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
   \( X_i = E_i(X_1, \ldots, X_n, C_1, \ldots, C_m) \)
   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 (\( \pi \)), cartesian product (\( \times \)), union (\( \cup \)). 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 \( X_i = E_i(X_1, \ldots, X_n, C_1, \ldots, C_m) \); \( X_i \) are variables, \( C_j \) are constants. Let \( X_0 \) denote the set of all variables of \( S \). A query on \( S \) is a selection \( \pi_X \); 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 \( X_i \); 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

Proceedings of the 13th VLDB Conference, Brighton 1987
863 produces equations of the following form:

\[ x_i = E_i(x_1 \ldots x_n, c_1 \ldots c_m) = U_j=1 \ldots n_i T_{ij} \]

with \( T_{ij} = \prod_{L_{ij}} \bigwedge_{P_{ij}} \bigwedge_{C_{ij}} \)

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

We associate to each system \( S \) a directed dependency graph \( G(S) = (N, E) \) [CGL 863], defined as follows:

- \( N = X_S \)
- \( E = \langle x_i, x_j \rangle \implies x_j \text{ occurs in } E_i \)

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

\[ R \Join S = \prod_{T_i} R \Join_{i=1}^2 S \]

**Example 1.** Throughout this paper we consider the system:

\[
\begin{align*}
  x_1 &= C_1 \cup (x_1 \circ x_3) \cup x_2 \\
  x_2 &= (x_1 \circ x_3) \cup C_3 \\
  x_3 &= (x_3 \circ C_2) \cup C_4
\end{align*}
\]

This system is obtained as the syntax-directed translation [CGL 863] of the Datalog program:

\[
\begin{align*}
  X_1(Y,Z) &:= C_1(Y,Z) \\
  X_1(Y,Z) &:= X_1(Y,T), X_3(T,Z) \\
  X_1(Y,Z) &:= X_2(Y,Z) \\
  X_2(Y,Z) &:= X_1(Y,T), X_3(T,Z) \\
  X_2(Y,Z) &:= C_3(Y,Z) \\
  X_3(Y,Z) &:= X_3(Y,T), C_2(T,Z) \\
  X_3(Y,Z) &:= C_4(Y,Z)
\end{align*}
\]

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:

\[ x_i = U_j=1 \ldots n_i T_{ij} \]

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

a. Union (U) Equations, which are fully characterized by a tuple: \( \langle x_j, U_j \rangle \), where \( x_j \) is the left side variable, \( U_j \) is the set of variables which appear in the union.

b. Join (J) Equations, which are fully characterized by a quadruple: \( \langle x_j, L_j, P_j, J_j \rangle \), where \( x_j \) is the left side variable, \( L_j \) is a projection list, \( P_j \) is a predicate, and \( J_j \) 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:

\[
\begin{align*}
  x_1 &= C_1 \cup N_1 \cup x_2 \\
  x_2 &= N_1 \cup C_2 \\
  x_3 &= N_1 \circ C_4 \\
  N_1 &= x_1 \circ x_3 \\
  N_2 &= x_3 \circ C_2
\end{align*}
\]

The corresponding dependency graph is shown in Fig. 1.

**3.2 Determination Of Common Subexpressions**

Determining common subexpressions allows reducing the computation by factoring operations [Fin 823]; common subexpressions can be searched on the two sets of U and J equations separately.

a. A common subexpression of two U-equations for \( x_j \) and \( x_j \) corresponds to any subset \( U \) of both \( U_j \) and \( U_j \). We replace \( \langle x_j, U_j \rangle \) with \( \langle x_j, U_j \rangle \), \( \langle x_j, U_j \rangle \), \( \langle x_j, U_j \rangle \), \( \langle x_j, U_j \rangle \), where \( x_j \) 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

