Graph Algorithms for Database Schema Design

Hua Chang

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

September 1986

ISBN 0-315-97701-9

Canada

ABSTRACT

Graph Algorithms for Database Schema Design

Hua Chang

In this report we use graph methods to represent the inference axioms for manipulation of relational database schemas. The weighted graph concept is introduced and the simplification of the inference rules can produce the closure of a given set of functional dependencies (FD) by using the Schnorr algorithm [25] with the linear average time complexity. The problem of nonredundant covers and elimination of extraneous attributes on the left and right side of FD's may be simplified into a problem of reduction of redundant arcs and redundant compound nodes which are used to map possible subsets of attributes on the left side of each functional dependency.

For a lossy join database schema D on a given set of attributes U and under a given set of functional dependencies F, we use the union of the keys of all their relation schemes in D to construct a third normal form relation scheme and compute a new keys union repeatedly until a keys union K having an empty functional dependency O is obtained. The scheme so obtained <K,O> satisfies the

third normal form conditions, K->U with respect to the closure F+ and the minimality of cardinality of keys union |k|. Only the final keys union K is a real key of U, and the scheme <K,O> is in third normal form. Furthermore, its database schema D := D\/<k,O> achieves the lossless join property. Thus, the NP-hard kernel problem [11] of the FD-graph closure in our approach can be avoided.

ACKNOWLEDGEMENTS

I wish to express my great gratitude and grateful thanks to my supervisor, Dr. B.C. Desai for suggesting the problem and for his guidance, enlightenment and the many hours he spent helping me do research on this project. His knowledge and insight stimulate my thinking and understanding of this research topic.

I would like to express my sincerest thanks to Dr. F. Sadri. His numerous clarifications and suggstions have been proven invaluable during the course of this project.

I am also grateful to Dr. P. Goyal for advices and discussions which make the completion of this project possible.

Last, but not least, I wish to express my special thanks to my wife Mei, son Victor and daughter Susan for their moral support and encouragement during my studies.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Graph and hypergraph methods can be used as efficient and formal tools for manipulation of data dependencies and their related problems in relational database schema design. Hypergraphs have been used in [1,2,3] for manipulation of join dependencies and multivalued dependencies with the universal relation assumption. In hypergraphs, attributes of database schema are represented by nodes, and functional dependencies among attributes are representd by directed edges. The concept of hypergraph is a generalization of graph where each arc links only two nodes. Trees and directed acyclic graphs have been used in [4,5] to determine whether a functional dependency can be derived from a set of functional dependencies by Armstrong inference rules [6]. Multigraphs have been used in [7] to represent functional dependencies and multivalued dependencies. In the later approach, the nodes correspond to the attributes, and three kinds of labeled arcs are used to describe the dependencies among attributes. An extension of the usual concept of graphs by introdction of compound nodes have been used in [8,9]

for functional dependency manipulation. In this approach [8,9], two types of labeled arcs are used to represent the functional dependencies among compound nodes and single nodes respectively.

We would expect that if a graph can carry maximally the required information, then it will simplify the related manipulation. In the abovementioned papers [7] and [8,9], the graphs may carry only two or three kinds of information by their arc labels respectively. Naturally, weighted directed graphs will eliminate the above limitations. There may be many kinds of arcs with different labeled weights. As we shall see and prove later that graph manipulation and Armstrong inference rules may be simplified when weighted graphs are used. Consequently, it also simplifies all the related problems, such as the FD-closure problem and the minimum cover problem under a set of functional dependencies.

CHAPTER 2

BASIC CONCEPTS OF DATABASE SCHEMA DESIGN

In relational database manipulation, data dependencies are used as important constraints on relationships between data. This report is concerned with only functional dependencies. A functional dependency (FD) X->Y holds in a relation scheme (XY, X->Y) if and only if (abbreviated iff) each value of X is associated with exactly one value of Y. A database description is called a database schema. A database schema consists of a set of relation schemes. Database schema and relation scheme are syntactic objects, database and relation are database contents. In a number of recent papers [1-9], the universal relation assumption is introduced; this concept provides a simple user interface and it allows us to specify relations in terms of attributes of a universal relation.

Usually, database schema design is related to database normalization. There are two approaches for normalization : normalization through synthesis [4,12,13] and normalization through decomposition [14,18]. Both the

methods use a universal relation scheme <U,F> as an input, where U is a set of attributes and F are data dependency constraints. The decomposition method stepwisely decomposes the universal scheme into more and more simple schemes. The synthesis method searches simple components in a universal relation scheme. Database schema produced by decomposition has the lossless join property with respect to the universal scheme <U,F>. The synthesis approach achieves only the third normal form (3NF). It may not have the lossless join property, for which the kernel problem [5,10] of the FD-graph closure usually is required to be solved. Our graph algorithms are closely related to the synthesis approach. In this approach, a *minimum cover of data dependencies* needs to be solved.

We need now to list some definitions and notations which are used in the next chapters.

Let us define a universal relation scheme as <U,F> where U = {A1, A2, ..., An} is a finite set of attributes, and F = {R1->S1, ..., Rm->Sm} is a set of functional dependencies. We use A, B, C, ... for single attributes and use ..., X, Y, Z for subsets of attributes. A database schema is a finite set of relation schemes D = { <X1,F1>, ..., <Xk,Fk> }.

A in Xi is a prime attribute of <Xi,Fi>, iff A is an element of any key of <Xi,Fi>. A relation scheme <Xi,Fi> is in 3NF iff none of its nonprime attributes is transitively dependent on any of its keys.

Chapter 3

GRAPH REPRESENTATION AND INFERENCE RULES

In this chapter we shall introduce weighted directed graphs denoted by FD-graphs for representation of functional dependencies. Our work is based on the graphs suggested by Ausiello et al [3]. We shall show that our weighted directed graph concept may simplify manipulation of functional dependencies.

Definition. Given a set of functional dependencies F on a finite set of attributes U = {A, B, C, ... }, the FD-graph G = (V,E) representing F is defined such that :

(a) for each single attribute A in U, there is a node labeled A in V0 where V0 represents a set of nodes for single attributes.

(b) for each functional dependency of single nodes A->B in F, there is a directed full arc --> from a node labeled A to a node labeled B in E0 where E0 represents a set of full arcs. The arc weight for a such full functional dependency of A-->B is defined as w(A,B) = 1.

(c) for functional dependency containing a subset of attributes on its left side such as X --> A, then there

exists a compound node labeled X in V1 where V1 represents a set of compound nodes. To represent the reflexivity of Armstrong axioms [6], there exist |X| dotted arcs with weight w=1 for each full functional dependency from the compound node X to its |X| component nodes. The arc from the compound node X to the node A is a full arc. To represent the union rule, there exist |X| dotted arcs with weight w=1/|X| for each partial functional dependency from its |X| component nodes to their compound node X. All the |X| component nodes together fully functionally determine the compound node X with the total w=1.

We can show that a schema with n attributes may produce 2**n - 1 nodes. There are n single nodes for single attributes and (2**n - n -1) compound nodes for all the combinations of subsets of attributes. All these nodes can be coded as a set of n-bit patterns. For example, if n=3, then 001 010 100 represent three single nodes and 110 011 101 111 represent four compound nodes. The number of compound nodes is not a linear function of the number of attributes. However, most of the (2**n -n -1) compound nodes represent only the reflexivity to their component nodes. Since the reflexivity is independent of functional dependencies, these compound nodes may be deleted from the graph. For example, let n be the number of attributes and |F| be the number of functional dependencies, then at most |F'| number of compound nodes may be involved in the given set of functional

dependencies F where |F'| is the number of functional dependencies having a subset of attributes on the left side of a functional dependency. Thus, the number of compound nodes needed to specify the functional dependencies is of the same order as |F|. Furthermore, many of the |F'| compound nodes may be redundant. We shall demonstrate how to reduce redundant compound nodes from a given set of the |F'| compound nodes in chapter 4.

Example. Given a set of functional dependencies F = {AB-->D, BC-->D}, the corresponding FD-graph G = (V,E) is defined in Figure 1:

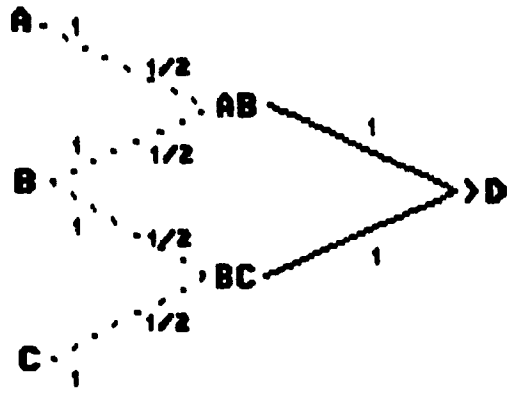

Figure 1.

where A.....AB represents both the dotted arc A<.....AB with the weight 1 and the dotted arc A.....>AB with the weight 1/|AB| = 1/2.

It is easy to see from Figure 1 that such an FD-graph representation has some advantages. Each directed arc

carries a weight information. If the weight of an arc is equal to one, then there is a full functional dependency, such as the full arc in AB-->D and the dotted arc in AB...>A. The weight of the dotted arc A...>AB is equal to 1/|AB| = 1/2 indicating that it represents the fact that node A partially determines the compound node labeled AB.

