

A Practical Algorithm for Reformulation of Deductive Databases

Michael Genesereth and Abhijeet Mohapatra

Stanford University, Stanford, CA - 94305, USA
{genesereth, abhijeet}@cs.stanford.edu

Abstract. Database Reformulation is the process of rewriting the data and rules in deductive databases in a functionally equivalent manner, ideally in ways that decrease query processing time while keeping storage costs within acceptable bounds.

Early research in this area focussed on materializing existing views, i.e. caching those views as data. Subsequent research extended this work to the problem of inventing new views to afford more opportunities for materialization. The *Bounding Theorem*, introduced in this latter effort, is significant in that it gives a finite bound on the number of useful reformulations for conjunctive queries. Unfortunately, the number of possibilities allowed by the Bounding Theorem is doubly exponential.

In this paper, we show that we can reduce that double exponential to a single exponential. We first present an improved version of the Bounding Theorem, called the *Subgoal Theorem*. We then present an additional result, called the *Projection Theorem*. Finally, we describe a reformulation algorithm that runs in exponential time; and, using the Subgoal Theorem and the Projection Theorem, we show that the algorithm is correct and complete for all conjunctive queries, i.e. it produces reformulations that are as good as or better than any other reformulations (in terms of worst-case query evaluation performance with at most linear growth in storage space).

1 Introduction

Database Reformulation is the process of rewriting the data and rules in deductive databases in a functionally equivalent manner, ideally in ways that decrease query processing time while keeping storage costs within acceptable bounds.

As an example, consider a deductive database with three *base* (i.e. stored) relations p , q , and r and a *view* relation `query` defined in terms of these base relations. The definition of `query` is shown below (using Prolog-like syntax).

$$\text{query}(X, Y, Z) \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z)$$

The relation `query` is true of an object X and an object Y and an object Z if and only if there is an object A such p is true of X and A and q is true of X and A and r is true of Y

and Z .

If we assume that the data associated with each base relation is stored as a linear list, then we can use this rule to determine whether query is true of any specific combination of values for X and Y and Z . Assuming relational indexing and nested loop evaluation, the worst case query processing cost is $O(n^3)$, where n is the cardinality of the domain of objects in the database.

In an effort to improve query processing time, we could materialize the query relation. However, it is potentially much larger than any of the base relations; and, for large domains, materializing query might be impractical. Moreover, checking a specific instance can still take $O(n^3)$ steps. All is not lost. Suppose we define a new relation v in terms of p and q as follows.

$$v(X) \text{ :- } p(X, A) \ \& \ q(X, A)$$

Once we have v , we can define *query* in a functionally equivalent way as shown below. The relation *query* is true of X and Y and Z if and only if v is true of X and r is true of Y and Z .

$$\text{query}(X, Y, Z) \text{ :- } v(X) \ \& \ r(Y, Z)$$

If we materialize v and use this rule to compute *query*, we get a worst case runtime that is $O(n^2)$. For large n , this cost is much lower than the original version. Moreover, we pay little in storage - since the cardinality of v is smaller than that of either p or q , the storage required to store v is no greater than the storage required to store p or q .

Existing research on database reformulation describes various methods for finding reformulations of this sort. This research includes selecting views to materialize [1–3, 8–11, 13] and inventing additional views of the base data to afford more opportunities for materialization [5, 4, 6, 7].

For the purposes of this paper, the most interesting result in this work is Chirkova's *Bounding Theorem*, introduced in [4]. The Bounding Theorem is significant in that it gives a finite bound on the number of reformulations that must be considered for conjunctive queries. The downside is that the set of possibilities is doubly exponential in terms of the size of the query.

In this paper, we show that we can reduce that double exponential to a single exponential. We first present an improved version of the Bounding Theorem, called the *Subgoal Theorem*. We then present an additional result, called the *Projection Theorem*. Finally, we describe a reformulation algorithm that runs in singly exponential time; and, using the Subgoal Theorem and the Projection Theorem, we show that the algorithm is correct for all conjunctive queries, i.e. it produces reformulations that are as good as or better than any other reformulations (in terms of worst-case query evaluation performance with at most linear growth in storage space).

