OPTIMIZATION OF SYSTEMS OF ALGEBRAIC EQUATIONS FOR EVALUATING DATALOG QUERIES

S. Ceri (',"), L. Tanca (",^)

(') Dipartimento di Matematica, Universita' di Modena, Italy
(") Dipartimento di Elettronica, Politecnico di Milano, Italy
(^) Dipartimento di Matematica e Applicazioni, Universita' di Napoli, Italy

ABSTRACT

A Datalog program can be translated into a system of fixpoint equations of relational algebra; this paper studies how such a system can be solved and optimized for a particular query. The paper presents a structured approach to optimization, by identifying several optimization steps and by studying solution methods for each step.

1. INTRODUCTION

The optimization of Datalog programs is gaining increasing interest in several recent papers [Ban 86a, Ban 86b, HeN 84, KiL 86, Gar 86]. In particular, one approach to optimization is based on the equivalence between Datalog programs and fixpoint equations of relational algebra [AhU 79, DeA 86, CGL 86, KiL 86]. In [CGL 86] it has been shown how a syntax-directed translation can be applied to Datalog clauses to generate corresponding algebraic equations. Such equations have the following features:

a. They are of the general form $X_i=E_i(X_1..X_n,C_1..C_m)$, where X_i are relational variables and C_j are constant relations from an extensional database EDB. We denote such equations as "simple" (because the left hand side is a relational variable) and "recursive" (because X_i can occur within E_i).

b. Expressions E_i contain the following relational operations: selection (σ), projection (π), cartesian product(x), union(U). These operations correspond to positive relational algebra (RA+); all expressions in RA+ are monotone.

Datalog queries are, without loss of generality, translated into selections over one relational variable X_i; the evaluation of a query corresponds to producing all the tuples of X_i that satisfy the selection condition and can be deduced from the Datalog program and the EDB. The optimization of a query evaluation consists of determining an efficient strategy for evaluating those tuples; efficiency is measured in terms of the required interaction with the underlying EDB.

In this paper, we present a structured approach to the optimization of queries for a given system of simple, recursive, algebraic equations. In Section 2 we introduce our terminology; in Section 3 we present our approach as a set of independent optimization steps; in Section 4 we compare our approach to previous work.

2. TERMINOLOGY

A system S is a set of n equations $X_i = E_i(X_1..X_n,C_1..C_m)$; X_i are variables, C_j are constants. Let X_S denote the set of all variables of S. A query on S is a selection $\sigma_p X_p$; we denote X_p, the variable involved in the query, as principal variable of the system. We restrict the query predicate p to be the conjunction of simple predicates (i.e., column equal value), with at most one simple predicate for each column of X_p; this is the algebraic translation of Datalog queries, where some of the places of the query clause are bound to constant values.

The syntax-directed translation from Datalog clauses to fixpoint algebraic equations [CGL

86] produces equations of the following form:

$X_i = E_i(X_1..X_n, C_1..C_m) = U_{j=1..n_i} T_{ij}$

with $T_{ij} = \pi_{L_{ij}} \sigma_{P_{ij}} CP_{ij}$

L_{ij} is a list of column numbers; P_{ij} is the conjunction of simple predicates corresponding to selection and join conditions; each CP_{ij} is a cartesian product involving variables or constants.

We associate to each system S a directed dependency graph G(S)=<N,E> [CGL 86], defined as follows:

- $N = X_S$
- $E = \{<X_i,X_j> \Leftrightarrow X_j \text{ occurs in } E_i\}$

In many examples, we will make use for brevity of the composition operation, obtained by projecting out the join columns from the join of two relations. For two binary relations R and S:

$$R \text{ o } S = \pi_{1,4}\ R \bowtie_{2=1} S$$

Example 1. Throughout this paper we consider the system:

$$X_1 = C_1 \text{ U } (X_1 \text{ o } X_3) \text{ U } X_2$$
$$X_2 = (X_1 \text{ o } X_3) \text{ U } C_3$$
$$X_3 = (X_3 \text{ o } C_2) \text{ U } C_4$$

This system is obtained as the syntax-directed translation [CGL 86] of the Datalog program:

```
X1(Y,Z) :- C1(Y,Z).
X1(Y,Z) :- X1(Y,T),X3(T,Z).
X1(Y,Z) :- X2(Y,Z).
X2(Y,Z) :- X1(Y,T),X3(T,Z).
X2(Y,Z) :- C3(Y,Z).
X3(Y,Z) :- X3(Y,T),C2(T,Z).
X3(Y,Z) :- C4(Y,Z).
```

We will always assume binary relations for our examples.

3. STRUCTURED APPROACH TO SYSTEM OPTIMIZATION

In this section we propose a structured approach to system optimization, based on several progressive optimization steps.

3.1 Reduction To Union-Join Normal Form

Let $n_i > 1$ in E_i; then, for each subterm T_{ij} we introduce a new variable N_{ij} and we rewrite the equation E_i as follows:

$X_i = U_{j=1..n_i} N_{ij}$

$N_{ij} = T_{ij} = \pi_{L_{ij}} \sigma_{P_{ij}} CP_{ij}, \forall j=1..n_i$

Thus, we reduce our equations to either of the two following types:

a. Union (U) Equations, which are fully characterized by a tuple: $<X_i,U_i>$, where X_i is the left side variable, U_i is the set of variables which appear in the union.

b. Join (J) Equations, which are fully characterized by a quadruple: $<X_i,L_i,P_i,J_i>$, where X_i is the left side variable, L_i is a projection list, P_i is a predicate, and J_i is the set of variables and constants appearing in the right side cartesian product.