32 Proceedings of the 13th VLDB Conference, Brighton 1987
variables.

b. A common subexpression of two J-equations for \( X_i \) and \( X_j \) corresponds to any subset \( J \) of both \( J_i \) and \( J_j \); however, we are really interested to isolate common subexpressions if relations of \( J \) 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 \( J_1 = (J_1^1 \ldots J_1^{m_1}) \) and \( J_2 = (J_2^1 \ldots J_2^{m_2}) \) be such that \( J_i = J_j^k \) for \( k \leq r \leq \min(m_1, m_2) \); let \( P_i = P_{i_1} \land P_{i_2} \land \ldots \land P_{i_r} \) and \( P_j = P_{j_1} \land P_{j_2} \land \ldots \land P_{j_r} \) be such that \( P_{ih} = P_{jh} \) for \( h \leq s \leq \min(m_1, m_2) \). Then, the common subexpression is:

\[
\sigma_{P_{i_1} \land \ldots \land P_{i_r}}(J_1^x \ldots J_r^x).
\]

We introduce a new equation for \( X_i \) and \( X_j \); the construction is rather cumbersome and is omitted.

The problem of finding common subexpressions among \( n \geq 2 \) 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 \( S_3 \):

\[
\begin{align*}
X_1 & = C \lor C \lor X_2 \lor X_3 \\
X_2 & = X_4 \land X_5 \land X_3 \\
X_3 & = C_4 \land X_1 \land X_2
\end{align*}
\]

After the determination of common expressions, we have:

\[
\begin{align*}
X_1 & = C \lor X_2 \\
X_2 & = X_4 \land X_5 \\
X_3 & = X_4 \land X_5 \\
X_4 & = C_4 \land X_1 \\
X_6 & = X_1 \land X_2
\end{align*}
\]

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 \( D(X_p) \subset X_3 \) as follows:

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

Let \( S_p \subset S \) 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 \( G(S_p^p) \) is connected. Each strong component of \( G(S_p^p) \) corresponds to a subsystem of mutually recursive equations. Let \( S_{C_i} \) and \( S_{C_j} \) be two strong components of \( G(S) \) connected by some edges from \( S_{C_i} \) to \( S_{C_j} \) (but not vice versa, otherwise \( S_{C_i} \) and \( S_{C_j} \) would not be strong components); then, the strong component \( S_{C_j} \) should be solved before \( S_{C_i} \). This rule defines a partial order among strong components:

\[
S_{C_j} < S_{C_i} \iff \langle X_i, X_j > \in E(G(S)) \land (X_i \in S_{C_j} \land X_j \in S_{C_i})
\]

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 \( S_{C_i} \), the variables of \( S_{C_j} \) can be considered as constants for \( S_{C_i} \). In particular, it is possible to reduce them in the context of subsystem \( S_{C_i} \) 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 \( X_4 \), then query subsetting is ineffective (i.e., \( S = S_p \)). However, the system can be separated into two strongly connected subsystems:

\[
\begin{align*}
S_1 & : X_1 = C \lor N_1 \lor X_2 \\
X_2 & = N_1 \land X_3 \\
N_1 & = X_1 \land X_2 \\
X_3 & = N_2 \land C_3 \\
S_2 & : X_2 = N_2 \land C_4 \\
N_2 & = X_2 \land C_2
\end{align*}
\]

with \( S_2 < S_1 \) (see Fig.1). If the query is applied on \( X_2 \) or \( N_2 \), query subsetting is effective and the original system \( S \) reduces to \( S_2 \).

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 \( g \)-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) \( X_i \) or constant(s) \( C_i \) in \( E_i \), and mark them with \( n \) \((X_i : n \) or \( C_i : n)\). Further, if that column is involved in equi-joins with column(s) \( q \) of different variable(s) \( X_j \) or constant(s) \( C_j \), then mark them with \( q \) \((X_j : q \) or \( C_j : q)\).

Marking rules are motivated by the general equation:

\[ dp X_i = \sum E_i. \]

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 \( a \) be the arity of \( X_p \); let the query predicate be an equality predicate over the column \( i \) of \( X_p \), \( 1 \leq i \leq a \).

a. Initially, mark variable \( X_i \) with \( i \) (denoted \( X_i : i \)).