2 Preliminaries

The *vocabulary* for a database is a collection of *object constants* and *relation constants*. Each relation constant has an associated *arity*, i.e. the number of *arguments* allowed in any expression involving that constant.

A *factoid* is an expression formed from an n -ary relation constant and n object constants. In what follows, we write factoids in traditional mathematical notation - the relation constant followed by its *arguments* enclosed in parentheses and separated by commas. For example, if r is a binary relation constant and a and b are object constants, then $r(a, b)$ is a factoid.

The *Herbrand base* for a database is the set of all factoids that can be formed from the constants in its vocabulary. For example, for a vocabulary with just two object constants a and b and a single binary relation constant r , the Herbrand base is $\{r(a, a), r(a, b), r(b, a), r(b, b)\}$. A *dataset* is any subset of the Herbrand base, i.e. an arbitrary set of the factoids that can be formed from the vocabulary of the database.

It is possible to define relations in terms of others. In this presentation, we encode such definitions in a logic language called Datalog. We could equally well use other database languages, like SQL; however, Datalog is well-studied; it is easy to understand; and it serves our purposes here.

One key difference between Datalog and the language of data is the inclusion of a new type of symbol, called a *variable*. Variables allow us to state relationships among objects without explicitly naming those objects. In what follows, we use individual capital letters as variables, e.g. X, Y, Z .

Datalog *atoms* are analogous to factoids except that they can optionally contain variables in place of object constants. For example, if r is a binary relation constant, if a is an object constant, and if X and Y are variables, then $r(a, X)$ is an atom, as are $r(a, Y)$ and $r(X, Y)$ and $r(X, X)$.

A Datalog *rule* is an expression consisting of a distinguished atom, called the head, and zero or more atoms, together called the body. We write rules as in the example shown below. Here, $r(X)$ is the head, the expression $p(X, Y) \ \& \ q(Y)$ is the body; and $p(X, Y)$ and $q(Y)$ are subgoals.

$$r(X) \text{ :- } p(X, Y) \ \& \ q(Y)$$

Semantically, a rule is something like a reverse implication. It is a statement that the conclusion of the rule is true whenever the conditions are true. For example, the rule above states that r is true of any object X *if* there is an object Y such that p is true of X and Y and q is true of Y . For example, if we know that $p(a, b)$ and $q(b)$ are true, then, using this rule, we can conclude that $r(a)$ is true.

A rule is *safe* if and only if every variable that appears in the head also appears in at least one subgoal. Note that it is okay to have variables in the body that do not appear in the head. In what follows, we assume that all rules are safe.

A Datalog program is a collection of safe Datalog rules. A program consisting of a single rule of the form shown above is called a *conjunctive query* (or CQ). In this paper, we concentrate on conjunctive queries and Datalog programs that are equivalent to such queries in that they give the same results for the head of the conjunctive query when applied to the same set of base relations.

Given a dataset and a rule, we can apply the rule to the factoids in the dataset to determine the truth or falsity of a given instance of the rule's head or to compute all instances of the head that satisfy the body. In what follows, we assume relational indexing and nested loop evaluation. By relational indexing, we mean that, for every relation, we have a list of all factoids in the dataset containing that relation. By nested loop evaluation, we mean that we evaluate each subgoal in turn and we evaluate the remaining subgoals for each solution.

Analyzing the worst-case complexity of this algorithm is quite simple. Consider a dataset with binary relation constant p and unary relation constant q and a domain consisting of n object constants; and consider the rule shown above. In this example, the worst case cost of determining the truth of a specific instance of r is n^2 . Given a value for X , we search all factoids in the dataset containing p to find those with that value as first argument. For each value of Y that we find, we search all factoids in the dataset containing q to find those with that value of Y as argument. Evaluating the first subgoal can require as many as n^2 steps. There are as many as n solutions. For each such solution, as many as n steps are required to check the second subgoal. Consequently, in this case, the algorithm requires $O(n^2)$ operations.