Now we need to find the inference rules for the weighted FD-graph representations. The following set of inference axioms can be proved as a complete and independent set.

For X, Y, Z, W subsets of U :

A1 (reflexivity) Y in X implies X->Y.

A2 (transitivity) X->Y and Y->Z imply X->Z.

A3 (union) X->Y and X->Z imply X->YZ.

The augmentation, projectivity, and pseudo-transitivity can be derived from the subset of A1, A2 and A3.

A4 (augmentation) X->Y and Z in U imply XZ->Y.

A5 (projectivity) X->YZ implies X->Y.

A6 (pseudo-transitivity) X->Y and YZ->W imply XZ->W.

Let us derive axioms A4, A5 and A6 using the axioms reflexivity, transitivity and union.

Augmentation : use A1 to obtain XZ->X (X in XZ). According to A2, XZ->X and X->Y imply XZ->Y. Thus, augmentation has been derived.

Projectivity : use A1 to obtain YZ->Y (Y in YZ). According to A2, X->YZ and YZ->Y imply X->Y.

Pseudo-transitivity : we just proved that X->Y and Z

in U imply XZ->Y, and Z in XZ implies XZ->Z. According to A3, XZ->Y and XZ->X imply XZ->YZ. Using transitivity, XZ->YZ and YZ->W imply XZ->W. Thus, pseudo-transitivity has been proved.

Given axioms A1, A2 and A3, we can prove the rest. Thus, A1, A2 and A3 form a complete subset in A1 to A6. It is also an independent set since none among them can be derived from the other two.

Corresponding to the complete and independent set of the inference axioms, we need to find the rules suitable for manipulation of FD-graphs. In this section, we shall define a rule that can implicitly include all the three axioms, namely the reflexivity, transitivity and union.

Definition. Given an FD-graph G=(V,E) and two nodes i and k in V, a weighted directed path (i,k) from i to k is defined as the following :

(a) Transitivity : if there exist a node j in V, an arc (i,j) with weight w(i,j)=1 and an arc (j,k) in E, then there is a path (i,k) with the weight w(i,j) * w(j,k) where w(i,j) and w(j,k) are the weights for arc (i,j) and arc (j,k) respectively.

(b) Additivity of the arc weights : if there exists a compound node j composed of its component nodes j1, j2, ..., jn, then the weight of the arc (i,j) is defined as the sum of w(i,j1) + w(i,j2) + ... + w(i,jn).

(c) If an arc with weight w(i,j) < 1, then the path (i,k) may not be formed using j since any arc or path (i,j) with

a fractional weight must be terminated at the compound node j.

(d) An FD-path (i,k) is defined as dotted if all its arcs leaving i are dotted, otherwise it is full.

Example. given a set of functional dependencies F = {E->B, AB->D, BC->D}, the corresponding FD-graph is depicted in Figure 2 :

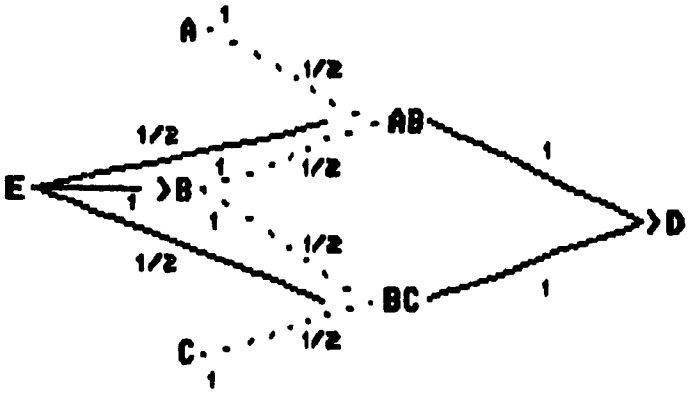

Figure 2.

We use only the graph transitivity rule. The graph union rule is implied by the additivity of the arc weights. Since the reflexivity rule A1 is independent of functional dependencies, and the reflexivity is also implied by the definition of compound node. Therefore, the above defined graph transitivity, the additivity of the arc weights and the definition of compound node implicitly include all the three inference axioms. It is worth to note that if a set of functional dependencies consists only of "single to many" dependencies, i.e. there is only a single attribute

on the left side of each functional dependency, then the weight of any arc is equal to one and there is no compound node. The above rule becomes the classical transitive operations on the graph.

If we use weighted FD-graphs and are able to transform a complete set of the inference rules into a weighted transitivity rule, then many FD-graph problems may become very close to the classical graph problems, such as the transitive closure [19] and the transitive reduction problems [20].

Now we use some examples to demonstrate the weighted graph concept.

Let us first take a primitive example to demonstrate the basic ideas. U = {A, B, C} and F = {AB->C, C->B}. Examples with complicated graphs will be demonstrated at a later time. Since the reflexivity rule is independent of the given set of functional dependencies, the compound nodes BC, CA and ABC are redundant and they are not needed to put on the following graph :

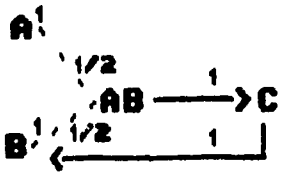

Figure 3.

Since the reflexivity rule is implied by the related compound nodes, we can derive all the possible functional dependencies for a three-attribute system if we put the compound nodes involved onto the above graph. For example, assume there is an attribute D not involved in any functional dependency of a graph. If there exists a functional dependency B->C and we need to infer BD->CD, then we simply add a compound node BD and a compound node CD on the graph. BD->CD can then be depicted as shown in Figure 4:

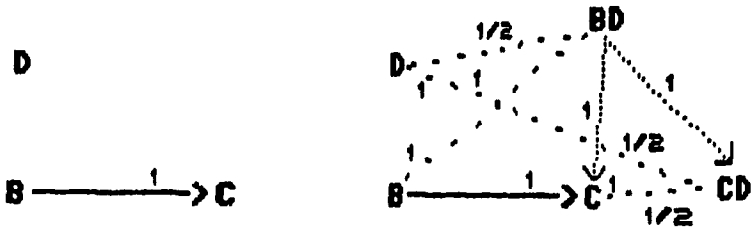

Figure 4.

It is well known that the derivation of a functional dependency from a derivation sequence was suggested by Beeri, Bernstein [4] and Maier [5]. Each step of such a sequence can be clearly depicted by our weighted graphs. For example, let F = {AB->E, AG->J, BE->I, E->G, GI->H}. Its graph is depicted in Figure 5.

It is easy to show that the following functional dependencies can be derived from the graph of Figure 5 : AB->BE, AB->I, AB->G, AB->GI, AB->H, AB->GH, EI->GH, AB->EGHI. For clarity, here we depict only the paths for

derivation of AB->GH as shown in Figure 6.

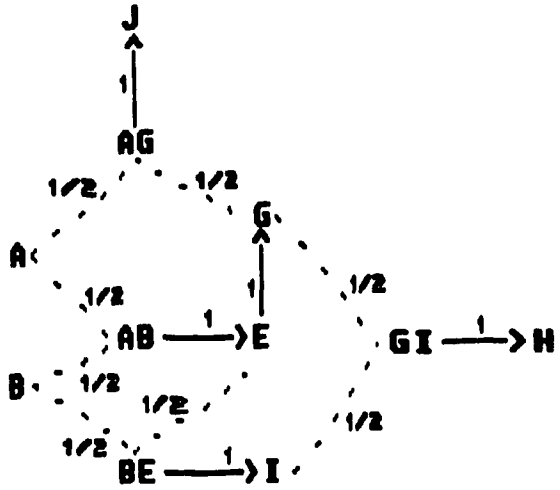

Figure 5.

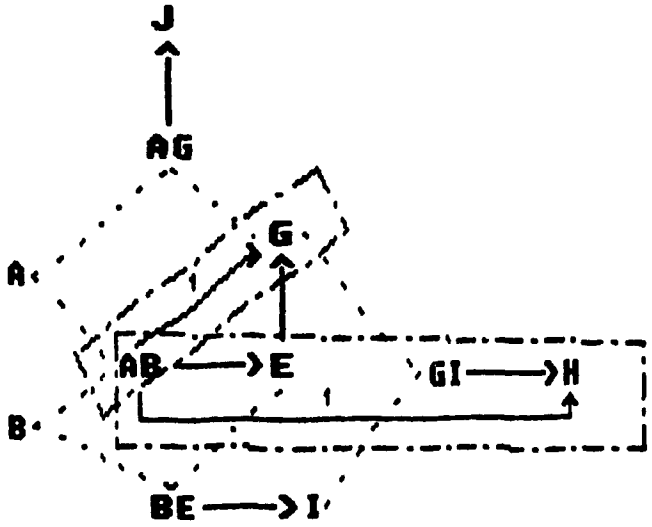

Figure 6.

It is worth to note that most of (2 ** |U| - |U| - 1) compound nodes are redundant if they represent only the

reflexivity. Thus, the total number of nodes of an FD-graph is less than or equal to (|U| + |F'|) where |F'| is the number of functional dependencies having a subset of attributes on their left side. We have shown that a complete and independent set of the Armstrong axioms A1, A2 and A3 can be implied from the weighted transitivity rule on weighted FD-graphs. It is also not difficult to show that the axioms A4, A5 and A6 can be represented on weighted FD-graphs.