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 \( S' \) previously generated.

Example 5.

a. Let \( Q_1: d_1 = a X_2 \) on the system \( S = S_1 \cup S_2 \) of Example 4. We obtain:

\[
\begin{align*}
(X_1:1) &= (C_1:1) \cup (N_1:1) \cup (X_2:1) \\
(X_2:1) &= (N_1:1) \cup (C_2:1) \\
(N_2:1) &= (X_1:1) \circ X_3 \\
X_3 &= N_2 \cup C_4 \\
N_2 &= X_3 \circ C_2
\end{align*}
\]

i.e., marking is propagated to all variables of \( S_1 \).

b. Let \( Q_2: d_2 = a X_3 \) on the system \( S_2 \) of Ex. 4. We obtain:

\[
\begin{align*}
(X_3:2) &= (N_2:2) \cup (C_4:2) \\
(N_2:2) &= X_3 \cup (C_2:2) \\
X_3 &= N_2 \cup C_4 \\
N_2 &= X_3 \circ C_2
\end{align*}
\]

c. Consider the query \( \sigma_{i=2} X_1 \) on the system:

\[
\begin{align*}
X_1 &= X_2 \circ C_1 \\
X_2 &= X_3 \cup C_2 \\
X_3 &= C_1 \circ X_2
\end{align*}
\]

The marking algorithm yields:

\[
\begin{align*}
(X_1:1) &= (X_2:1) \cup C_1 \\
(X_2:1) &= (X_3:1) \cup (C_2:1) \\
(X_3:1) &= (C_1:1) \circ X_2 \\
X_2 &= X_3 \cup C_2 \\
X_3 &= C_1 \circ X_2
\end{align*}
\]

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:
\[ V_1 = (d_1 \cap C_1) \cup V_2 \cup V_2 \]
\[ V_2 = V_3 \cup (C_1 \setminus C_2) \]
\[ V_3 = V_1 \cup X_3 \]
\[ X_3 = N_2 \cup C_1 \]
\[ N_2 = X_3 \cap C_2 \]
and our query to: \( Q = V_2 \)

b. In Example 5 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: \( X_i \) 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 use 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 \( X \) 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 \( C \). 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 \( X_i \cap X_j \) 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 \( X \) 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 \( \langle h, k \rangle \), 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 \( E'_C \) includes the expression of a reduced constant \( C_1 \) and the expression \( E'_C \) for the reduction of \( C_2 \) includes the expression of the reduced constant \( C_2 \). This case is covered by solving the system for \( V'_C \) and obtaining the reduced relations \( C_1' \) and \( C_2' \).
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 "newtime".

3.6.3. Basic step of the traversal algorithm

Let <x_i,x_j> ∈ 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 incoming dependencies; t is the current timestamp (to be defined later). The basic step consists of rules which dictate how to build the set of C-dependencies OUT_{jt}, called outgoing dependencies, 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 fail, in which case the entire constant reduction fails, and C is irreducible. Rules of the basic step are as follows:

1. E_i is a U-equation:
   a. OUT_{jt} = IN_{jt}.
   b. A_{ijt} = ∅.

2. E_i is a J-equation (with at least one join operation); set initially A_{ijt} = ∅ and OUT_{jt} = ∅. Several cases are possible:
   a. The traversal fails if there exists one occurrence of X_j in E_i such that, for all <h,k> ∈ 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 succeeds if for all occurrences of X_j in E_i there exists <h,k> ∈ 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 S_{ij} = {<h,k> | s=1...n_s} ⊆ IN_{jt} be the set of dependencies such that the occurrence of X_j is joined on column k, with the column w_s of some constant C_s. Then:
      A_{ijt} := A_{ijt} U (C_x_h_n_i C_s) D <
      .. h_n = w_n C_s
      OUT_{jt} is not modified. We say that the propagation of IN_{jt} is arrested.