The resulting system S is in Union-Join Normal Form (UJNF).

Example 2. The UJNF of the system of Example 1 is:

$$X_1 = C_1 \text{ U } N_1 \text{ U } X_2$$
$$X_2 = N_1 \text{ U } C_3$$
$$X_3 = N_2 \text{ U } C_4$$
$$N_1 = X_1 \text{ o } X_3$$
$$N_2 = X_3 \text{ o } C_2$$

The corresponding dependency graph is shown in Fig. 1.

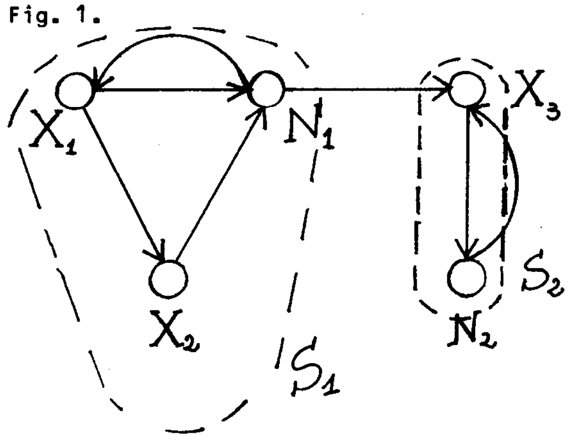

Fig.1

3.2. Determination Of Common Subexpressions

Determining common subexpressions allows reducing the computation by factoring operations [Fin 82]; common subexpressions can be searched on the two sets of U and J equations separately.

a. A common subexpression of two U-equations for X_i and X_j corresponds to any subset U_c of both U_i and U_j. We replace $\{<X_i,U_i>, <X_j,U_j>\}$ with $\{<X_i,U_i-U_c>, <X_j,U_j-U_c>, <X_c,U_c>\}$, where X_c is a new variable. This construction can be iterated until it cannot be further applied; it generates the same final set of U-equations, where equality is intended after a renaming of variables.

b. A common subexpression of two J-equations for X_i and X_j corresponds to any subset J_c of both J_i and J_j; however, we are really interested to isolate common subexpressions if relations of J_c have the same join conditions both in E_i and E_j; this makes the search for common subexpressions of J-equations much more difficult.
Consider two join equations $<X_i,L_i,P_i,J_i>$ and $<X_j,L_j,P_j,J_j>$. Let $J_i=\{J_{i1},..J_{im}\}$ and $J_j=\{J_{j1},..J_{jn}\}$ be such that $J_{ik}=J_{jk}$ for $k \leq r \leq \min(m,n)$; let $P_i=p_{i1}\wedge p_{i2}\wedge..p_{iu}$ and $P_j=p_{j1}\wedge p_{j2}\wedge..p_{jv}$ be such that $p_{ih}=p_{jh}$ for $h \leq s \leq \min(m,n)$. Then, the common subexpression is:

$$\sigma_{p_{i1}\wedge..p_{ih}}(J_{i1}x..xJ_{ik}).$$

We introduce a new equation for X_c and modify equations for X_i and X_j; the construction is rather cumbersome and is omitted.

The problem of finding common subexpressions among n (>2) J-equations is quite complex, and the iteration of this construction does not necessarily generate a unique final set of J-equations.

Example 3. Consider the system S_2:

$$\begin{aligned}X_1 &= C_1 \cup C_2 \cup X_3\\ X_2 &= C_2 \cup X_3 \cup C_3\\ X_3 &= X_1 \circ X_2 \circ X_4\\ X_4 &= C_4 \circ X_1 \circ X_2\end{aligned}$$

After the determination of common expressions, we have:

$$\begin{aligned}X_1 &= C_1 \cup X_5\\ X_2 &= X_5 \cup C_3\\ X_3 &= X_6 \circ X_4\\ X_4 &= C_4 \circ X_6\\ X_5 &= C_2 \cup X_3\\ X_6 &= X_1 \circ X_2\end{aligned}$$

3.3. Query Subsetting

This step depends on the variable upon which the query is applied, but does not depend on the particular query predicate. Consider a query Q over X_p, the principal variable of S. We derive the set $D(X_p) \subseteq X_S$ as follows:

1. X_p belongs to $D(X_p)$.
2. If $<X_i,X_j>$ belongs to E(G(S)) and X_i belongs to $D(X_p)$, then also X_j belongs to $D(X_p)$.

Let $S_p \subseteq S$ be the system of equations for the variables $D(X_p)$; then the query Q can be evaluated on S_p. By construction, the graph $G(S_p)$ is connected. Each strong component of $G(S_p)$ corresponds to a subsystem of mutually recursive equations. Let SC_i and SC_j be two strong components of G(S) connected by some edges from SC_i to SC_j (but not viceversa, otherwise SC_i and SC_j would not be strong components); then, the strong component SC_j should be solved before SC_i. This rule defines a partial order among strong components:

$$SC_j < SC_i <=>$$

$$(<X_i,X_j> \in E(G(S))) \wedge (X_i \in SC_i) \wedge (X_j \in SC_j))$$

By construction, the last strong component according to this partial order includes the variable X_p.