A4 (augmentation) X->Y and Z in U imply XZ->Y.

According to the given FD X->Y and Z in U, a compound node XZ can be formed. Since all the component nodes of X are also the components of XZ, therefore, XZ->X. Using the given condition X->Y and A2, we obtain XZ->Y as shown in Figure 7:

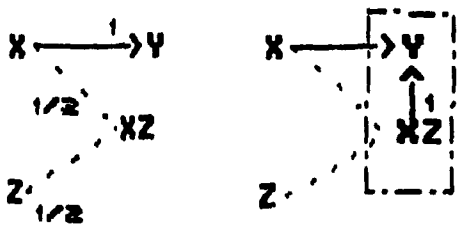

Figure 7.

A5 (projectivity) X->YZ implies X- Y.

The graph representation of X-YZ may be depicted as the following. Here we do not represent the component nodes for X, Y and Z explicitly. For clarity, assume X = {A, B, C}, Y = {D, E} and Z = {F, G}. We can also show that X->YZ implies X->Y. In such a case, X is a compound node

ABC, Y is a compound node DE and Z is a compound node FG. Thus, ABC->DEFG implies ABC->DE, i.e. X->D and X->E. This is shown in Figure 8:

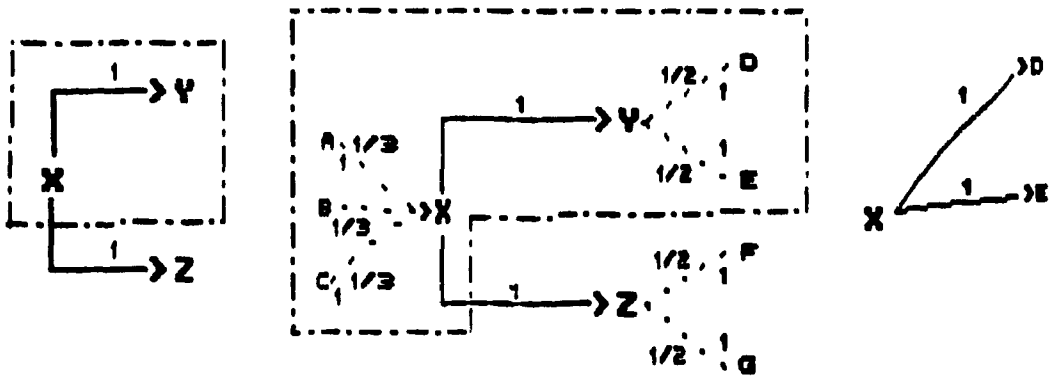

Figure 8.

A6 (pseudo-transitivity) X->Y and YZ->W imply XZ->W. According to XZ->W, XZ must be a compound node. Using the graph represetation of A4 and the union rule, we obtain :

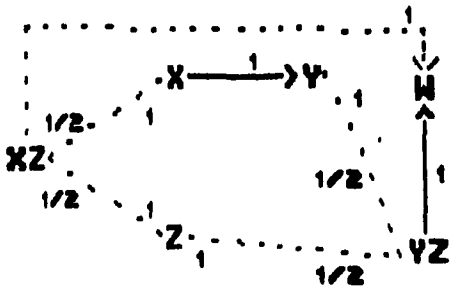

Figure 9.

For simplicity, not all the component nodes of the compound nodes X, Y, Z and W are listed explicitly on the above three graphs.

3.1 The FD-graph closure and the transitive closure problems

Let us demonstrate how the FD-graph closure problem

is closely related to the transitive closure problem when the weighted graph concept is used. Let F be a set of functional dependencies for a relation scheme r(U). The closure of F, denoted by F+, is the smallest set containing F such that the inference rules A1, A2 and A3 cannot be applied to the set F to get a functional dependency not in the set. Correspondingly, we define the closure of an FD-graph G=(V,E), denoted by G+ = (V, E+), with functional dependencies F under our weighted graph transitivity rule that implies the union rule through the additivity of their arc weights.

Definition. The closure of an FD-graph G=(V,E) is the graph G+ = (V, E+) where the set of nodes is the same as in G. The arcs E+ are defined as the following :

Dotted arc : (E+)1 = { (i,j) | i,j are in V, there exists a dotted FD-path (i,j), and its weight is equal to the product of the arc weights alone the FD-path }.

Full arc : (E+)0 = { (i,j) | i,j are in V, (i,j) not in (E+)1 and there exists a full FD-path (i,j) }.

Example. Given an FD-graph G=(V,E) with F = { A->BCD, BCD->E, CD->E, E->H }, its FD-graph is depicted as the following :

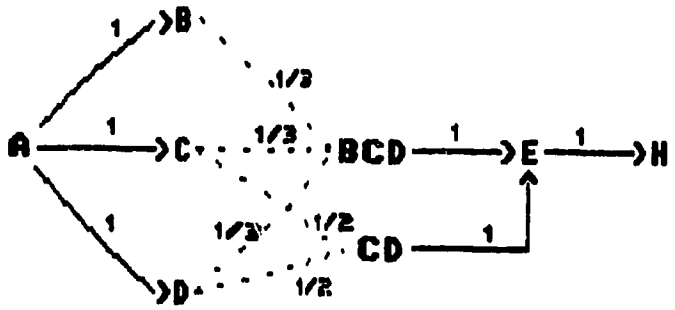

Figure 10.

then its FD-closure is depicted as the following.

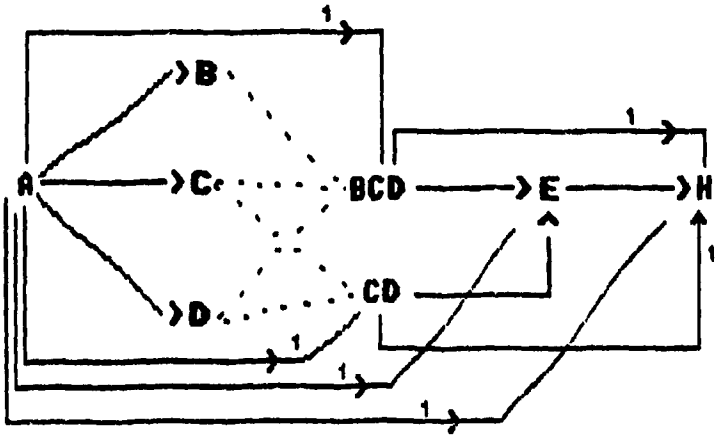

Figure 11.

Let us now give the algorithm for the FD-closure and define the adjacency lists and arc weights as the following :

L0[i], L1[i] : input set {j|j is in V and (i,j) in E0, E1};

L0+[i], L1+[i] : output set {j|j is in V and (i,j) in E0+, E1+};

w0[i,j], w1[i,j] : input weight for each arc in E0, E1;

w0+[i,j], w1+[i,j] : output weight for each arc in E0+, E1+;

c[i] : composition list {j|j is a component node of compound node i};

s[i] : working set for L0+[i] or L1+[i];

```
Initialize w0+[i,j] = w0[i,j] for each arc;
           w1+[i,j] = w1[i,j] for each arc;
#If w0+[i,j] or w1+[i,j] is not defined
#Initialize w0+[i,j] := 0 or w1+[i,j] := 0;
#Endif
/* such construct can avoid having |V|**2 matrix elements
*/

For all i in V with outdegree > 0 do
  {
   L0+[i] := L1+[i] := empty;
   s[i] := L0[i] \/ L1[i];
   while s[i] not = empty  do
     {
      select j from s[i];
      /* closure includes only meaningful full FD's */
      if w0+[i,j] = 1 then L0+[i] := L0+[i] \/ j
      else
        if w1+[i,j] = 1 then L1+[i] := L1+[i] \/ j;
      for all k in L0[j] \/ L1[j] do
        {
         if w0+[i,j] >= 1 then
            w0+[i,k]:= w0+[i,k] + (w0+[j,k] or w1+[j,k]);
         if w1+[i,j] >= 1 then
            w1+[i,k] :=w1+[i,k] + (w1+[j,k] or w0+[j,k]);
         if (w0+[i,k] >=1 OR w1+[i,k] >=1) AND
             NOT (k in s[i]) then s[i] := s[i] \/k;
```

```
        }
      }
       for all arcs (i,j) in E do
     /* for all arcs in E0 and E1,
      here the subscript 0 and 1 are not indicated */
       {
        if w[i,j] < 1 AND NOT (i in c[j]) then
        /*only component node may have w<1, otherwise w=0*/
           w[i,j] := 0;
        if w[i,j] > 1 then w[i,j] := 1;
        /* for normalization of possible multiplicity */
       }
   }
```

Let us consider the above algorithm. Suppose that j is a single node, then w[i,j] must be equal to one and the path from node i to node k through node j is established by the transitivity rule. If j is a compound node, then w[i,j] might be equal to one or less than one. If it is one, then the path (i,k) is established by the arc (i,j) and the arc (j,k). If the weight w[i,j] is less than one, then the arc (i,j) must be terminated at the compound node j since only its component node involved may have fractional weight. Such a node may not create a full functional dependency over the compound node. Suppose that w[i,k] < 1, then a full path can be formed only if the accumulated weight w[i,k] := w[i,k] + w[i,j] * w[j,k] reaches to one by the additivity of the arc weights via

all its component nodes j. Here the union rule is implied.

