on VisualNets, such as dynamic growth and shrink, link collapse, and timestamped prune. Visual reasoning on VisualNets and a learning algorithm for the user profile are discussed in section IV. The reasoning algorithms to be discussed include ZOOM-IN, ZOOM-OUT, CROSS-PROJ. In section V, an concrete example is given to illustrate the operations of user query based on the concept of user profile, and the incremental construction of the user profile. Section VI and VII will be conclusion and references, correspondingly.

II. VisualNet

A VisualNet is defined as a five tuple:

\[ \text{VisualNet} = \{P, T, I, O, u, D\} \]

where,

- \( P = \{p_1, p_2, \ldots, p_n\} \) is a finite set of places, which is used to represent the regions (internal places) or data (leaf places) in a database.
- \( T = \{t_1, t_2, \ldots, t_n\} \) is a finite set of transitions, which are used to represent the relationship among the objects in \( P \). The types of relationships are defined in \( D \).
- \( \text{I} : T \rightarrow 2^{p_1 \times \ldots \times p_n} \) is an input function, where \( N \) is a natural number, and \( D \) is the set of possible token colors. The input function, together with output function, defines the relationship among places and transitions. They also determine the token movement, including the color and number of tokens, in VisualNets.
- \( \text{O} : T \rightarrow 2^{p_1 \times \ldots \times p_n} \) is an output function, where \( N \) is a natural number, and \( D \) is the set of possible token colors.

\[ D = \{\text{UP}, \text{DOWN}, \text{CROSS}, \text{GLOBAL}\} \]

is a set of semantic functors which may be associated with each transitions. Functor UP is the reverse functor of DOWN, and vice versa. The meaning of the function will be discussed shortly.

A user model is introduced in section 1. Section II describes the VisualNet model, a G-net model, for the user model. Moreover, reasoning activities about user profile, and the incremental construction of the user profile will be discussed in the following paragraphs.

Each transition \( t \) may be in one of two states, "active" or "passive". A transition \( t \) can fire from two directions, i.e. forward fire, or backward fire. When forward firing \( t \), tokens move from its input places to its output places; while backward firing \( t \), tokens move from its output places to its input places. A transition \( t \) can forward fire iff \( t \) is active and there is one token with the same functor as \( t \) does in every of its input places. \( t \) can backward firable iff \( t \) active and there is at least one token with the reverse functor of \( t \)'s functor at every of its output places.

The D field in the definition is a list of functors which specify the possible relationship among the regions or objects represented by the places. The set of functors is defined as \( \{\text{UP}, \text{DOWN}, \text{CROSS}, \text{GLOBAL}\} \). Each functor is in the form of function \( \text{AL} \), which is a list of attributes associated with the functor. Each attribute defines the relationship among the regions or objects. The input function \( \text{I} \) and output function \( \text{O} \) in the VisualNet definition are the input and output functions, which are similar to the one in the traditional Petri-Nets [1,8]. In case of forward firing, the input function determines, once a transition \( t \), \( i = 1 \ldots m \), is fired, the number of tokens with certain colors defined by \( D \) should be removed from each of the input places of \( t \). The output function determines, once \( t \) is fired, the number of tokens with certain color defined by \( D \) are added to the output places of \( t \). In case of backward firing, the roles of the input function \( I \) and output function \( O \) are reversed.

An example of VisualNet is given in figure 2. Notice that there is a DOWN functor associated with transition \( t \), with attributes, "sm" (semantic memory), "kr" (knowledge representation),..., connecting places "psychology" and "AI", which means that AI is a branch of psychology in
term of semantic memory, knowledge representation, ... There is also a DOWN functor at \( t_2 \), which connects place "CS" to "AI". Place "AI", therefore, has two parents. There is also a functor CROSS with attributes "kr", and "reasoning", at transition \( t_3 \), connecting place "AI" and "OA" (office automation). This connection indicates that office automation and AI can be related in terms of knowledge representation and reasoning, although they don't have parent and child relationship. Notice that each place has an adjacent GLOBAL transition connecting to the corresponding place in the global view, which are not shown in the figure.