The reason for introducing this ordering in the system resolution is that, after solving SC_j, the variables of SC_j can be considered as constants for SC_i. In particular, it is possible to reduce them in the context of subsystem SC_i with the method that will be defined in Section 3.6.

Example 4. Consider the system of Example 2. If the query is applied on variables X_1, X_2, or N_1, then query subsetting is uneffective (i.e., $S=S_p$). However, the system can be separed into two strongly connected subsystems:

$$\begin{aligned}S1 : X_1 &= C_1 \cup N_1 \cup X_2\\ X_2 &= N_1 \cup C_3\\ N_1 &= X_1 \circ X_3\end{aligned}$$

$$\begin{aligned}S2 : X_3 &= N_2 \cup C_4\\ N_2 &= X_3 \circ C_2\end{aligned}$$

with S2 < S1 (see Fig.1). If the query is applied on X_3 or N_2, query subsetting is effective and the original system S reduces to S2.

3.4. Marking

Marking and the subsequent optimizations depend on the predicate p of the query but do not depend on the particular values used in the selection predicate; in other words, they apply to initial Datalog queries with the same "adornment" [Ull 85].

Markings of relations denote the propagation of the query predicate to the various equations, according to marking rules:

1. Propagation for U-equations: Let X_i be marked $(X_i:m)$, and consider the equation $<X_i, U_i>$; then, give mark m to all variables X and constants C of U_i.
2. Propagation for J-equations: Let X_i be marked $(X_i:m)$, and consider the equation $<X_i, L_i, P_i, J_i>$; using L_i, transform m into the corresponding column(s) n of variable(s) X1 or constant(s) C1 in E_i, and mark them with n (X1:n or C1:n). Further, if that column is involved in equi-joins with column(s) q of different variable(s) X2 or constant(s) C2, then mark them with q (X2:q or C2:q).

Marking rules are motivated by the general equation:

$$\sigma_p X_i = \sigma_p E_i.$$

Their correctness comes from distributivity of selections to unions, commutativity of selections with selections and projections, and distributivity of selections to cartesian products.

The Marking algorithm operates on the system S of n equations and generates a new system S'. Let a_p be the arity of X_p; let the query predicate be an equality predicate over the column i_p of X_p, $1 \le i_p \le a_p$.

a. Initially, mark variable X_p with i_p (denoted $X_p:i_p$).
b. Use recursively the marking rules to mark all possible variables of S; consider the marked variables as new variables of S'.
c. Include recursively in S' all equations $X_i=E_i$ of S such that X_i is mentioned in some of the equations of S' previosly generated.

Example 5.

a. Let $Q1:\sigma_{1=a} X_2$ on the system S = S1 U S2 of Example 4. We obtain:

$$(X_1:1) = (C_1:1) \cup (N_1:1) \cup (X_2:1)$$
$$(X_2:1) = (N_1:1) \cup (C_3:1)$$
$$(N_1:1) = (X_1:1) \circ X_3$$
$$X_3 = N_2 \cup C_4$$
$$N_2 = X_3 \circ C_2$$

i.e., marking is propagated to all variables of S1.

b. Let $Q2:\sigma_{2=a} X_3$ on the system S2 of Ex. 4.

We obtain:

$$(X_3:2) = (N_2:2) \cup (C_4:2)$$
$$(N_2:2) = X_3 \circ (C_2:2)$$
$$X_3 = N_2 \cup C_4$$
$$N_2 = X_3 \circ C_2$$

c. Consider the query $\sigma_{1=a}X1$ on the system:

$$X_1 = X_2 \circ C_1$$
$$X_2 = X_3 \cup C_2$$
$$X_3 = C_1 \circ X_2$$

The marking algorithm yields:

$$(X_1:1) = (X_2:1) \circ C_1$$
$$(X_2:1) = (X_3:1) \cup (C_2:1)$$
$$(X_3:1) = (C_1:1) \circ X_2$$
$$X_2 = X_3 \cup C_2$$
$$X_3 = C_1 \circ X_2$$

3.5. Push of Selection Conditions: Reduced Variables

Let us compare the systems S and S' as obtained after executing the marking. Given that the initial system S is connected, each variable X of S appears at least once in S', either marked or unmarked; it is also possible that S' contains several different markings for the same variable X. In fact, each variable X of S can correspond in S' to either:

1. One unmarked variable, and no marked variable;
2. One or more marked variables, and no unmarked variable;
3. One or more marked variables and one unmarked variable.

We denote those variables for which condition (2) holds as reduced variables.

After executing the marking algorithm, we face two alternatives: either we consider the original system S, or the transformed system S'. Examples 5a and b indicate two extreme cases where it is rather clear how to behave:

a. In Example 5a, |S|=|S'|. Since 3 variables of S' are reduced, S' is more efficient than S. Notice that efficiency comes from the propagation of selections to constant relations appearing in the equations of the reduced variables.

   Example 5a (continued). The system of Example 5a reduces, with a renaming of variables, to:

$$\begin{aligned} V_1 &= (\sigma_{1=a} C_1) \cup V_3 \cup V_2 \\ V_2 &= V_3 \cup (\sigma_{1=a} C_3) \\ V_3 &= V_1 \circ X_3 \\ X_3 &= N_2 \cup C_4 \\ N_2 &= X_3 \circ C_2 \end{aligned}$$

and our query to: $Q=V_2$

b. In Example 5b there are no reduced variables, i.e. all variables of S are also unmarked variables of S'. Hence $S \subseteq S'$, and S is more efficient than S' (there is no advantage in adding equations to S).