To prove correctness of the closure algorithm, it is sufficient to show that the algorithm examines every FD-path in G=(V,E) for transitivity operations and their weight is not overassigned which implies the union operation.

Suppose that there is an FD-path (s,t) from node s to node t in G=(V,E), then node t is reached from node s via consecutive sets L[s] containing i, L[i] containing j, L[j] containing k, ..., L[r] containing t, i.e. s -> i -> j -> k, ... r -> t. Thus, any path starting from node i is examined via the local sets L[j], L[k], L[l], ...

Every weight cannot be overassigned since weight is assigned at each step only for each outgoing arc for the processing node i keeping L[i], L[j], L[k], ... unchanged.

Note that if L[i], L[j], ... = {1, 2, ..., |V| } and all the functional dependencies have a single attribute on its left side, i.e. w[i,j] = 1, then the FD-closure algorithm becomes Warshall's transitive closure algorithm [19]. There will be two passes over nodes i = 1, 2, ..., |V|, and the time complexity of the algorithm was obtained as o(|V| ** 3). However, the time complexity of the algorithm of the closure of FD-graph is o(|V'| |E|) where V' is the set of nodes having outdegree greater than zero. This algorithm processes each node of V'. For each node,

this algorithm is running no more than the sum of |L1| + |L2| + ... and it is equal to o(|E|). Thus, the time complexity is o(|V'||E|) where V' equals the total length of input string of all the left sides of a given set of functional dependencies F.

The complexity of the Warshall's algorithm is o(|V| ** 3). The gain in the complexity of the algorithm in this report is obtained due to using local sets L[i] instead of using a (|V| ** 2) matrix. Thus, the possible redundancies of the Warshall's sparse matrices are eliminated. Furthermore, our algorithm can also be used as a procedure for the node closure under a given F. The closure of a set of functionl dependencies, denoted by F+, may be considerably larger than its F. If we want to test only one functional dependency X->Y in F+, then this algorithm can generate only the closure of a node, such as for X. It is very efficient because it need not generate all the functional dependencies of F+. It simply replaces V by X into the algorithm.

In the above algorithm, there is a statement which is very meaningful :

" If w[i,j] >= 1

then w[i,k] :=w[i,k] + w[i,j]*w[j,k]; "

This statement implies the union rule by the additivity of the weights w[i,k] := w[i,k] + ... The transitivity rule is represented by the product of the weights w[i,j] * w[j,k]. The "if w[i,j] >= 1" Boolean condition guarantees

that an arc having a fractional weight cannot be transitively over a compound node. Thus, it makes the union operation and the transitivity operation non-commutative. For example, the node A in the graph <U,F> = {A->BCD, BCD->E} can be transitive to node E only after the weight at the arc from node A to the compound node BCD is accumulated to one. According to our definition, integer weight represents a full functional dependency. However, multiple paths may produce multiplicity of arc weight. For example, A-->B-->D and A-->C-->D produce w[A,D] = 2. For the weight normalization, the above statement should include such situation, and it may be written as :

" if w[i,j] >= 1 then w[i,k] := w[i,k] + w[j,k]; "

where w[i,j] is implicitly assumed to equal one, so it need not to be written explicitly as w[i,j] * w[j,k].
To avoid multiplicity of arc weights, all integer weights must be finally normalized to one. All fractional weight arcs except those involved with compound nodes should be finally deleted since they are not meaningful.

In comparison with the Ausiello's algorithm [3], the union operations are much simplified in our algorithm because the union operations are implicitly included in the transitive operations by the arc weight additivity. Therefore, our algorithm is simple and easy to be interpreted.

Our algorithm can be used for the node closure or the

membership algorithm [5]. For example, let F = { A->D, AB->E, BI->E CD->I, E->C }, find the node closure of AE or test the membership AE->B in F+. The related graph fragment can be depicted as the following :

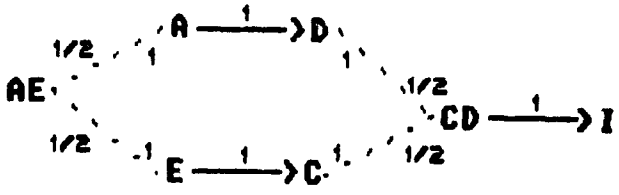

Figure 12.

then we obtain the node closure (AE)+ = {A, E, D, C, I}, and from its node closure find that AE->B is not a member in F+. The complexity of the node closure and the membership algorithm is o(|E|) and it is the same order as in the works of Maier [5] and Bernstein [4]. However, the concept in this report allows us to make further reduction of the complexity. It will be shown in the next chapters.

3.2 An efficient algorithm for FD-graph closure

As described in the above sections, the inference rules of functional dependencies may be implied from the transitivity rule on weighted FD-graphs. It may compute the FD-graph closure as with the classical graph transitive closure problem when the problem with arc weights can be solved. Except the additivity of arc weights and the meaning of compound nodes, the FD-graph

closure problem and the classical transitive closure problem are very similar. It is well known that significant progress has been made in solving the classical transitive closure problem. It is interesting to use the results obtained in the transitive closure problem to the FD-graph closure problem.

Schnorr [25] has suggested an algorithm to solve the transitive closure with linear average time complexity $o(n + m+)$ where $n=|V|$ and $m+=|E+|$. We shall show that an efficient algorithm for the FD-graph closure can be developed if we use the Schnorr algorithm [25] with some modifications. We define that an FD-graph G=(V,E) is determined by a set of arcs E and a set of nodes V=V0\/V1 where V0 represent single nodes for single attributes and V1 represent compound nodes for subsets of attributes. The adjacency lists L0[i] and L1[i] represent full arcs and dotted arcs respectively. Full arcs and dotted arcs should be manipulated separately. In case of L[i] without index 0 or 1, it is used either for full arcs or dotted arcs. Since there are no arc weights in the Schnorr algorithm, the arc weight concept must be introduced into the Schnorr algorithm. Furthermore, any modification should not violate the established linear complexity in the Schnorr algorithm.

Before we apply the Schnorr algorithm [25] to FD-graphs, we need to describe the Schnorr algorithm briefly. Schnorr assumes [25] that all possible random

graphs having n nodes and m arcs are equally probable. Based on this assumption, the average time complexity is O(n + m+) where m+ is the expected number of arcs in the transitive closure. The probability that the algorithm takes more than n**2 steps is less than 2**(-n) for all n.

The input node set is represeted by its adjacency lists L[i] = {j| i->j is in E) for i=1, 2, ..., n. The adjacency lists of the arc reversed graph is defined as (L[i])r = {j| j->i is in E) for i=1, 2, ..., n. j is called a successor of i and i is called a predecessor of j if there exists a path from i to j. The lists (L[j])r are constructed in a linear order by inserting i into (L[j])r for all j in L[i] in the order of succession i=1,2,..,n. The linearly ordered lists L[i] can be obtained by two times of application of the algorithm for the edge reversed graph : ((L[i])r)r -> L[i]. Both the L[i] and (L[i])r are needed in the next algorithms. The time for the arc reversal takes O(n + m) steps.

The Schnorr algorithm :

stage 1. Associate in a breadth first search manner to each node i a list s[i] of successors such that either (i) or (ii) holds:

(i) |s[i]| < tranc(n/2) + 1 and s[i] is the complete list of successors of i.

(ii) |s[i]| = tranc(n/2) + 1.

stage 2. Apply stage 1 to the arc reversed graph, i.e. associate to each node i a list p[i] of predecessors such that either (i) or (ii) holds with p[i] substituted for s[i].

stage 3. For i=1,2,..,n. form the adjacency lists

L*[i] = s[i]\/{j|i in p[j]}\/{j| |s[i]|=|p[j]|=n/2+1}

of the transitive closure as unions of three lists each.

The algorithm of the stage 1.

```
Initialize Q[j] := empty for all j;
For i = 1 to n do
  {
   s[i] := queue := i;
   stack:= empty;
   count:= 1;  mark i;
   while queue not = empty and counter < trunc(n/2) +1 do
     {
      j := top node of queue;
      remove j from top of queue;
      push j onto the top of stack;
      while counter< trunc(n/2)+1 AND L[j] not= empty do
        {
         a := first node of L[j];
         L[j] := L[j] - a;
         Q[j] :=Q[j]\/a;
         if a is not yet marked then
           {
            push a to the bottom of queue;
```

```
            mark a;
            s[i]:=s[i]\/a;
            counter:=counter + 1]
           }
        }
     }
for all j on stack do
  {
   unmark j;
   L[j] := L[j]\/Q[j];
   Q[j] := empty
  }
end
```

Stage 3 can be implemented in a linear time of their input data. Stage 2 is similar to stage 1. Therefore, it is sufficient to analyze the algorithm of stage 1. The algorithm is operated in a breadth first search manner. It ensures some global random properties of the sequence of visited nodes during the construction of s[i]. The restriction of (n/2 +1) steps reduces greatly the overlapping of the traversed arcs during the construction of s[i] and p[j] for i,j = 1, 2, ..., n. Furthermore, if |s[i]| = trunc(n/2+1), then node i is connected to all those j whose |p[j]| = trunc(n/2+1). Similarly, if |p[j]| = trunc(n/2 +1), then all those i whose |s[i]| = trunc(n/2 +1) are connected to j. This is a trick of the Schnorr

algorithm and it is based on the connectivity property of the classical graph, but it does not exist in weighted FD-graphs. This is a major problem in our case. In classical graphs, it is clearly that if |s[i]| = |p[j]| = trunc(n/2 +1), then s[i]/\p[j] cannot be empty since this implies |s[i]\/p[j]| > n. Non-empty of s[i]/\p[j] implies that i is connected to j. It is good for the transitive closure, but not for the FD-graph closure. It cannot provide the necessary weight information to determine whether i is fully functionally connected to j or not. We have to propose another approach.