3 Reformulation

Consider a conjunctive query q that defines an *output relation* r in terms of a given set of *base relations*. An *input* to q is an arbitrary dataset involving the base relations in q . The *output* of q on a given input D is the dataset that results from applying q to D .

A compound query consists of a collection of view definitions in terms of a given set of *base relations* and a definition of an *output relation* r in terms of the base relations and these view relations. An *input* to a compound reformulation is an arbitrary dataset involving the base relations of the compound query. Let D be an input to a compound query and let D' be the result of applying the view definitions to D . In this case, the output of the compound query is the result of applying the definition of r to $D \cup D'$.

We say that a compound query q' is a *reformulation* of a conjunctive query q if and only if they are equivalent, i.e. (1) they have the same base relations and the same output

relation and (2) the output of q' is the same as the output of q for every input.

Note that a conjunctive query can be viewed as a reformulation of itself in which the set of view relations is empty.

We say that one reformulation is *as good as or better than* another if and only if the asymptotic worst case query cost for the first is as good as or better than the asymptotic worst case query cost for the second *and* the storage cost for the intermediate view relations in the first reformulation grows no more than linearly with growth in the storage cost for the base relations in the second.

In [4], Chirkova presents two significant results on reformulation - the *Infinity Theorem* and the *Bounding Theorem*.

Infinity Theorem: There is a conjunctive query for which there are infinitely many distinct reformulations.

Bounding Theorem: For any reformulation q' of a conjunctive query q , there is a reformulation q'' of q in which the bodies of all views have no more subgoals than the original query and q'' is as good as or better than q' .

The Bounding Theorem is significant in that it selects finitely many reformulations from the infinity of possibilities guaranteed by the Infinity Theorem. Unfortunately, it still leaves a set of possibilities that is doubly exponential. The good news is that we can do better. In the next two sections, we introduce two theorems that further limit the number of reformulations that must be considered, and we then present a reformulation algorithm that produces an optimal reformulation in singly exponential time.

4 Subgoal Theorem

The Subgoal Theorem assures us that, to find an optimal reformulation of a conjunctive query, it suffices to look at views in which the bodies consist entirely of subgoals in the original query.

Theorem 1 (Subgoal Theorem). *For any reformulation q' of a conjunctive query q , there is a reformulation q'' that is as good or better than q' in which the bodies of all views are subsets of the body of q .*

Proof. Consider an arbitrary query q :- p_1, \dots, p_m , and consider a reformulation q' :- v'_1, \dots, v'_k , where v'_i :- p_{i1}, \dots, p_{im_i} . Without loss of generality, let us assume that the existential variables in the definition of each v'_i are distinct from the existential variables in the definitions of other views. (Existential variables are variables that appear in the body of a view definition but not in the head.)

Expanding the views in the definition of q' , we get $p_{11}, \dots, p_{1m_1}, \dots, p_{k1}, \dots, p_{km_k}$. Since q and q' are equivalent CQs, we know there is a containment mapping from this

expansion to the subgoals in the definition of q , i.e. there must be a binding of variables of the expansion that makes it identical to a subset of those subgoals.

Now consider the rewrite $q'' :- v''_1, \dots, v''_k$ where the definition of each view v''_i is replaced by the result of applying the containment mapping to v'_i .

As a simple example of this construction, consider the query shown below.

$$q(X, Y, Z) :- p(X, Y) \ \& \ p(Y, Z)$$

Now consider the following reformulation of this query.

$$\begin{aligned} q'(X, Y, Z) & :- v1'(X, Y, Z) \ \& \ v2'(X) \\ v1'(X, Y, Z) & :- p(X, Y) \ \& \ p(Y, Z) \\ v2'(X) & :- p(X, Y2) \ \& \ p(Y2, Z2) \end{aligned}$$

By substituting the views definitions into the definition of q' , we end up with the following expansion.

$$qe'(X, Y, Z) :- p(X, Y) \ \& \ p(Y, Z) \ \& \ p(X, Y2) \ \& \ p(Y2, Z2)$$

In this case, there is a simple containment mapping that shows that this query contains the original query. (The reverse direction is even simpler.)