However, Example 5c is more critical: X_1 is a reduced variable, but the number of equations of S' is larger than that of S. Here, deciding whether to use S or S' leads to a difficult trade-off. For making this choice, we propose a simple heuristic criterion:

CRITERION: The transformation of a system S into S' produced by the marking algorithm is convenient if there is _at least one_ reduced variable in S'.

The rationale of the above criterion is that, by evaluating system S', we omit computing at least one "large" unmarked variable relation, to the price of computing (possibly) several "small" marked variable relations. Further, we can in general rewrite S and S', by using equivalence transformations of relational algebra, so that the advantage of the transformation of S to S' becomes evident.

3.6. Push of Selection Conditions: Reduced Constants

We now consider constants which are marked in a system S and belong to the equations of unreduced variables. This situation can occur both if selection to variables succeeds or fails. Our aim is to reduce the size of the constant relation _before_ solving the system, by using the information that the constant is marked, i.e. somehow related to the selection condition of the query Q. This reduction, though rather complex to achieve, has a benefit over multiple iterations required by the solution methods of systems. The reduction succeeds in some cases and fails in some other cases; if the reduction succeeds, the constant is said to be _reducible_; else, it is _irreducible_.

Let constant C occur in the equation of variable X_c in S, and let C be marked (possibly by multiple markings) in S'. We initially build the _reduced dependency graph_ G(C), as the subgraph of G(S) which represents all equations involved in the reduction of C. We then give an algorithm for traversing G(C); if the algorithm succeeds, then we build an equation E_C' for a new variable V_C' such that C and possibly other constant relations appear in E_C'. We then show that V_C' can be evaluated independently of S, yielding a result relation C' contained in C (1). Finally, we consider the system S' obtained by substituting C' to C in S and we show that S and S' are equivalent (i.e. they produce the same answer) w.r.t. the query Q.

3.6.1. Definitions

The _C-dependency set_ $X(C) \subseteq X_S$ is built as follows:

a. X_c belongs to X(C).

b. If $<X_i,X_j>$ belongs to E(G(S)) and X_j belongs to X(C), then also X_i belongs to X(C).

The _C-dependency graph_ G(C) is the projection of G(S) over X(C). By construction, the dependency graph includes X_p and all paths from X_p to X_c.

For each variable X_i of X(C), let M_i be the _marking set_ of X_i obtained by collecting all marks of X_i determined by the marking algorithm of Section 3.4. By construction, the marking set of variable X_c is not empty, and at least one mark is propagated to C. After this construction, the remainder of this algorithm uses just the graph G(C).

A _C-dependency_ for the variable X of X(C) is a pair <h,k>, where h is a column of C and k is a column of X. It indicates that the k-th column of X is "influenced" by the h-th column of C, i.e., that some of the values in the k-th column of X are evaluated from some of the values of the h-th column of C.

3.6.2. Reduction algorithm

INPUT: A marked constant C in the equation of

(1) It is also possible that two or more mutually recursive equations define the reduction of two or more constant relations; this happens if E'_{C1} includes the expression of a reduced constant C2 and the expression E'_{C2} for the reduction of C2 includes the expression of the reduced constant C1. This case is covered by solving the system for V'_{C1}, V'_{C2} and obtaining the reduced relations C1' and C2'.

an unreduced variable; the corresponding graph G(C) and C-dependencies.

OUTPUT: Either "C is irreducible" or "C is reducible", with a new equation $V_C'=E_C'$ for evaluating the reduced constant C'.

The reduction algorithm is based on the traversal of the graph G(C), which in turn is based on a basic step. The algorithm requires the use of timestamps associated to the events at which nodes are examined; we assume that timestamps are unique and progressive. A new timestamp is produced by the function "newtime".

3.6.3. Basic step of the traversal algorithm

Let $<X_i,X_j> \in E(G(C))$; consider a traversal operation from X_j to X_i. Let IN_{jt} be a set of dependencies associated to X_j, called <u>incoming dependencies</u>; t is the current timestamp (to be defined later). The <u>basic step</u> consists of rules which dictate how to build the set of C-dependencies OUT_{it}, called <u>outcoming dependencies</u>, and the term A_{ijt}, which contributes to the equation $V_C'=E_C'$, while traversing the edge from X_j to X_i.

The basic step can <u>fail</u>, in which case the entire constant reduction fails, and C is irreducible. Rules of the basic step are as follows:

1. <u>E_i is a U-equation</u>:
   a. $OUT_{it} = IN_{jt}$.
   b. $A_{ijt} = \phi$.
2. <u>E_i is a J-equation</u> (with at least one join operation); set initially $A_{ijt}= \phi$ and $OUT_{it}=\phi$. Several cases are possible:
   a. The traversal <u>fails</u> if there exists one occurrence of X_j in E_i such that, for all $<h,k> \in IN_{jt}$,
      1. The k-th position of X_j is not joined with a constant relation, and
      2. k is not in the marking set of X_j.
   b. The traversal <u>succeeds</u> if for all occurrences of X_j in E_i there exists $<h,k> \in IN_{jt}$ such that either:
      1. The k-th position of X_j is joined with a constant relation, or:
      2. k is in the marking set of X_j.