As we discussed earlier, the weighted graph concept may combine all the Armstrong rules into one weighted transitivity rule. If we can decompose the manipulation on an FD-graph nearly into a maximal set of the transitivity operations and a minimal set of the union-transitivity operations separately, then the part of the transitivity operations can be solved by the Schnorr algorithm directly.

Let us decompose the FD-graph closure into two parts :

(1) the integration of fractional weight arcs for the union-transitive operations for each compound node within an area of a minimal number of nodes

(2) the remaining transitivity operations on the FD-graph.

Our weighted graph has the advantage to separate

functional dependency manipulation into the abovementioned parts. If we desire to switch off all the union operations, then we assign zero weight to all the fractional weight arcs. The remaining graph has only the transitivity operations to be performed. If we want to focus on the union operations, then we need only to make the related integration of fractional weight arcs.

For illustration of the upper bound of our proposed algorithm, first we propose the following working algorithm. The final algorithm will be listed after this one as an improved version.

The algorithm version 1 :

Stage 1. Associate with each compound node i of an arc reversed FD-graph its complete list p[i] of predecessors, i.e. compute only the node closure for each compound node on its arc reversed graph by using the algorithm of the section 3.1 .

Stage 2. Assigning zero weight to all the fractional weight arcs, switch off the already performed union operations in stage 1.

Stage 3. Use the Schnorr algorithm to perform the remaining only transitivity operations on all the integer weight (w=1) arcs.

Example. Let a given FD-graph be depicted as the following where the dotted arcs GH...>G and GH...>H are not listed :

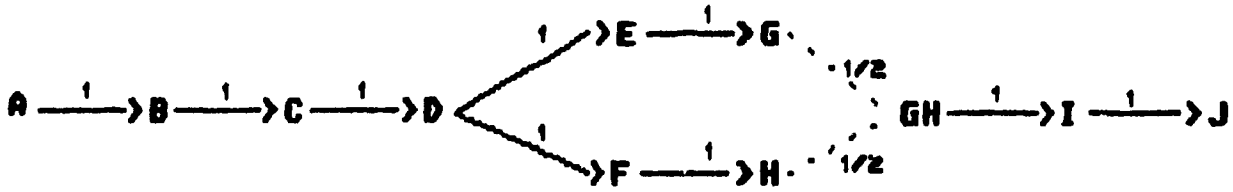

Figure 13.

then |V0\/V1| = 11 and (n/2 + 1) = 6. If the node A takes (n/2 + 1) = 6 successors including A itself : s[A] = {A, B, C, D, E, F}, then both the lists s[A] and p[J] are overlapped on the node E, i.e. their intersection s[A] /\ p[J] = {E}. However, it does not guarantee the existence of a full connectivity between the node A and the node J in the above FD-graph because here exists the fractional weight problem, i.e. the overlapping on the node E contributes only w[D,GH] = 1/2.

To solve this problem, we might first compute the complete list of predecessors for each compound node. Thereafter, assigning zero weight to all the fractional weight arcs, perform the remaining only transitive operations on the above FD-graph. For such a graph, we can directly use the Schnorr algorithm without any difficulty. Its time complexity is o(n + m+) where n is the number of nodes and m+ is the number of expected arcs. Obviously, the proposed computation of the complete list p[GH] = {D, C, B, A} using the node closure algorithm contains too many redundant operations. It is sufficient to compute only the connections between the compound node

GH and its nearest node D in stage 1, i.e. find the nearest neighbour having a full functional dependency to GH, w[D,GH] = 1.

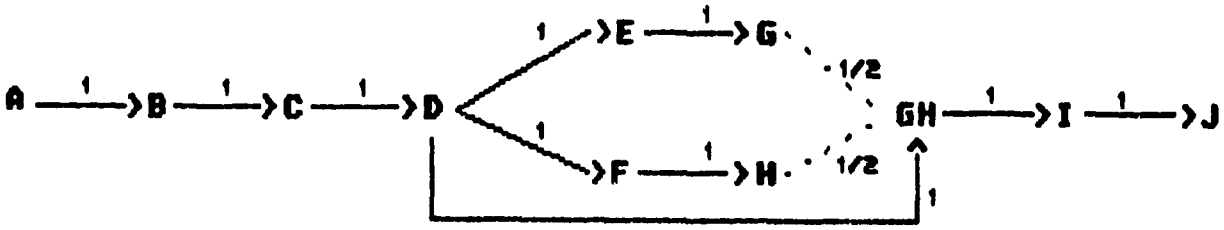

Figure 14.

However, finding such a local area for integration of fractional weights and estimation of its time complexity are not a simple problem. Our algorithm version 1 is specially designed to demonstrate the first approach and to show how to improve it. It can provide the upper bound of the complexity. Stage 2 and 3 contribute no more than o(n + m+) steps [25]. Stage 1 contains o(|V1| |E|) steps as described in section 3.1 . Therefore, the total time complexity o(n + m+ + |V1||E|) which is better than the existing algorithms with the complexity o(|V||E|) [3] where |V| is greater than |V1|, even though the algorithm of the node closure of section 3.1 is not so efficient.

Let us now show how to improve the algorithm version 1. Here the major problem is how to find the minimal area for the integration of the fractional weights. We assume that the following boundary criteria on the arc reversed graph for each compound node are reasonable.

(a) Any nearest neighbour node j of the compound node i having the accumulated weight w[i,j] >= 1 is an area boundary node.

(b) Any neighbour node j of the compound node i having only arc weight w[i,j] < 1 and zero outdegree is a trivial area boundary node.

Example. Let the local area of a compound node AB be searched as the following :

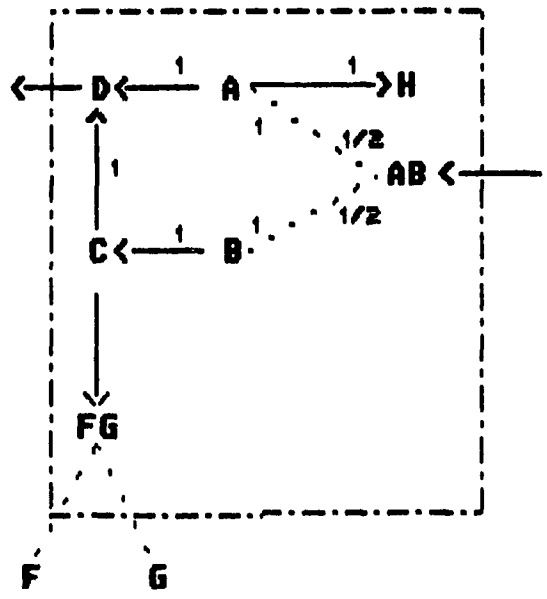

Figure 15

Here H is an end node and it is a trivial boundary, even though w[H,AB] = 1/2.

The node D is a nearest predecessor having the accumulated weight w[D,AB] = 1 and it is a boundary. Among the boundary D, FG, and H, only one node D is accessible fully functionally to the compound node AB.

Now we can clearly present an improved version of the

algorithm.

The algorithm version 2 :

stage 1. Associate with each compound node a minimal local area for integration of fractional weights, i.e. perform the union operations with connection of a minimal number of nodes.

stage 2. Assigning zero weight to all the fractional weight arcs, switch off the already performed union operations in stage 1.

stage 3. Use the Schnorr algorithm [25] to perform the remaining only transitivity operations on the integer weight (w=1) arcs.

The algorithm of stage 1:

```
c[i] : composition list {j|j is a component node of
compound node i};
for all i in V1 do
  {
   mark all j in c[i];
   queue := c[i];   Q := empty;
   while queue NOT = empty do
     {
      remove j from top of queue;
      if j is marked then
       {examined[j] := true;
        /* examine each node at most once */
        for all k in p[j] with w[j,k] = 1 do
         {
```

```
      Q := Q \/ k;
      w[i,k] := w[i,k] + w[i,j];
      if w[i,k] < 1
        AND k is not (marked OR examined) then
      /*the Boolean condition for continuing search*/
           {
            mark k;
            push k into the bottom of queue
           }
           /* the breadth first search manner */
       else
          if w[i,k] = 1 AND k is marked then
          /* the Boolean condition of boundary */
           {
            unmark k;
            unmark all elements in L[k]
           }
    }
}
 for all k in Q do
    {
     if w[i,k]<1 then w[i,k]:=0;
     marked[i]:= False;
     examined[i] := False;
    }
 for all k in c[i] do
    {
```

```
        marked[k]:= False;
        examined[k] := False;
      }

  }
```

The algorithm of stage 1 is to find the nearest predecessors on each possible path fully functionally connected to each compound node, i.e. find the restricted local list of predecessors for each compound node. Such list is searched in the breadth first manner. For such a process, if it traverses no more than (n/2 + 1) nodes, then it satisfies the criteria of the Schnorr algorithm of construction of predecessor lists. Therefore, the complexity of searching the restricted predecessors for |V1| compound nodes is o(n + m+).