![Diagram](image)

**Figure 2. An example of VisualNet**

We now introduce the concept of visual distance. Visual distance is defined as the distance between two points in the visual space. The visual distance between two regions is defined as the distance between the center points of the two regions. Visual distance between two regions or objects can be explained in two ways. (1) structural explanation, where visual distance is defined as the number of transitions in the shortest path between two objects. In terms of database hierarchy, it is the number of levels between the two objects. (2) semantic (conceptual) explanation, where visual distance is defined purely by the semantic or conceptual difference between the two objects. If we consider database hierarchy is constructed based on the semantic relationship at certain aspects, the first definition also contain certain semantic information by default.

The concept of visual distance can be used in two ways for our purposes. (1) It can be used in VisualNet construction to control the size of user profile; (2) it can be used as a constraint in visual reasoning. The second usage of visual distance will be discussed in section IV, and the first usage is discussed below.

As illustrated in figure 1(a), during the construction of an user profile, there are often many linear links, i.e. multi-links between two places without branch. Significant time may be wasted for traversing such paths. Such linear paths are often seen with DOWN and UP links. The solution, as indicated in figure 1(b), is to collapse such linear links (multi-transition) into one link (one transition). This can be done in VisualNet as follow: If a linear path \( p_{a_1}, p_{a_2}, ..., p_{a_l} \) is detected in a VisualNet, and transition \( t_{b_1}, ..., t_{b_r} \) share the same function, the intermediate places and transitions are deleted. A new transition \( r \) with the common function is created, the names of deleted places are stored in the CPL of \( r \), and the union of the attribute lists associated with \( t_1, ..., t_r \) are stored in the AL of \( r \). The visual distance value associated with the new transition \( r \) is set to \( j + \) accumulated visual distance in the transition \( t_1 \) to \( t_r \).

### III. Operations on VisualNet structure

The success of VisualNet model heavily depends on its dynamic feature, i.e. dynamically growth and shrinkage. In this section, we will discuss several self-adjusting operations on VisualNets which support such features. The operations discussed in this section, which are to be performed on the user profile, include: insertion, link collapse, and timestamped prune.

First of all, an insertion to the user profile can only happen when a search in the user profile has to be extended to the global view. This implies that the new places to be inserted must have certain relations with the existing places in the user profile. This relation can be traced from the place where the search is extended (through GLOBAL transition) to the global view. Assume the search is extended to the global view from place \( p_1 \), the data found in the global view is \( p_2 \), which is either an internal node or a leaf node, and the closest common ancestor shared by \( p_1 \) and \( p_2 \) is \( p_3 \), as shown in figure 3. Note that \( p_1 \) and \( p_2 \) may or may not be the same node. The insertion operation discussed here is different from the insertion of other data structures, e.g. binary search trees, in that not only the place \( p_3 \) is inserted into the user profile, but also, more importantly, the search path from \( p_1 \) to \( p_3 \) is inserted to the user profile. That is, after the insertion, it is guaranteed that \( p_3 \) is reachable from \( p_1 \) in the user profile.

![Diagram](image)

**Figure 3: Extended search in the global view.**

An example of insertion to the user profile are shown in figure 4. The formal insertion algorithm is presented in [4].

![Diagram](image)

**Figure 4. Example 1 of insertion to the user profile.**

A timestamp is associated with each place in the user profile to indicate how long the place has not been accessed or searched. If a place has not been searched for certain time, say then we assume that this place may not be accessed in the future. To keep the size of the user profiles in a manageable size, such portion of the user profile should be removed. Periodically, the system should go through the user profile, and delete those places which have not been accessed for \( \delta \) time units. This operation is referred to timestamp prune. It is assumed that each time a place is accessed, its timestamp is updated. The time stamped prune is implemented in our system in the following way. The system goes through the user profile in the depth-first fashion. Whenever is place \( p \) is found not accessed in \( \delta \) time units, the entire subtree under the place \( p \) is deleted. The deletion of the subtree rooted by \( p \) is done, again, by trace the subtree in depth-first order, remove the CROSS transitions associated with the places in the subtree, and then remove the whole subtree. Although there may be places in the subtree whose timestamp is less than \( \delta \) (these are caused by the accesses through the CROSS transitions), the deletion of the whole sub-
tree is reasonable because most of the accesses are done through inheritance (DOWN and UP transitions) relationship.