$$\begin{aligned} Y2 & \rightarrow Y \\ Z2 & \rightarrow Z \end{aligned}$$

Applying this containment mapping to the given reformulation results in the reformulation shown below.

$$\begin{aligned} q''(X, Y, Z) & :- v1''(X, Y, Z) \ \& \ v2''(X) \\ v1''(X, Y, Z) & :- p(X, Y) \ \& \ p(Y, Z) \\ v2''(X) & :- p(X, Y) \ \& \ p(Y, Z) \end{aligned}$$

It is easy to see that q' and q'' are equivalent. After applying the rules defining each v''_i in the definition of q'' , we end up with the same result as applying the containment mapping to the expansion of q' (after removing duplicates and reordering).

Moreover, the new reformulation is as good as or better than the given reformulation. Each v'_i contains v''_i , which can be shown using the containment mapping discussed above. Consequently, the size of v''_i is less than or equal to the size of v'_i , meaning that computational cost and storage size for v''_i are bounded by the computational cost and storage size for v'_i .

Finally, for the purposes of our theorem, the new reformulation is defined entirely in terms of subgoals of the original query. \square

5 Projection Theorem

The value of the Subgoal Theorem is that it gives us an easy method for enumerating the view definitions used in useful reformulations - we just need to consider subsets of the subgoals in the original query and no more. In this section, we show that we can further narrow the set of possible reformulations by taking advantage of the linear storage growth constraint.

A *projection view* is a view in which the body of a view definition includes at least one subgoal that contains all of the variables in the head of the definition.

For example, the view defined below is a projection view. All of the variables in the head appear in the first subgoal in the body.

$$v(X, Y) :- p(X, Y) \ \& \ q(Y, Z)$$

By contrast, the following view is not a projection view. Although the rule is safe, neither of the subgoals contains all of the variables in the head.

$$v(X, Z) :- p(X, Y) \ \& \ q(Y, Z)$$

Theorem 2 (Projection Theorem). *In any reformulation that satisfies the linear growth constraint, every view must be a projection view.*

Proof. Since all view definitions are safe, then every variable in the head must appear in at least one subgoal. If there is no subgoal whose variables include all of the variables in the head, then the head variables must be split across multiple subgoals. Let u and v be two such variables. Consider a dataset that satisfies the body of the view definition with n distinct values for u and n distinct values for v . This results in at least n^2 instances of the head. Moreover, any increase in n leads to a quadratic increase in the number of factoids that satisfy the head. Hence, the linear storage constraint is violated. \square

The merit of projection views is that the cardinality of each defined relation is guaranteed to be no larger than the cardinality of the relation containing the head variables. Although the size of a non-projection view *can* be smaller than the size of the relations in terms of which it is defined, it can also be larger, and we are concerned here with worst-case analysis. The upshot is that, the only way we can *guarantee* to satisfy the linear growth constraint is to allow only projection views in our reformulations.

6 Reformulation Algorithm

The Subgoal Theorem and the Projection Theorem decrease the number of reformulations we need to consider to find an optimal reformulation of a conjunctive query. We need look only at views that are defined in terms of subgoals and, among these, we need look only at projection views.

We start this section by describing an algorithm that computes a reformulation that is as good as or better than any other reformulation *and* the algorithm runs in time that is singly exponential in the size of the original view definition. The input to the algorithm is a rule defining a conjunctive query, and the output is an optimal reformulation of the given query.

- (1) **Construct the set of maximal projection views.** Let R be the empty set. For each s_i in the body of the original query, create an atom consisting of a new relation constant v_i and the arguments in the head of s_i , form a rule with this new atom as head and with the subgoals in the original query as body, and add the resulting rule to R . Once that is done, add to R an overall rule in which the head is the head of the original rule and the body consists of the heads of the rules in R .
- (2) **Remove unneeded occurrences of variables.** Consider all subsets of variable occurrences in the heads of the rules in R . For each such subset, drop those variables from the view heads in R and the overall rule and check for equivalence to the original query. Then compare all surviving possibilities to each other to find a set of reformulations that minimizes worst-case query cost. Let R be the set reformulations resulting from this process.
- (3) **Minimize view definitions.** If the head of a view definition in R contains a subset of the variables in the head of some other view definition, delete the former view definition from R and delete the corresponding view head from the overall rule.