If an FD-graph contains a set of random distributed compound nodes, then the local area of each compound node may be less than (n/2 + 1) nodes. Our assumed boundary criteria are too strong and these criteria split an FD-graph into |V1| small local areas not completely overlapped. The complexity is o(n + m+) either for stage 1, or stage 2 and 3. If compound nodes are positioned at nearly end of their FD-graph, then the search of a local area might be extended to the whole graph and the complexity becomes o(|V1||E|). It happens as an exceptional case and it serves as the upper bound of the complexity. Its probability is expected to be very low.

In fact, the Schnorr algorithm was established for the random graphs [25] with the average time complexity of O(n + m+). We have shown that a similar algorithm may be presented for the computation of the FD-closure. Its average time complexity becomes O(n + m+) if all the compound nodes are randomly distributed on their FD-graph.

CHAPTER 4

FD-GRAPH MANIPULATION AND DATABASE SCHEMA DESIGN

As described in the previous chapters, we use weighted directed graphs to represent functional dependencies. Such a graph is denoted as an FD-graph. Manipulating on such an FD-graph, we intend to obtain a minimal set of relation schemes by searaching desired components on an FD-graph by a series of processing.

In Codd's decomposition approach [14], a decomposition is to start with a relation scheme. Decomposing a relation scheme means repeatedly breaking the relation scheme into a pair of relation schemes such that the decomposed relation schemes satisfy the functional dependencies losslessly and each relation scheme is in third normal form (3NF).

Our graph method may be classified as Bernstein's synthesis approach [4,12]. However, it will be shown that our graph method is much simple than the synthesis method in [4,12].

The synthesis approach starts with finding a minimum cover with respect to the given set of FD's F. As stated

in [4,5,12], it is necessary to find a nonredundant cover and eliminate extraneous attributes from the left and right side of each functional dependency before a minimum cover can be obtained.

A nonredundant cover with respect to a set of FD's F is defined such that if there is no set of FD's F' properly contained in F such that (F')+ = F+. In fact, the process of finding a nonredundant cover is also a process of minimizing the number of functional dependencies in F.

A set of FD's F in L-minimum is defined such that F is minimum and for every FD X->Y in F, there is no X' properly contained in X with X'->Y in F+. R-minimum is defined similarly for the right side of each functional dependency.

Before we give the details, some definitions are necessary.

Let U = {A, B, C, ... } be a finite set of attributes. For some X in U and a set of FD's Fi = {X1->Y1, ..., Xk->Yk} such that the union of (Xi \/ Yi) over i is in X, then <X, X1->Y1, ..., Xk->Yk> represents a relation scheme. A database schema is a finite set of relation schemes D = {<X1,F1>, ..., <Xm,Fm>}. An attribute is prime iff (if and only if) A is an element of any key of a relation scheme.

A database schema design problem may be stated as the following :

Given a finite set of attributes U and a set of FD's F, find a database schema D = {<X1,F1>, ... <Xm,Fm>} such that

- X1 X2 ... Xm = U,
- F is completely embodied in D that is F = { Ki->Xi | Xi in U and Ki is a key of Xi},
- the number of relation schmes |D| is minimal,
- every relation scheme <Xi,Fi> is in 3NF with respect to F.

We leave the lossless join property with respect to <U,F> unspecified for the moment.

Using weighted directed graphs, a finite set of attributes U is represented by a set of nodes V0, and a set of FD's F is represented by a set of arcs E and a set of compound nodes V1. Thus, a given <U,F> is mapped onto an FD-graph G=(V,E). Such mapping correspondence always exists, we need not mention each time in the next chapters.

Definition. The sets of attributes X and Y are equivalent under a set of FD's F, written as X<-->Y, if X->Y and Y->X are in F+.

Example. An FD-graph for two sets of equivalent nodes AB<-->CD is represented as in Figure 16. There is no arc inside the set {A,B} or inside the set {C,D} on the graph of Figure 16. It is easy to derive the full arc linking the compound nodes AB and CD. The equivalency of AB and CD does not mean that there exist the equivalencies of

their component nodes, such as A<-->C and B<-->D. Otherwise, there is a dotted arc linking the equivalent compound nodes AB and CD as shown in Figure 17.

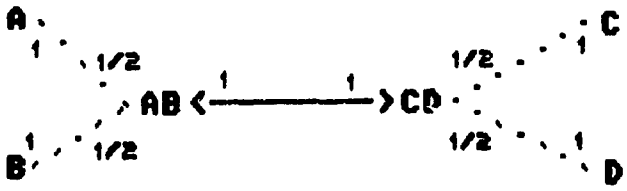

Figure 16

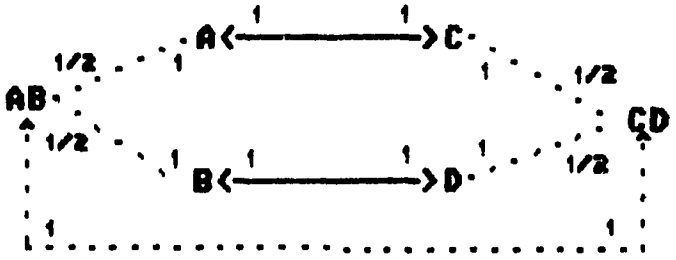

Figure 17.

4.1 FD-graph manipulation

Let us manipulate weighted FD-graphs for the following problems in the database schema design :

1. Mapping a given <U,F> onto an FD-graph.

2. Finding a nonredundant covering with respect to a given set of FD's F.

3. Finding equivalent nodes for merging the related equivalent keys.

4. Elimination of extraneous attributes on the left and right side of functional dependencies (LR-minimum covering problem).

5. Elimination of redundant arcs with respect to F.

To achieve a nonredundant covering, it must eliminate redundant nodes and redundant arcs which represent redundant functional dependencies with respect to F. A single node representing a logical attribute may not be redundant because a single node itself cannot map a functional dependency. However, a compound node having only the dotted arcs to its component nodes is a redundant node. Such compound node together with its dotted arcs represents only the reflexivity rule. The reflexivity is independent of any given set of functional dependencies. It must be redundant.

A compound node having only dotted arcs not only to its component nodes is also redundant. For example, in the following FD-graph :

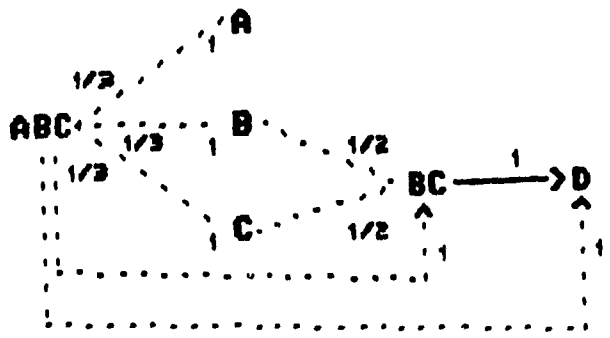

Figure 18.

the dotted arc in ABC->BC represents the reflexivity rule and the dotted arc from node ABC to node D is redundant. Thus, the compound node ABC together with all its dotted

arcs are redundant. Furthermore, if all the full arcs leaving a compound node can be replaced by their corresponding dotted arcs, then the compound node is also redundant. For example, in the following FD-graph :

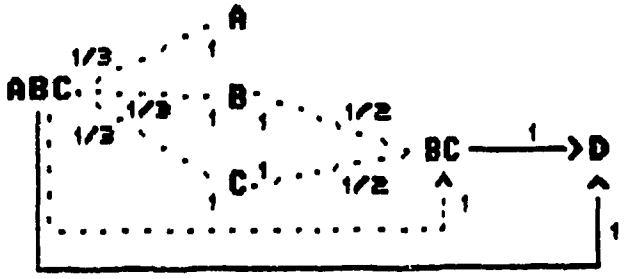

Figure 19.

the full arc ABC->D can be replaced by a dotted arc ABC...>D The compound node together with all its leaving arcs are redundant. Therefore, a compound node having only dotted arcs or all its leaving full arcs which may be replaced by the dotted arcs is a redundant node. Elimination of redundant compound nodes and redundant arcs produces nonredundant covering. Elimination of redundant arcs is an opposite side of the FD-graph closure problem. We shall solve such problem in the next section.