Link collapse refers to the operations illustrated in figure 2, where the list of places and transitions form a linear list. Since only those places, which have been searched, are inserted in the user profile, there may be many such linear paths exist in the user profile, which will cost search time, and intermediate places and links are often unnecessary in the user profile. The link collapse operation is to delete more intermediate places and transitions. Although the insertion algorithm presented above performs the link collapse implicitly, an explicit link collapse algorithm is still necessary, since the timestamped prune, or more specifically, the deletion operations may create linear paths in the user profile. The detail of link collapse algorithm is also presented in [4].

IV. Visual Reasoning and Learning on VisualNets

As mentioned in section I, visual reasoning refers to reasoning activities of making inferences for database queries based on visually presented clues. A visual clue is some region in the visual space, plus additional directional information [3]. In this section, we discuss how to accomplish user's query to very large databases using visual reasoning on VisualNets. In section II, we suggested that a CROSS transition is used to connect two entities in the database which are related by certain attributes, and such relations are usually not reflected by the database hierarchy (i.e. the two entities don't have inheritance relationship). The CROSS transitions make the implicit relations among the entities in the visual space explicit in the user profile, and therefore, allow the direct accesses between the entities [11]. Such connections can greatly reduce the search time in some cases, and therefore, improve the database access efficiency.

The basic reasoning primitives for visual reasoning include: ZOOM-IN, ZOOM-OUT, and CROSS-PROJ. A visual reasoning activity is an intelligent combination of these reasoning primitives. A visual reasoning activity may be started from a visual example, in form of visual icons. ZOOM-IN refers to select a more restrictive region Y from the visual space X, i.e. \( Y = \text{ZOOM-IN}(X; \text{constraints}) \). ZOOM-OUT primitive has the reverse effect of the ZOOM-IN primitive. From the restrictive region X, the enlarged region Y is found, such that \( Y = \text{ZOOM-OUT}(X; \text{radius}) \). CROSS-PROJ(X; constraints) primitive, on the other hand, is to make cross-reference from the current visual space X to a space Y based on certain constraints. Notice that the resulted space Y is not necessarily a subspace of X. The constraints in each of the above primitives may be a list of attributes, or any boolean expression on the attributes. For instance, when a user tries to access all the books about knowledge representation dated after 1988, and his current visual space is on the level of computer science, he may use the ZOOM-IN predicate with the following constraints: "(attribute KEYWORD = 'kr') and (attribute DATE > '1988')."

The reasoning primitive specified above is defined by the means of transition firing and token movements. The primitives are defined as follow:

**ZOOM-IN(X; constraints):**

1. Assign a token \( t_i \) at the place \( p_i \) corresponding to visual space \( X \);
   \( \text{incolor}(t_i) = \text{UP} \);
   \( \text{incolor}(t_i) = \text{UP} \);
2. Create a result list RL;
3. When receive a token \( t_i \) at place \( p_i \):
   3.1. Compare the region/subject specification with \( \text{incolor} \) (constraints).
      If one or more of the constraints is satisfied and no other constraints are explicitly violated, then insert \( t_i \) to RL;
      otherwise delete \( t_i \), and GOTO 4.
   3.2. FOR output transition \( t_j \), with function \( \text{DOWN} \), of \( p_i \) DO
      3.2.1. duplicate one token \( t_i \) at place \( p_i \)
      3.2.2. mark \( t_j \) as active;
4. IF there is a transition fireable in the space, fire the transition, and set the transition passive;
5. Repeat step 3-4 until no transition is fireable, and select the most promising place (satisfying most constraints) in RL to represent the resulting visual space. If RL is an empty set, announce ZOOM-IN fail.