      The two subcases above are kept separated:

      1. Consider one occurrence X_j in E_i. Let $SJ_{ij}=\{<h_s,k_s> s=1..n_s\} \subseteq IN_{jt}$ be the set of dependencies such that the occurrence of X_j is joined on column k_s with the column w_s of some constant C_s. Then:

         $A_{ijt} := A_{ijt} \cup (..(C \bowtie_{h_1=w_1} C_1) \bowtie .. \bowtie_{h_{n_s}=w_{n_s}} C_{n_s})$.

         OUT_{it} is not modified. We say that the propagation of IN_{jt} is <u>arrested</u>.

         If any of the C_i is a marked constant and is also reducible for a different application of this algorithm, then it is possible to substitute C_i with the variable V_i introduced for reducing C_i to C'_i.
      2. For all occurrences X_j such that the above case (1) does not hold, A_{ijt} is not modified, while OUT_{it} is modified as follows: let $<h,k> \in IN_{jt}$ and w be a column of X_i which corresponds, through the projection list L_i, to the column k of the considered occurrence of X_j in E_i; then, enter <h,w> in OUT_{it}. We say that dependencies of IN_{jt} are <u>passed</u> to OUT_{it}.
3. <u>E_i is a "Projection J-equation"</u> (i.e. a J-equation without join operations but with just one projection):
   a. OUT_{it} is derived as in case 2.b.2.
   b. $A_{ijt} = \phi$.

3.6.4. Traversal algorithm

Let a_C be the arity of C. The <u>initial step</u> of the traversal algorithm is as follows:

a. If E_C is a U-equation, then $OUT_{C0}=\{<1,1>,<2,2>,...<a_C,a_C>\}$.
b. If E_C is a J-equation, set initially $OUT_{C0} = \phi$. Let k be a position of X_C which corresponds, through the projection list L_C, to a position h of C; then, include <k,h> into OUT_{C0}. Iterate this costruction for all positions of X_C.
c. Let $M_C=\{i_1,..i_n\}$ be the marking set of C; initialize E_C' to the expression:

$E_C' = \bigcup_{j=1..n} \sigma_{i_j="c"} C =$

$= \sigma_{i_1="c" \vee .. \vee i_n="c"} C$

where "c" is the query constant.

The dependency graph traversal algorithm is as follows:

1. Perform the initial step; set $D_j = \emptyset$ for all variables X_i of G(C). Assume X_C as the current node.
2. Visit the nodes of the graph in breadth-first order. Let X_j be the current node in the search. The <u>node analysis</u> consists of the following steps:
   a. Generate a new timestamp t: t=newtime.
   b. Evaluate the set IN_{jt} as:

      $$IN_{jt} = \bigcup_{q<u<t} OUT_{ju} - IN_{jq}$$

      where q is the timestamp of the latest traversal operation at which node X_j was analyzed. In practice, IN_{jt} accumulates all dependencies which have been produced for node X_j since its last visit.
   c. Perform the <u>termination test</u>:
      $IN_{jt} \subseteq D_j$.
   d. If the condition is not true, then, set:
      $D_j := D_j \cup IN_{jt}$.

      and for each edge $<X_i, X_j>$ perform the <u>basic step</u> defined in the previous subsection; evaluate terms OUT_{it} and A_{ijt}; accumulate terms A_{ijt} into E_C':

      $E_C' := E_C' \cup A_{ijt}$.
   e. The breadth-first traversal is continued until one of the two following conditions occur:
      1. At all nodes of G(C), the termination condition holds. Then, the algorithm outputs "reducible" and the equation E_C'.
      2. One of the basic steps fails. Then, the algorithm outputs "irreducible".

3.6.5. Termination

The different C-dependencies that can be possibly added to D_j at each node X_j are a finite number. At each traversal, the C-dependencies of IN_{jt} are accumulated in D_j; the traversal takes place iff some C-dependencies of IN_{jt} are not present in D_j, and these are accumulated in D_j. The termination condition imposes that, when all the possible dependencies of IN_{jt} have been included in D_j, node X_j will not be the source of additional traversal operations. This ensures the termination of the algorithm in a finite number of steps.

3.6.6. Application and correctness

Assume that the algorithm terminates successfully. We have now an equation $V_C' = E_C'$, which by construction includes C, other constant relations, possibly V_C' itself, and possibly other variable relations introduced for reducing other constants. We then evaluate the minimal fixpoint C' of the equation $V_C' = E_C'$ (possibly, by solving a system of equations; see Section 3.8). By construction, $C' \subseteq C$. Consider the system S' obtained by substituting C' to C in S; the following, fundamental result holds.

<u>Theorem 1</u>. S and S' are equivalent with respect to the query Q, i.e. they produce the same answer.

The proof of Theorem 1 is omitted, and can be found in [CeT 86].

3.6.7. Computational complexity

The complexity of the method depends on the following factors:

a. Complexity of the basic step.
b. Number of basic steps.

We use the following parameters:

a. N_C = number of columns of C;
b. N_j = number of columns of X_j.
c. N_{ij} = number of occurrences of X_j in E_i.
a. The number of C-dependencies in D_j is $O(N_C * N_j)$; each IN_{jt} or OUT_{jt} is bound by D_j.
b. The complexity of each basic step from X_i to X_j is proportional to the number N_{ij} of different occurrences of X_i in E_i, and to the number of C-dependencies, hence is $O(N_C * N_j * N_{ij})$.

c. The maximum number of traversals is bound by the maximum D_j times the number of nodes in G(C), hence is $O(N_c * N_j * |X_c|)$.