As an example of this algorithm in operation, consider the query shown below. Note that every subgoal contains variables that are not used in the head.

$$\text{query}(X, Y, Z) \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z)$$

In step 1, we construct a set of maximal projection views and a new definition of the original query.

$$\begin{aligned} \text{query}(X, Y, Z) & \text{ :- } v1(X, A) \ \& \ v2(X, A) \ \& \ v3(Y, Z) \\ v1(X, A) & \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \\ v2(X, A) & \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \\ v3(Y, Z) & \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \end{aligned}$$

In step 2, we try removing the variables X and A from the first view and discover that the resulting reformulation is equivalent to the original, so we drop those variables from the head of $v1$ and modify the overall rule accordingly. We discover that we can remove A from the second view at the same time and still preserve equivalence, and so we drop that occurrence as well. Finally, we by comparing to other equivalent we find that this reformulation is as good as or better than any other.

$$\begin{aligned} \text{query}(X, Y, Z) & \text{ :- } v1() \ \& \ v2(X) \ \& \ v3(Y, Z) \\ v1() & \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \\ v2(X) & \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \\ v3(Y, Z) & \text{ :- } p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \end{aligned}$$

Finally, in step 3 we check each view to see whether it can be eliminated. In this case, we can remove either $v1$, but we must keep one of the two view definitions. Assuming we remove $v1$, we end up with the following reformulation.

$$\begin{aligned} \text{query}(X, Y, Z) & :- v2(X) \ \& \ v3(Y, Z) \\ v2(X) & :- p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \\ v3(Y, Z) & :- p(X, A) \ \& \ q(X, A) \ \& \ r(Y, Z) \end{aligned}$$

As another example, consider the following query. This one is more interesting in that it involves multiple existential variables, and a final reformulation in which it is essential to retain at least one of these variables in the heads of view definitions.

$$\text{query}(X, Z) :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z)$$

In step 1, as before, we form a new definition of the original query and add the set of maximal projection views.

$$\begin{aligned} \text{query}(X, Z) & :- v1(X, A) \ \& \ v2(A, Z) \ \& \ v3(X, B) \ \& \ v4(B, C) \ \& \ v5(C, Z) \\ v1(X, A) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v2(A, Z) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v3(X, B) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v4(B, C) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v5(C, Z) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \end{aligned}$$

In step 2, we minimize the set of variables in the heads of the views, subject to the containment constraint and select an optimal reformulation from among the surviving possibilities. The result is shown below.

$$\begin{aligned} \text{query}(X, Z) & :- v1(X, A) \ \& \ v2(A, Z) \ \& \ v3(X, B) \ \& \ v4(B, C) \ \& \ v5(C, Z) \\ v1(X, A) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v2(A, Z) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v3() & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v4() & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v5() & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \end{aligned}$$

In step 3, we eliminate any views with head variables that are contained in the heads of other views, in this case eliminating views $v3$, $v4$, and $v5$.

$$\begin{aligned} \text{query}(X, Z) & :- v1(X, A) \ \& \ v2(A, Z) \ \& \ v3(X, B) \ \& \ v4(B, C) \ \& \ v5(C, Z) \\ v1(X, A) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \\ v2(A, Y) & :- p(X, A) \ \& \ q(A, Z) \ \& \ r(X, B) \ \& \ s(B, C) \ \& \ t(C, Z) \end{aligned}$$

Theorem 3 (Correctness Theorem). *For any reformulation q' of a conjunctive query q , the reformulation algorithm produces a reformulation q'' that is as good as or better than q' .*

Proof. The Bounding Theorem assures us that, for any reformulation of a conjunctive query, there is a reformulation in which the body of any constituent consists entirely of

subgoals of the original query. Adding additional subgoals from the original query to any view in any optimal reformulation cannot change equivalence of the reformulation since containment in the original query still applies in both directions. And it cannot change optimality since the only effect is to produce possibly smaller views. Hence, without loss of generality, we can focus on reformulations in which all bodies contain all subgoals of the original query.