   2. For all occurrences X_j such that the above case (1) does not hold, A_{ijt} is not modified, while OUT_{jt} is modified as follows: let <h,k> ∈ IN_{jt} and w be a column of X_j which corresponds, through the projection list L_{ijt}, to the column k of the considered occurrence of X_j in E_i; then, enter <h,w> in OUT_{jt}. We say that dependencies of IN_{jt} are passed to OUT_{jt}.

3. E_i is a "Projection J-equation" (i.e. a J-equation without join operations but with just one projection):
   a. OUT_{jt} is derived as in case 2.b.2.
   b. A_{ijt} = ∅.

3.6.4. Traversal algorithm

Let a_c be the arity of C. The initial step of the traversal algorithm is as follows:

a. If E_i is a U-equation, then OUT_{0} = {<1,1>, <2,2>, ..., <a_c,a_c>}

b. If E_i is a J-equation, set initially OUT_{0} = ∅. 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_{0}. Iterate this construction for all positions of X_c.

c. Let M_c = { l_1, ..., l_n } be the marking set of C; initialize E_{C'} to the expression.

Proceedings of the 13th VLDB Conference, Brighton 1987
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 node analysis consists of the following steps:
   a. Generate a new timestamp $t$: $t = \text{new time}$.
   b. Evaluate the set $IN_{jt}$ as:
      
      $$IN_{jt} = \bigcap_{q < t} OUT_{ju} - IN_{iq}$$
      
      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 termination test:
      
      $$IN_{jt} \subseteq D_j$$
      
      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 basic step 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 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 if 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.

Theorem 1. $S$ 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. $N_c = \text{number of columns of } C$;

b. $N_j = \text{number of columns of } X_j$.

c. $N_{ij} = \text{number of occurrences of } X_j \text{ in } E_i$.

a. The number of C-dependencies in $D_j$ is $O(N_c \cdot N_j)$; 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 $X_j$ in $E_i$, and to the number of C-dependencies, hence is $O(N_c \cdot N_j \cdot N_{ij})$.
The maximum number of traversals is bound by the maximum $D_j$ times the number of nodes in $G(C)$, hence is $O(N^c \times N_j \times |X_C|)$. 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 well-known 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 $Q_2: \sigma_2 = a X_3$ on system $S_2$. The marked system is:

$$(X_2:2) = (N_2:2) \cup (C_2:2)$$
$$(N_2:2) = X_3 ^ o \circ (C_2:2)$$
$$X_3 = N_2 \cup C_4$$
$$N_2 = X_3 \circ C_2$$

We can reduce constants $C_4$ and $C_2$:

$$V_C = \sigma_2 = a C_4 \cup C_4 \bowtie_2 = 1 V_C$$
$$V_C = \sigma_2 = a C_2 \cup C_2 \bowtie_2 = 1 V_C$$

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:

$$V_R = \sigma_2 = a R \cup R \bowtie_2 = 1 V_R$$
$$V_C = \sigma_2 = a C \cup (C \bowtie_2 = 1 V_R) \bowtie_2 = 1 L$$

The reduction of $R$ corresponds to the "cone" of the Magic Set method ([Ban 86a]).

Example 6.c

Consider the system:

$$X_1 = L \circ X_2 \circ R$$
$$X_2 = \prod_{1=1}^{3} X_3$$
$$X_3 = X_1 \cup C$$

and the query $Q: \sigma_2 = a X_3$, producing the marking:

$$(X_4:2) = (X_1:2) \cup (C:2)$$
$$(X_2:2) = L \circ X_2 \circ (R:2)$$
$$X_1 = L \circ X_2 \circ R$$
$$X_2 = X_1 \cup C$$

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

$$V_R = \sigma_2 = a R \cup R \bowtie_2 = 2 L$$
$$V_C = \sigma_2 = a C \cup (C \bowtie_2 = 1 V_R) \bowtie_2 = 2 L$$