The worst-case complexity of the reduction algorithm is given by the product of complexities (b) and (c).

3.6.8. Queries with multiple selection constants

We can apply the push mechanism for variables and constants to the case in which the selection is done over two positions of the principal variable. The marking and reduction algorithms are in this case applied twice; possibly, the same variables or constants may be reduced twice by effect of two applications of the methods. The final result is independent of the order of application of algorithms.

3.7. Examples

Though the algorithm in Section 3.6 appears very difficult, this is due to the intrinsic difficulty of the problem in its most general formulation; but the algorithm is easily applied to many simple cases which correspond to "reasonable" Datalog programs. This subsection shows the results produced by the algorithm on some examples of progressive difficulty; Examples abc correspond to wellknown problems (ancestor, same generation cousin, unstable same generation cousin; see [Ban 86a]).

Example 6.a.

We now use reduced constants to optimize the system from example 5.b, where no optimization was possible with reduced variables.

Let Q2:$\sigma_{2=a}\ X_3$ on system S2. The marked system is:

$$(X_3:2) = (N_2:2) \cup (C_4:2)$$
$$(N_2:2) = X_3 \circ (C_2:2)$$
$$X_3 = N_2 \cup C_4$$
$$N_2 = X_3 \circ C_2$$

We can reduce constants C_4 ad C_2:

$$V_{C_4} = \sigma_{2=a}\ C_4 \cup C_4 \ltimes_{2=1} V_{C_2}$$
$$V_{C_2} = \sigma_{2=a}\ C_2 \cup C_2 \ltimes_{2=1} V_{C_2}$$

Example 6.b

Consider the system:

$$X_1 = L \circ X_2 \circ R$$
$$X_2 = X_1 \cup C$$

and the query Q: $\sigma_{2=a}\ X_2$, producing the marking:

$$(X_2:2) = (X_1:2) \cup (C:2)$$
$$(X_1:2) = L \circ X_2 \circ (R:2)$$
$$X_1 = L \circ X_2 \circ R$$
$$X_2 = X_1 \cup C$$

The reduction of variables fails, but the reduction of constants is successful, yielding:

$$V_R = \sigma_{2=a}\ R \cup R \ltimes_{2=1} V_R$$
$$V_C = \sigma_{2=a}\ C \cup ((C \ltimes_{2=1} V_R) \ltimes_{1=2} L)$$

The reduction of R corresponds to the "cone" of the Magic Set method ([Ban86]).

Example 6.c

Consider the system:

$$X_1 = L \circ X_2 \circ R$$
$$X_2 = \pi_{21}\ X_3$$
$$X_3 = X_1 \cup C$$

and the query Q: $\sigma_{2=a}\ X_3$, producing the marking:

$$(X_3:2) = (X_1:2) \cup (C:2)$$
$$(X_1:2) = L \circ X_2 \circ (R:2)$$
$$X_1 = L \circ X_2 \circ R$$
$$X_2 = \pi_{21}\ X_3$$
$$X_3 = X_1 \cup C$$

The reduction of variables fails, but the reduction of constants is successful, yielding:

$$V_R = \sigma_{2=a}\ R \cup R \ltimes_{2=2} L$$
$$V_C = \sigma_{2=a}\ C \cup ((C \ltimes_{1=1} V_R) \ltimes_{2=2} L)$$

Example 6.d

Consider:

$$X_1 = N_1 \cup X_2$$
$$X_2 = N_2 \cup K$$
$$N_1 = X_2 \circ R$$
$$N_2 = X_1 \circ C$$

and the query: $Q = \sigma_{2=a} X_1$, producing the marking:

$$(X_1:2) = (N_1:2) \cup (X_2:2)$$
$$(X_2:2) = (N_2:2) \cup (K:2)$$
$$(N_1:2) = X_2 \circ (R:2)$$
$$(N_2:2) = X_1 \circ (C:2)$$
$$X_1 = N_1 \cup X_2$$
$$X_2 = N_2 \cup K$$

$N_1 = X_2 \circ R$
$N_2 = X_1 \circ C$

Here the reduction of variables is not possible either; we obtain the reduced constants:

$$V_K = \sigma_{2=a}K \cup K \bowtie_{2=1} V_R \cup K \bowtie_{2=1} V_C$$

$$V_C = \sigma_{2=a}C \cup C \bowtie_{2=1} V_R \cup C \bowtie_{2=1} V_C$$

$$V_R = \sigma_{2=a}R \cup R \bowtie_{2=1} V_C$$

3.8. Iterative Solution Methods

After going through all transformations of sections 3.1 to 3.7, we obtain a final system S of equations which has to be solved. We can use different iterative approaches; they apply to a vector V of variables X_i, initially all set to ϕ. Termination occurs at iteration f such that $V^f = V^{f+1}$; termination is ensured by the finiteness of the EDB and monotonicity of equations in RA+.

a. The Jacobi method iterates the evaluation of X_i^j, using:

$$X_i^j = E_i(X_1^{j-1} .. X_n^{j-1}).$$

b. The Gauss-Seidel method is similar to the Jacobi Method, however it uses in the course of the evaluation of X_i^j the values already produced for X_k^j, k<i:

$$X_i^j = E_i(X_1^j .. X_{i-1}^j \quad X_i^{j-1} .. X_n^{j-1}).$$

The Gauss-Seidel method has in general better convergence than the Jacobi method.

c. The Chaotic method is typically used for parallel computation and consists in evaluating equations in any order; subcases of chaotic methods are the "lazy" or the "data flow" evaluation, where each variable is evaluated respectively at the latest or at the earliest convenience.