The Projection Theorem assures us that, for any reformulation of a conjunctive query, there is a reformulation in which the variables in the head of any constituent view are included in the variables of at least one of the subgoals. Consequently, we know that the head variables in any optimal reformulation are included among the head variables of at least one of the views in the output of step 1.

By construction, the result of step 2 is equivalent to the original query and no other variables can be removed without violating this equivalence.

At the start of step 3, we drop any views where the set of head variables is a subset of the variables in another view. It is easy to see that doing so preserves equivalence and improves query execution time. Moreover, we cannot delete any other views. If that were possible, the head variable occurrences would have been eliminated in step 2. \square

A straightforward computational analysis of the algorithm shows that it runs in time that is exponential in the size of the original query.

The reformulation algorithm, as given, can be improved in a variety of ways to eliminate some of the existential variables that must be considered in step 2 and thereby decrease the runtime of that step. For example, if an existential variable occurs in just one view head, it can be dropped from that view head.

Another such pruning heuristic is to consider only reformulations that preserve *connectivity* among head variables. Given a query, two variable sets X and Y are *connected* with respect to the query if either (a) $X = Y$, (b) the body of the query contains an atom with variable sets X and Y , or (c) there exists a variable set U such that X and U are connected and U and Y are connected.

For example, in the following query, there are two paths between the head variables X and Y , one of which goes through A and the other B .

$$\text{query}(X, Y) \text{ :- } r1(X, A) \ \& \ r2(A, Y) \ \& \ r3(X, B) \ \& \ r4(B, Y)$$

Lemma 1 (DPP Lemma). *For every pair of head variables that are connected in a query, there exists a unique path connecting the variables in an optimal reformulation of the query.*

A proof of the DPP lemma is presented in [12]. We can apply the DPP lemma in Step 3 of our reformulation algorithm to filter the list of reformulations to be checked for

equivalence, which is potentially an expensive operation.

While these pruning heuristics can safely decrease or even eliminate the exponential in some cases, they cannot always eliminate that cost. There are cases where exponential search is still necessary. In fact, it is possible to show that the problem of computing an optimal reformulation of a conjunctive query is NP-hard by proving that the problem is isomorphic to the Steiner Tree problem [12].

Note that it is also possible to decrease the number of subgoals in each view definition by eliminating subgoals that are not connected to the variables in the heads of the view definitions. This has no effect on worst-case optimality of query evaluation. However, it can decrease the cost of materialization, which in some cases may be significant.

For example, applying this rule to the first reformulation above would lead to the equivalent but simpler reformulation shown below.

$$\begin{aligned} \text{query}(X, Y, Z) & :- v2(X) \ \& \ v3(Y, Z) \\ v2(X) & :- p(X, A) \ \& \ q(X, A) \\ v3(Y, Z) & :- r(Y, Z) \end{aligned}$$

A demonstration of the reformulation algorithm is available through the following link.

<http://logic.stanford.edu/epilog/experiments/reformulation.html>

The code for the algorithm, written in Javascript, can be found by examining the source for the demonstration.

7 Conclusion

One arguable limitation of the Bounding Theorem, as well as the results reported here, is that they concentrate exclusively on conjunctive queries. The good news is that the results can easily be extended to more complex queries, e.g. union queries and semi-positive queries [14]. Unfortunately, the results break down in the presence of full negation or recursion or aggregates; and further work is needed to deal with such cases.

Another limitation is that we do not take into account known database constraints, e.g. functional dependencies, inclusion dependencies, or mutual exclusion constraints. Fortunately, it is possible to accommodate some of these constraints in many cases by using a variant of the chase algorithm to fold the constraints into the view definitions before applying the reformulation algorithm (as suggested in [6, 7]). (The main problem in doing this arises when the chase algorithm does not terminate. Also, for some constraints, our results would need to be extended to conjunctive queries with equalities and/or inequalities.)