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:

$$(X_4:2) = (N_2:2) \cup (X_1:2)$$
$$(X_3:2) = (N_2:2) \cup (K:2)$$
$$(N_2:2) = X_2 \circ (R:2)$$
$$(N_2:2) = X_2 \circ (C:2)$$
$$X_1 = N_1 \cup X_2$$
$$X_2 = N_1 \cup K$$

Proceedings of the 13th VLDB Conference, Brighton 1987
Here the reduction of variables is not possible either; we obtain the reduced constants:

\[ V_K = \sigma_{2}^{a} K U K \bowtie_{2}^{a} = V_R U K \bowtie_{2}^{a} V_C \]
\[ V_C = \sigma_{2}^{a} C U C \bowtie_{2}^{a} = V_C \]
\[ V_R = \sigma_{2}^{a} R U R \bowtie_{2}^{a} V_C \]

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 0. 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:
\[
X_i^j = E_i(X_1^{j-1}, \ldots, X_n^{j-1})
\]

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 \( X_k^j \), \( k<i \):
\[
X_i^j = E_i(X_1^{j-1}, \ldots, X_{i-1}^{j-1}, X_i^{j-1}, \ldots, X_n^{j-1})
\]
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.


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 \( E \) which is linear with respect to its own variable \( X \); 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 \( X = E \) is linear with respect to its own variable:

\[ E_L(X_1, \ldots, X_l) = E_L(X_1, \ldots, X_l, \ldots, X_n) \]

Then the classical Jacobi algorithm:

**Algorithm A1**

FOR \( i:=1 \) TO \( n \) DO \( X_i = \phi \);
REPEAT
\[ \text{cond} := \text{true}; \]
FOR \( i:=1 \) TO \( n \) DO \( S_i := X_i \);
FOR \( i:=1 \) TO \( n \) DO
\[ \text{BEGIN} \]
\[ X_i := E_i(S_1, \ldots, S_n); \]
\[ \text{IF} \ X_i \neq S_i \text{ THEN} \text{cond} := \text{false}; \]
\[ \text{END}; \]
UNTIL \( \text{cond}; \)
FOR \( i:=1 \) TO \( n \) DO \( \text{OUTPUT}(X_i) \).

The advantage of algorithm A2 with respect to A1 is that the term \( D_l \) represents just the "difference" tuples evaluated at each iteration, while \( S_l \) represent the "accumulation" of all tuples of the previous iterations; given that \( |D_l| << |S_l| \), 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 [CGU 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 Xi appearing more than once in E; 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 selects a selection predicate which is "looser" than the initial one, and it completes 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 Xi as follows:

\[ X_i: m = E_i(X_{i,m_1} \ldots X_{i,m_n}) \]
\[ X_i: k = E_i(X_{i,k_1} \ldots X_{i,k_n}) \]

it is possible to substitute them with:

\[ X_i: m \text{ or } k = E_i(X_{i,m_1} \text{ or } k_1 \ldots X_{i,m_n} \text{ or } k_n) \]

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:

\[ R = B \cup \bigcup_{i=1}^3 \bigcup_{j=1}^3 \bigcup_{k=1}^3 (A_{i,j,k} \cap S_{i,j,k} \cup S_{i,j,k} \cap T_{i,j,k}) \]

with the query \( Q(x,y) = R(x,y,a). \) Our push of selections to variables succeeds, producing:

\[ (R:2v3)=(B:2v3) \cup \bigcup_{i=1}^3 \bigcup_{j=1}^3 \bigcup_{k=1}^3 (A_{i,j,k} \cap S_{i,j,k} \cup S_{i,j,k} \cap T_{i,j,k}) \]

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 [Bau 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 [Coo 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 of a given arity \( n \), applied to \( n \) terms \( T_j \). Thus, \( X = F(x, g(y, z)) \) is a valid equation. They introduce a notion of uniform 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 "Meteo"., of the Italian CNR, and of the MPI 40% Project.

REFERENCES


[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)


[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)