As seen in the above procedure, ZOOM-IN operation is a process of constraint satisfaction. The process stops when no transition is fireable, i.e., no constraint can be further explored. Since ZOOM-IN process operate within the current visual space, the resulting visual space must be its sub-space. Tokens carry the constraints of the ZOOM-IN operation to allow the constraint propagation. The process is data-driven since the goal is reached is unclear beforehand, and so is the following ZOOM-OUT process. In ZOOM-OUT process, tokens move along the reverse direction comparing to the token movement in ZOOM-IN process, i.e., tokens climb from lower level to upper level in the hierarchy. The lifetime of the tokens is determined by the visual distance associated with transitions and radius presented in the arguments. Instead of associating a list of constraints with the tokens, a visual distance value \( vd \) is associated with each token to record how far the token has traveled.

**ZOOM-OUT(X; radius):**

1. Assign a token \( t_i \) to the place \( p_i \) corresponding to X;
2. Create a result list RL;
3.3. When receive a token \( t_i \) at place \( p_i \):
   3.3.1. Add \( t_i \) to RL;
4. IF \( v_d > \text{radius} \) then
   3.4. Add \( t_i \) to RL;
   3.5. FOR every input transition \( t_j \), with function \( \text{UP} \) DO
      3.5.1. duplicate one token \( t_i \) at place \( p_i \), and set \( t_i \) active;
      3.5.2. ELSE discard \( t_i \);
4. IF there is a fireable transition THEN
   3.6. Mark \( t_i \) as passive;
   3.7. Repeat step 3-4 UNTIL no transition is fireable.
5. RETURN RL;

Given the reasoning primitives, it is now the time to introduce the reasoning algorithm, VisualReasoner, which uses the above primitives to reason in the visual space of user profile, and it is necessary to examine the visual space of the global view to provide intelligent and efficient database retrieval. The major concerns here, again, are how to deal with very large space whose structure is unclear to use and how to save search time in such...
VisualReasoner:

1. Get initial query or example Q from user; * Q may be a constraint list */
2. Search from the root of the user profile using ZOOM-IN(user profile, Q).
   2.1. If ZOOM-IN fails, THEN extend the search to the global view, and add result list to user profile; 
   2.2. Set the result of ZOOM-IN as current visual space CVS;
3. WHILE the current visual space CVS is undesirable DO
   3.1. IF there is a clue (may be user supplied) THEN
       CASE the clue indicates ZOOM-OUT
           i = 1;
           WHILE CVS is not large enough DO
           i = i + 1;
           CVS = ZOOM-OUT(CVS, i);
       IF CVS reach the boundary of the user profile THEN
           add the result of the ZOOM-OUT to the CVS;
       CASE the clue indicates ZOOM-IN:
           current-restraints = current-restraints ∪
           (the restraints in the clue);
           CVS = ZOOM-IN(CVS, current-restraints);
       CASE the clue indicates CROSS-PROJ:
           current-restraints = the restraints in the clue;
           CVS = CROSS-PROJ(CVS, current-restraints);
       IF the result of CROSS-PROJ contains places which are not in the user profile THEN
           add the places and corresponding transitions to the user profile;
   3.2. IF no clue presents THEN
       generate one or more typical examples from CVS, along with the typical attributes of each example;
       get user rating for every example;
       generate clues (restraints) based on user rating;

The function of the VisualReasoner, as seen in the above procedure is to control the search in both user profile and the global view. It is also responsible for the growth of the user profile. Whenever a search cannot be conducted in the user profile, the search is extended to the global view, and the result of the search is added as a new part of the user profile.

V. An Example

In this section, an example is presented to illustrate the VisualNet as a representation for the user profile, the process of interactive user query, as well as the operations on the user profile. For the sake of simplicity, the queries in this example are all based on simple attributes matching. Conceptually, any boolean expression about the attributes should be allowed in the queries.