For solving L-minimum covering, we need to examine for every FD X->Y in F, is there X' properly contained in X with X'->Y. Thus, we need to check each compound node X together with its component nodes, i.e. is there (X - X') ->Y redundant. For example,

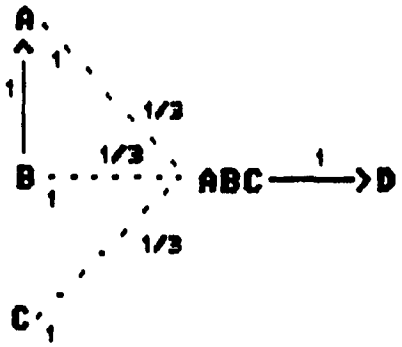

Figure 20.

the arc from node ABC to node A is redundant. Thus, elimination of extraneous attributes on the left side of a functional dependency leads to elimination of redundant dotted arcs or full arcs. Deletion of the dotted arc ABC->A accompanies the update of its composition list c[ABC] = ABC into c[BC] = BC. Then the arc weight equal to 1/|c| is automatically readjusted from |c[ABC]| to |c[BC]|. Similarly, the examination of R-minimum covering will lead to elimination of redundant arcs from a prime node to the related nonprime nodes. As stated above that elimination of redundant compound nodes is not a difficult problem. The major problem in manipulation of weighted FD-graphs is related to the problem of elimination of redundant full arcs and redundant dotted arcs. It will involve the problem of redundant compound nodes and extraneous attributes on the left side and right side of functional dependencies.

We shall show that a subset of equivalent nodes create a cycle on its FD-graph. Thus, there exist the

problems of elimination of redundant arcs on acyclic graphs and on cyclic graphs.

One of the advantages of the weighted graph concept is that it can simplify the problem of redundant arcs of FD-graphs into the transitive reduction problem because the three Armstrong axioms A1, A2 and A3 are implied from the weighted transitivity rule.

4.2 Elimination of redundant arcs in acyclic FD-graphs

According to Ullman [20] the time complexity of the best algorithm for finding the transitive reduction of a graph is the same as the time to compute the transitive closure. We shall show how the problem of elimination of redundant arcs in FD-graphs is similar with the classical transitive reduction problem when the weighted concept is used. We also show how to reduce the usual time complexity of o(|V| **3) into o(|V'||E|) or into o(|V| + |E|) for the algorithms of the above problem. In case of cyclic FD-graph, it needs to find strong connected components of equivalent nodes [22]. After the processing strong connected components, the final algorithms are similar as with the acyclic case.

The graph algorithms were published mostly for non-multigraphs. These algorithms were developed for the classical transitive closure [7,9], the transitive reduction [20] and the minimum equivalent graphs [21,22]. The Armstrong axioms may not be applied to these problems.

We have to use the concept of weighted arcs to imply the union rule of the Armstrong axioms. Schnorr [25] derived a linear transitive closure algorithm. Moyles, Thompson [21] and Hsu [22] showed that the time complexity for the algorithm of finding a minimum equivalent digraph is o(|V| ** 3). Obviously, this time complexity can be reduced to o(|V'||E|) using adjacency lists where |V'| is the number of nodes with outdegree greater than zero, and |E| is the number of arcs.

In order to develop an algorithm for reduction of redundant arcs, the path set p[i,j] are needed. To avoid having |V|**2 matrix elements, i and j are not necessarily over the full range of |V|. The path set is initialized by the arc weights.

Let w[i,k] be the weight of an arc (i,k). If p[i,k] = 1 and p[i,j] * p[j,k] = w[i,k], then the arc (i,k) is redundant and it can be deleted or assigning it a zero weight.

The algorithm for reduction of redundant arcs :

```
L0[i], L1[i] : input set {j|j in V and (i,j) in E0, E1};
w0[i,j], w1[i,j] : input and output arc weights;
s[i] : working set;
p[i,j] : path weight;
Initialize p[i,j] := w[i,j] for all arcs;
#If p[i.j] not defined
 initialize p[i,j] := 0;
#Endif;
```

```
/* only those nodes with outdegree > 0 may have transitive
arcs */
for all i  in V with outdegree > 0 do
  {
  /*use stack for depth first search of transitive arcs*/
   stack := s[i] := {j|(i,j) in E and |LO[j]\/L1[j]|>0};
   while stack NOT = empty do
     {
      remove j from stack;
      /* examine each node no more than once */
      if j not marked then
        {
         mark j;
         push j into queue;
         if p0[i,j] = 1 OR p1[i,j] =1 then
            {
             for all k in L0[j]\/L1[j] do
               {
                if w0[i,k]>0 OR w1[i,k]>0 then
                /*if path p exists, delete the redundant*/
                   if p1[i,k] = 1 then
                      w1[i,k]:=w1[i,k]-(w0[j,k]\/w1[j,k]);
                   else
                    if p1[i,k]=1/|c[k]|/\ i,j in c[k] then
                       {
                        c[k] := c[k] - j;
                        w0[k,j] := 0;
```

```
                    w1[j,k] := 0;
                  }
               if p0[i,k] = 1 then
                  w0[i,k]:=w0[i,k]-(w0[j,k]\/w1[j,k]);
               if |L0[k]\/L1[k]|>0 AND (k not marked)
                  then push k onto stack;
            }
         }
for all j in queue do
  unmark j;
END.
```

It is not difficult to show that an arc with a fractional weight is meaningful if it is involved with a compound node and its component nodes. For example, suppose that the composition list of the compound node k contains i and j. When there is an arc from node i to node j, then the dotted arc (k,j) with the weight w[k,j] = 1 and the dotted arc (j,k) with the weight w[j,k] = 1/|c[k]| become redundant. Deletion of j from c[k] will update the arc weight automatically. Since this algorithm processes only nodes with outdegree greater than zero, and traverse each node no more than once. It guarantees that only the nonredundant arcs or their equivalent paths are traversed but not the both. In comparison with the Hsu algorithm, we reduce the time complexity from o(|V|**3) into o(|V'||E|) where V' stands for nodes with outdegree greater than zero, and E for nonredundant arcs or their

equivalent paths. Thus, we reached the time complexity as it for the FD-closure. If we simulate the Schnorr transitive closure algorithm [24] here, then the time complexity for reduction of redundant arcs may be further reduced.

CHAPTER 5

AN ALGORITHM FOR THE LOSSLESS JOIN PROPERTY

The graph algorithms of the above chapters may not produce a database schema having the lossless join property. To this goal it needs to find a kernel of the FD-graph closure. Such a problem is NP-hard [10,11].

A kernel of an FD-graph closure G=(V,E+) is a subset V' in V such that no two nodes in V' are joined by an arc in E+ and such that for every node in (V - V') there is a node in V' for which (A,B) in E+.

The semantic constraints of FD's F on a given universal relation scheme <U,F> require that the database schema { <X1,F1>, <X2,F2>, ..., <Xn,Fn> } has the lossless join property. To check whether the database schema produced by our graph algorithms has the lossless join property, we may use the algorithm of Aho, Beeri and Ullman [23]

The algorithm Test [24]:

Input : a universal relation scheme <U,F> where U = {A1, A2, ..., Ak}; a database schema D = { <X1,F1>, ..., <Xn,Fn> }.

Other expressions :

d[i,j] array with 1<= i <= n, 1<= j <= k;

a, b1, ..., bn constants, totally ordered by < as follows a < b1 < .. < bn.

Step 1 [define "characteristic matrix" of D];

define d[i,j] = a if Aj in Xi,

= bi otherwise.

Step 2 [apply functional dependencies of F]

while array d[i,j] can be modified do :

if R->S in F, i1 not = i2 and d[i1,j] = d[i2,j] for all j such that Aj in R. Then for all j such that Aj in S, set d[i,j] := d[i2,j] := minimum (d[i1,j], d[i2,j]).

Step 3 [test for the lossless join property]

If there is an i0 such that d[i0,j] = a for all 1<= j <=k, then output " D has the lossless join property with respect to <U,F> ". Otherwise, output " D does not have the lossless join property with respect to <U,F> ".

Biscup, Daual and Bernstein [24] proved that if a database schema D has the lossless join property with respect to <U,F>, then there is a component <Xi0, Fi0> in D such that Xi0 -> U in F+, otherwise determine any key Y of <U,F> and output D := D \/ {<Y,0>} where 0 is an empty functional dependency.

It is obviously that the key Y defined by Biscup et al [24] may not be the kernel of its FD-graph because it does not require that the subset U-Y has empty functional dependency. Otherwise, if there is an empty functional

dependency, then <U,F> becomes a relation scheme satisfying 3NF and the lossless join property, and no more computations are needed

Finding a key Y of <U,F> and satisfying <Y,O> means that the key Y of the universal relation scheme has empty functional dependency such that the formed relation scheme <Y,O> satifies 3NF. Obviously, it is not an easy problem. Biscup et al [24] did not provide any algorithm for a such problem. However, it has been shown that if there is a component <XiO, FiO> in D such that XiO -> U with respect to F+, then the database schema D has the lossless join property with respect to <U,F>. Following this idea, we may construct such a component <XiO, FiO> if it does not exist in the database schema D. To gain this goal, we start from a keys union Ku1 constructed from the keys of all the relation schemes in database schema D having not the lossless join property.