3.9. Efficient Evaluation of Linear Equations.

Efficient algorithms for evaluating single linear equations, reviewed in [CGL 86], can also be applied to systems of equations. We consider the case of one equation E_l which is linear with respect to its own variable X_l; the result is trivially extended to an arbitrary number of linear equations within the same system, and to equations of any fixed degree (along the direction shown in [CGL 86] for a single equation).

Suppose that the equation $X_l = E_l$ is linear with respect to its own variable:

$$E_l(X_1,..,X'_l \cup X''_l ... X_n) =$$
$$E_l(X_1,..,X'_l ... X_n) \cup E_l(X_1,..,X''_l ... X_n)$$

Then the classical Jacobi algorithm:

ALGORITHM A1

```
FOR i:=1 TO n DO Xi=ϕ;
REPEAT
  cond := true;
  FOR i:=1 TO n DO Si:=Xi;
  FOR i:=1 TO n DO
    BEGIN
    Xi:=Ei(S1,.....Sn);
    IF Xi ≠ Si THEN cond := false;
    END;
UNTIL cond;
FOR i:=1 TO n DO OUTPUT(Xi).
```

can be substituted by:

ALGORITHM A2

```
FOR i:=1 TO n DO Xi=ϕ;
Dl=ϕ;

REPEAT
  cond := true;
  FOR i:=1 TO n DO Si:=Xi;
  FOR i:=1 TO n DO
  BEGIN
    IF i=l THEN
      BEGIN
      Dl:=El(S1..,Dl,..Sn)-Sl;
      Xl:=Dl U Sl;
      IF Dl ≠ ϕ THEN cond := false;
      END
    ELSE
      BEGIN
      Xi:=Ei(S1,.....Sn);
      IF Xi ≠ Si THEN cond := false;
      END;
    END;
UNTIL cond;
FOR i:=1 TO n DO OUTPUT(Xi).
```

Theorem 2. If equation $X_l = E_l$ is linear with respect to X_l, then algorithms A1 and A2 are equivalent.

The theorem is proved in [CeT 86].

The advantage of algorithm A2 with respect to A1 is that the term Dl represents just the "difference" tuples evaluated at each iteration, while Sl represent the "accumulation" of all tuples of the previous iterations; given that $|Dl| \ll |Sl|$, the evaluation at each iteration is more efficient.

This final step has completed our structured approach. Prior to the evaluation, we determine common subexpressions and isolate the portion of the system related to the query; then we attempt reducing variables or constants as effect of the propagation of selections; then we order equations according to a partial order between strong components; then we apply a solution method (Jacobi, Gauss-Seidel, Chaotic) to solve the system; we can improve solution methods when the degree of some equations is known.

4. COMPARISON WITH OTHER WORK

This paper is logically the follow-up of [CGL 86]; we borrow from it the syntax-directed translation from Datalog clauses to algebraic equations.

Our push of selections to variables is in fact an extension of the method of Aho and Ullman [AhU 79] for a single equation. In our approach, we push selection conditions to any relational variable, and not just to the "principal variable". Further, in [AhU 79], the optimization was not possible for equations having the variable X_i appearing more than once in E_i; we do not have such restriction.

The "static filtering" method by Kifer and Lozinskii [KiL 86] achieves an analogous simplification as our push of selections to variables. In their method, the systems of equations get translated into a unique equation, which is sometimes cumbersome. Most important, the static filtering method creates a selection predicate which is "looser" than the initial one, and applies it to the principal variable; our method reaches the same result by creating different equations for the same variable marked in different ways. This can be considered an improvement on Kifer and Lozinskii's method if equations are evaluated in parallel; anyway, our form is easily reducible to theirs: whenever S' contains two equations for X_i as follows:

$$(X_i:m)=E_i((X_1:m_1)....(X_n:m_n))$$

$$(X_i:k)=E_i((X_1:k_1)....(X_n:k_n))$$

it is possible to substitute them with:

$$(X_i:m \text{ or } k)=E_i((X_1:m_1 \text{ or } k_1)....(X_n:m_n \text{ or } k_n)).$$

As an example, consider the following non-linear Datalog rules, that are also proposed in [KiL 86]:

```
R(x,y,z):-B(x,y,z).
R(x,y,z):-A(x,u,v),R(u,y,z,),R(v,z,y).
```

with the query Q(x,y) = R(x,y,a). The above rules are translated into:

$$R = B \cup \pi_{1,5,6}((A \bowtie_{2=1} R) \bowtie_{6=2,5=3} R)$$

with the query $Q=\sigma_{3=a}R$. Our push of selections to variables succeeds, producing:

$$(R:2v3)=(B:2v3) \cup \pi_{1,5,6}((A \bowtie_{2=1} (R:2v3)) \bowtie_{6=2,5=3} (R:2v3))$$

which amounts to taking the selection of R over the second or the third column.