Another limitation of our results is the emphasis on worst-case analysis. The theorems presented here promise only that reformulations are as good as or better than other

reformulations in the worst case. More sophisticated processing is necessary to show results about average time computation or to take database statistics into account.

A final limitation worth noting is that these results do not take into account the costs of creating or updating materialized views in the face of changes. If materialization and update costs are considered, reformulations that would be appropriate for static databases may not longer be justified for rapidly changing databases. Once again, there may be good news here. A preliminary investigation of update costs suggests that, in the case of differential update, the reformulation algorithm presented here produces reformulations for which the update costs are bounded by the costs of using the resulting reformulations, so this may not be a significant worry.

The Bounding Theorem is significant in that it selects finitely many reformulations to consider from the infinity of possible reformulations. Unfortunately, it still leaves a set of possibilities that is doubly exponential in query size. The reformulation algorithm presented here is significant in that it is correct yet runs in singly exponential time, making it feasible to employ database reformulation in practical deductive database systems.

The broader value of this work is that it illustrates the feasibility and computational value of automated problem reformulation, i.e. changing the vocabulary / conceptual basis of problems as a precursor to ordinary, formulation-preserving deductive reasoning.

References

1. Abiteboul, S., Duschka, O.M.: Complexity of answering queries using materialized views. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA. pp. 254–263 (1998), <http://doi.acm.org/10.1145/275487.275516>
2. Baralis, E., Paraboschi, S., Teniente, E.: Materialized views selection in a multidimensional database. In: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece. pp. 156–165 (1997), <http://www.vldb.org/conf/1997/P156.PDF>
3. Chirkova, R.: The view-selection problem has an exponential-time lower bound for conjunctive queries and views. In: Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA. pp. 159–168 (2002), <http://doi.acm.org/10.1145/543613.543634>
4. Chirkova, R., Genesereth, M.R.: Linearly bounded reformulations of conjunctive databases. In: Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings. pp. 987–1001 (2000), https://doi.org/10.1007/3-540-44957-4_66
5. Chirkova, R., Genesereth, M.R.: Linearly bounded reformulations of unary databases. In: Abstraction, Reformulation, and Approximation, 4th International Symposium, SARA 2000, Horseshoe Bay, Texas, USA, July 26-29, 2000, Proceedings. pp. 144–163 (2000), https://doi.org/10.1007/3-540-44914-0_9
6. Chirkova, R., Genesereth, M.R.: Database reformulation with integrity constraints (extended abstract). CoRR abs/cs/0506026 (2005), <http://arxiv.org/abs/cs/0506026>

7. Chirkova, R., Genesereth, M.R.: Equivalence of SQL queries in presence of embedded dependencies. In: Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA. pp. 217–226 (2009), <http://doi.acm.org/10.1145/1559795.1559829>
8. Chirkova, R., Halevy, A.Y., Suciu, D.: A formal perspective on the view selection problem. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy. pp. 59–68 (2001), <http://www.vldb.org/conf/2001/P059.pdf>
9. Gupta, H.: Selection of views to materialize in a data warehouse. In: Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings. pp. 98–112 (1997), https://doi.org/10.1007/3-540-62222-5_39
10. Gupta, H., Mumick, I.S.: Selection of views to materialize under a maintenance cost constraint. In: Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings. pp. 453–470 (1999), https://doi.org/10.1007/3-540-49257-7_28
11. Levy, A.Y., Mendelzon, A.O., Sagiv, Y., Srivastava, D.: Answering queries using views. In: Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA. pp. 95–104 (1995), <http://doi.acm.org/10.1145/212433.220198>
12. Mohapatra, A., Genesereth, M.: Optimal CQ reformulation is NP-Complete. Tech. Rep. LG-2018-01, Stanford University, Stanford, CA (2018), <http://logic.stanford.edu/reports/LG-2018-01.pdf>
13. Ross, K.A., Srivastava, D., Sudarshan, S.: Materialized view maintenance and integrity constraint checking: Trading space for time. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996. pp. 447–458 (1996), <http://doi.acm.org/10.1145/233269.233361>
14. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. *J. ACM* 27(4), 633–655 (1980), <http://doi.acm.org/10.1145/322217.322221>