Ku1 = K1 \/ K2 ... \/ Km

where K1, K2, ..., Km are the keys in the database schema D = { <K1->X1,F1>, <K2->X2,F2>, ..., <Km->Xm,Fm> }. It is obviously that

closure (Ku1) = U

but the scheme <Ku1,Fu1> constructed from the keys union Ku1 and the subset of FD's Fu1 of F among the attributes of Ku1 may not satisfy the third normal form conditions and Ku1 is not a real key of the scheme. For a such purpose we may map the scheme <Ku1, Fu1> onto an FD-graph

G=(Vu1,Eu1) with the node set Vu1 and the arc set Eu1. Then we can apply the same algorithms as described in the chapter 2-4 to search each 3NF component from G=(Vu1,Eu1). Thereafter, we construct a new keys union Ku2 from the new database schema formed by G=(Vu1,Eu1). We may continue such procedures repeatedly until a scheme <Kum,O> having empty functional dependecy is obtained. Then Kum becomes a real key of U with respect to F+.

Theorem. The scheme <Kum,O> constructed by the abovementioned procedures will satisfy the following properties :

- Kum -> U with respect to F+,
- <Kum,O> is in 3NF,
- |Kum| reaches a minimum.

Proof. Since we use the weightd FD-graph algorithms, the keys union is repeatedly constructed from the keys on the FD-graphs in the normalization processes. Then it has the sequence Ku1 Ku2 ... Kum and the sequence Fu1 Fu2 ... Fum satisfying dependencies Kum -> Ku(m-1), ..., Ku2 -> Ku1 -> U under Fum Fu(m-1) Fu2 Fu1 in F+. Hence, we obtain that Kum -> U with respect to F+ and <Kum,O> is in 3NF because Fum is empty. Since Fum is empty, ther is no further processing can be done for extracting a subset from Kum. Thus, |Kum| reaches a minimum.

Example. Let <U,F> = { <ABCDE, A->B, B->C, C->D> and its output D = { <AB, A->B>, <BC, B->C>, <CD, C->D>, <E, O> }. It may test that D does not have the lossless join

property. Let us use the above described procedures, we obtain :

Ku1 = A\/B\/C\/E Fu1 = {A->B, B->C}

Ku2 = A\/B\/E Fu2 = {A->B}

Ku3 = A\/E Fu3 = empty

Insert the new component <Ku3,O> = <AE,O> into D. then D := D \/ <AE,O> has the lossless join property.

We shall show that the time complexity of the above computation is not nonpolynomial.

Theorem. The time complexity of the computation of a third normal form relation scheme <Kum, Fum> with Kum ->U in F+ is not nonpolynomial.

Proof. Suppose that the first keys union Ku1 consists of n nodes. If the relation scheme is not in 3NF, then decompose <Ku1, Fu1> into several 3NF relation schemes and construct a new keys union Ku2. |Ku2| must be <= |Ku1|, otherwise there is no change. The final relation scheme contains at least one node. Thus, the computations are at most o(n) steps. At each step we use the graph algorithms of the above chapters. They are polynomial, consequently, the total time complexity is polynomial.

Note that the obtained <Kum,O> satisfies the minimality criteria because at each step the minimum cover is used.

Now we can show that the relation scheme <Kum,O> is a subset of Ku1 = {K1, K2, ..., Km} :

<Kum,O> = {Ki | Ki in Ku1 and indegree(Ki) = 0}

where indegree(Ki) is the number of directed arcs incident

into the node Ki. In the above example of this chapter, the nodes A and E do not have any incident directed arc, thus <Kum,O> = {A, E}. Here we can see that the solution of the lossless join property is so much simplified due to using the FD-graph representation.

CHAPTER 6

CONCLUSIONS

We show in this report that the weighted FD-graph method is a very useful concept. It simplifies a complete and independent set of the inference rules for functional dependencies into one rule on weighted FD-graphs. This is derived from the fact that each arc in a weighted FD-graph carries a specified information via its weight. In case of the set of the reflexivity, transitivity and union rules, the weighted transitivity rule may imply all the three rules on weighted FD-graphs because the additivity of the weights implicitly represents the union rule, the reflexivity is independent of functional dependencies and it is implied by the related compound nodes.

As Ullman [20] indicated that the transitive reduction and the transitive closure may have the same time complexity. Thus, it is possible further to reduce the complexity of nonredundant cover in FD-graphs because the transitive closure, the transitive reduction and the nonredundant cover are mutually related if the weighted graph concept is introduced. Particularly, the FD-graph

closure as an example was solved in this report with a linear average time complexity.

In this report we also suggested the algorithms to obtain the lossless join property for design of database schema D. The union of all keys in relation schemes of a databse schema D is a good starting node set. Force the keys union K to satisfy the third normal form conditions, the universal closure condition K -> U with respect to F+ and the minimality of |K|. Repeat such a process, the final keys union Kum will have empty functional dependency O. Then D := D \/ <Kum, O> will have the lossless join property. Therefore, the solution of the conventional NP-hard kernel problem of FD-graph closure can be avoided.

References

1. Fagin, R., Mendelzen, A. O., and Ullman, J. D.,
A Simplified Universal Relation Assumption and its Properties. ACM Trans. Database Syst. 7 (1982), 343-360.

2. Batini, C. and D'Atri, A.,
Schema Hypergraphs : A Formalizm to Investigate Logical Data Base Design. In Proc. on Graph Theoretic Concept in Computer Science, Lecture Notes in Computer Science 100, Springer-Verlag, New York, 1981, pp. 177-194.

3. Ausiello, G., D'Atri, A. and Moscarini, M.,
Minimal Coverings of Acyclic Database Schemata. In Adv. Data Base Theory, Vol. 2, Plenum Press, New York, 1984, pp. 27-52.

4. Beeri, C. and Bernstein, P. A.,
Computational problems related to the design of normal form relational schemas. ACM Trans. Database Syst. 4 (1979), 30-59.

5. Maier, D.,
Minimum covers in the relational database model. J. ACM 27 (1980), 664-674.

6. Armstrong, W.,
Dependency structure of database relationships. Proc. IFIP 74, North-Holland, Amsterdam, 1974, pp. 580-583.

7. Zaniolo, C. and Melanoff, M. A.,
A formal approach to the definition and the design of computational schemata for database systems. ACM Trans Database Syst. 7 (1982), 24-59.

8. Ausiello, G., D'Atri, A. and Sacca, D.,
Graph algorithms for functional dependency manipulation. J. ACM, 30 (1983), 752-766.

9. Ausiello, G., D'Atri, A. and Sacca, D.,
Graph algorithms for the synthesis and manipulation on data base schemes. In Proc. Graph Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 100, Springer_Verlag, New York, 1981, pp. 212-233.

10. Lewis, E. A., Sekino, L. C. and Ting, P. D.,
A canonical representation for the relational schema and logical data independence. IEEE COMPSAC '77 Conference, Chicago, Ill, November 1977, pp. 276-280.

11. Garey, M. R. and Johnson, D. S.,
Computer and Intractability, Freeman, San Francisco, 1979,

p. 204.

12. Bernstein, P. A.,
Synthesizing third normal form ralations from functional dependencies, ACM Trans. Database Syst. 1 (1976), 277-298.

13. Beeri, C., Bernstein, P. A. and Goodman, M.,
A sophisticate introduction to database normalization theory. In Proc. 4th Int. Conf. on Very Large Data Base (Berlin, Oct. 1978), ACM, New York, pp. 113-124.

14. Codd, E. F.,
Further normalization of the data base relational model. In Data Base Systems (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, N. J., 1972, pp. 33-64.

15. Delobel, C. and Leonard, M.,
The decomposition process in a relational model. Int. Workshop on Data Structure, IRIA, Namur (Belgium), May 1974.

16. Delobel, C. and Casey, R. C.,
Decomposition of a data base and theory of Boolean switching functions. IBM J. of Res. and Dev., 17:5 (Sept. 1972), 370-386.

17. Fagin, R.,
Functional dependencies in a relational database and propositional logic. IBM J. of Res. and Dev., 21:6 (Nov. 1977), 534-544.

18. Zaniolo, C.,
Analysis and design of relational schemata for database systems. Tech. Rep. UCLA-ENG-7769, Dept. of Computer Science, UCLA, July 1976.

19. Warshall, S.,
A theory on Boolean matrices. J. ACM, 9 (1962), 11-12.

20. Aho, A. V., Garey, M. R. and Ullman, J. D.,
The transitive reduction of a directed graph. SIAM J. Comput. 1 (1972), 131-137.

21. Moyles, D. M. and Thompson, G. L.,
An algorithm for finding a minimal equivalent graph of a digraph. J. ACM, 16 (1969), 455-460.

22. Hsu, H. T.,
An algorithm for finding a minimal equivalent graph of a digraph. J. ACM, 22 (1975), 11-16.

23. Aho, A. V., Beeri, C. and Ullman, J. D.,
The theory of joins in relational data base. Proc. 18th

IEEE symp. on Fundations of Computer Science, Oct. 1977, pp. 107-113.

24. Biscup, J., Dayal, M. and Bernstein, P. A., Synthesizing independent database schemas. In ACM SIGMOD 1979 Int. Conf. on Management of Data (Boston, Mass., May 30 - June 1), ACM, New York, pp. 143-152.

25. Schnorr, C. P., An algorithm for transitive closure witn linear expected time. SIAM J. Comput. 7 (1979), 127-132.