It can be noticed that the reduction of variables and constants includes some cases which are also considered by the Magic Sets method of [Ban 86a]. This happens, for instance, in the two cases of linear ancestor query. In fact, in the case of binding of one of the variables of the ANCESTOR relation, the Magic Set method achieves a simplification that is equivalent to the reduction of variables; in case of binding on the other variable, the Magic Set method reaches a reduction of the constant relation PARENT to its relevant part, which is the "cone" of the query constant, in the same way as we do with the reduction of constants (Example 6.a). However, in more difficult cases the comparison between these two methods is not so immediate. It is very important to notice that our algorithms include tests for deciding whether to accept or reject simplifications; while the magic set approach is applied "syntactically", without being able to evaluate on its convenience.

Terms generated by iterative methods for a single fixpoint equation can be efficiently evaluated either by factoring techniques or by parallelization [CoK 86]; these techniques are algebraic in nature and can be conveniently extended to systems of equations.

Courcelle, Kahn and Vuillemin have studied systems of simple recursive equations in the context of the fixpoint semantics of programming languages [Cou 74]. They study equations of the general form X = T, where a term T is recursively defined as either a constant, or a variable, or a function F of a given arity n, applied to n terms Tj. Thus, X=F(X,G(Y,Z)) is a valid equation. They introduce a notion of <u>uniform</u> equation (an equation without nested functions), and they show a construction C for transforming a generic system S into an equivalent uniform system S'. The uniform representation is the basis for deciding whether two systems of equations are equivalent, or whether any two terms are equivalent; both problems are decidable in polynomial time.
We could use the same formalism as in [Cou 74] by interpreting every union or join expression

as a distinct function. The major limitation of such interpretation is that we cannot use the semantics of join or union expressions, in particular for deriving common subexpressions and simplifying terms of equations. Thus, their notion of equivalence is correct but weaker than one that uses all the available knowledge on the meaning of algebraic operations. On the other hand, the latter is more difficult and constitutes currently an open problem.

5. CONCLUSION

In the debate about whether optimization of Datalog programs is better achieved at a high (source programs) or low (algebraic machine) level, this paper supports the latter thesis. In fact, all our algorithms are systematic and can be programmed into an algebraic machine. Our approach makes extensive usage of classical algebraic equivalence properties and of algebraic optimization transformations. Further, our algorithms produce solutions which are subject to quantitative evaluation; each step of our approach includes a trade-off analysis. Thus, we can either accept or reject the simplifications which are proposed at each step.

Ongoing work in this area includes the experimentation of the techniques discussed in this paper through the development of a prototype of the algebraic machine.

ACKNOWLEDGEMENT

We are grateful to Gill Kahn for indicating us previous related work, to Robert Demolombe for a general discussion on systems of equations, and to Georg Gottlob for helping us in the initial development of the method for pushing selections to constants. This work is conducted with the support of the Esprit Project "Meteor", of the italian CNR, and of the MPI 40% Project.

REFERENCES

[AhU 79] Aho, A. and J. Ullman, "Universality of data retrieval languages", Proc. 6th ACM Symposium on Principles of Programming Languages (1979)

[Ban 86a] Bancilhon, F., D.Maier, Y.Sagiv, J.D.Ullman, "Magic sets and other strange ways to implement logic programs", Proc. ACM-PODS, Cambridge (MA) (March 24-26 1986)

[Ban 86b] Bancilhon, F. and R. Ramakrishnan, "An amateur's introduction to recursive query processing strategies", in Proc. of ACM SIGMOD, ed. C.Zaniolo, Washington D.C. USA (May 28-30 1986)

[CeT 86] Ceri, S. and L. Tanca, Optimization of systems of algebraic equations for evaluating Datalog queries, Int. Rep. n.86-034

[CoK 86] Cosmadakis, S. and P. C. Kanellakis, "Parallel evaluation of recursive rule queries", Proc. ACM-PODS, Cambridge (MA) (March 24-26 1986)

[Cou 74] Courcelle, B., G.Kahn and J.Vuillemin, "Algorithmes d'equivalence et de reduction a des expressions minimales, dans une classe d'equations recursives simples", 2nd Int. Coll. on Automata, Languages and Programming, vol. 14, pp. 200-213, Springer, Berlin, Lect. Notes in Comp. Science, Saarbrucken (1974)

[CGL 86] Ceri, S., G. Gottlob and L.Lavazza, "Translation and optimization of logic queries: the algebraic approach", Proc. of 12th Int. Conf. on Very Large Data Bases, Kyoto, Japan (August 25-28 1986)

[DeA 86] Devanbu, P. and R. Agrawal, Moving selections into fixpoint queries, Internal Report (1986)

[Fin 82] Finkelstein, S., "Common expression analysis in database applications", in Proc. of ACM SIGMOD, ed. M. Schkolnick, Orlando, Florida USA (June, 2-4 1982)

[Gar 86] Gardarin, G. and C. de Maindreville, "Evaluation of database recursive logic programs as recurrent function series", in Proc. of ACM SIGMOD, ed. C.Zaniolo, Washington D.C. (May 28-30 1986)

[HeN 84] Henschen, L. and S. Naqvi, "On compiling queries in recursive first order databases", Journal of ACM, vol. 31, pp. 47-85 (January 1984)

[KiL 86] Kifer,M. and E. L. Lozinskii, "Filtering data flow in deductive databases", Proc. of Int. Conf. on Database Theory '86, Roma, Italy (September, 8-10 1986)

[Ull 85] Ullman, J., "Implementation of logical query languages for databases", ACM-TODS, 10:3, pp. 289-321 (1985)