Assume the database is composed of a set of research papers, p1, p2, . . . , p6, belonging to the various areas of computer science. The papers are stored in a loosely coupled tree structure, as shown in figure 5. The following abbreviations are used in the figure: CS (computer science), OI (office information systems), AI (artificial intelligence), HT (hypertext systems), DB (database systems), PS (problem solving strategies), RP (representation), KR (knowledge representation), INF (inference mechanisms), DM (data models), AS (access strategies), and QR (query mechanisms). The data are stored on the leaves, and the upper level nodes and links are the indices, where each node represents a region in the database, while a link, associated with a set of attributes, represents the relationship between two or more regions. For instance, OI is a parent (PS is a subregion of OI) of PS in terms of attributes {a1, a2, a3, a4}. Each link is associated with an attribute list AL, i = 1, 2, . . . . The attribute lists of the links are as follows:

\[
AL_1 = \{a_1, a_2, a_3, a_4, a_5, a_6\}, \\
AL_2 = \{a_3, a_5, a_6, a_9, a_{10}, a_{11}\}.
\]

Notice that the global view of the database may or may not be described using the VisualNet model, as long as the attributes are provided for the links.

When a user first access the database, his user profile G is an empty set. Suppose the first user query is "find the papers regarding to attributes {a1, a1}". Since G is empty, the search is started at the root of GV, i.e. region CS. The paper p6 is found using ZOOM-IN command (multiple zoom-in operations may be needed). The accepting path of the search is shown in figure 6(a), which is inserted to G using the insertion algorithm of [4]. The resulting G is shown in figure 6(b). The parameters in the transition i are function, attribute list AL, visual distance vs, and collapsed place list CPL, respectively.

Now, suppose the second user query is "find the papers about attributes {a2, a3}". The search is started from G. The first operation is ZOOM-IN from region CS to place p4. Since p4 is not the paper the user looks for, and therefore, the search is extended to the global view GV, starting from the corresponding place p4. Since p4 is a leave, ZOOM-OUT operation is used. Two resulting regions, REP and KR, are returned by the ZOOM-OUT, and the region KR is chosen by the user. The next operation invoked by the user is ZOOM-IN, and place p5 is returned, which is the paper the user wants. In this case the accepting path is shown in figure 7(a). The insertion algorithm is called to insert the path into G. The resulting G is shown in figure 7(b).

The last query in this example is "find paper about attributes {a5, a7}". Again the search starts from G using ZOOM-IN at region CS, and places p4 and p5 is returned as the results. p9 in accepted by the user, however, is not the ideal result, so the search is extended to the corresponding place in GV. Similarly, the first operation used is ZOOM-OUT result-
The path shown in GV is ZOOM-OUT and the common attribute of the accepting path is \( a_1 \). A learning rule [4] is invoked to add a new connection to G. The resulting G after the insertion and the execution of the learning rule is shown in figure 8(b).

Figure 7. The structure of G after the second query.

Figure 8. The structure of G after the third query.

Readers may notice that a potential problem of this model is that the attribute lists close to the top of the global hierarchy may become very large. The following method may be used to remit the problem: Instead of only using the primitive attributes, as in the above example, super attributes (structured concepts) may be defined, which contain other attributes as its sub-attributes (subconcepts). These concepts can be organized into a hierarchy using semantic networks [8], or G-Nets [5]. The relationship among the attributes can be found using inheritance based reasoning.

**VI. Conclusion**

The concept of the user profile is presented in this paper to conduct intelligent, interactive, and efficient accesses in very large databases. The user profile is a structure which reflects the user's knowledge about the database. The structure of the user profile is dynamic and transparent to the user, with its size much smaller than the global database hierarchy. A knowledge representation scheme, VisualNet model, has been suggested in this paper to represent user's profile of very large databases. The motivation behind the research is to make the user's database access easier and more efficient by providing intelligent and efficient search. The foundation of the idea is that a user usually accesses only a very small and stable region of a very large database; therefore, the size of the user profile is much smaller than the whole database view. We hope that most of the user's database search can be conducted in his user profile. Several reasoning primitives, ZOOM-IN, ZOOM-OUT, and CROSS-PROJ, together with the VisualReasoner are provided to enable users to conduct database retrieval in the user profile and the database hierarchy in an interactive and heuristic way through a visual interface. A set of algorithms are presented to maintain and control the size of the user profile and a learning algorithm is suggested to automatically add new connections into the user profile, which provides a shortcut for information retrieval. Although the reasoning algorithm is designed to conduct visual reasoning, we believe that the similar idea can be used in general very large database environment with or without visual language interface.

**VII. References**


