A Practical Algorithm for Reformulation of Deductive Databases

Michael Genesereth
Stanford University
Stanford, CA
genesereth@stanford.edu

Abhijeet Mohapatra
Stanford University
Stanford, CA
abhijeet@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 investigated the problem of inventing new views to afford different 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 the size of the query (not the database instance).

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 present a reformulation algorithm that runs in 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 reformulation (in terms of worst-case query evaluation performance with at most linear growth in storage space).

CCS CONCEPTS

• **Information systems → Data management systems**; **Database design and models**; **Database query processing**;

KEYWORDS

Logic Programming, Deductive Databases, Reformulation, Query Optimization, Schema Reformulation

ACM Reference Format:
Michael Genesereth and Abhijeet Mohapatra. 2019. A Practical Algorithm for Reformulation of Deductive Databases. In *Proceedings of ACM SAC Conference (SAC'19)*. ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/xx.xxx/xxx_x

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 four *base* (i.e. stored) relations `p`, `q`, `r`, and `s` and a *view* relation `query` defined in terms of these base relations. The definition of `query` is shown below (using Prolog-like syntax).

```
query(X,Y,Z) :- p(X,A) & q(X,A) & r(A,Y) & s(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 `A` and `Y` and `s` 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 $O(n^5)$, 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` and `r` as follows.

```
v(X,Y) :- p(X,A) & q(X,A) & r(A,Y)
```

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 `Y` and `s` is true of `Y` and `Z`.

```
query(X,Y,Z) :- v(X,Y) & s(Y,Z)
```

If we materialize `v` and use this rule to *query* a specific instance, 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 `p`, the storage required to store `v` is no greater than the storage required to store `p`.

To make this example concrete, imagine a database of cities. Let `p` be true of pairs of cities in the same country. Let `q` and `r` be true of pairs of cities connected by a non-stop train ride. Let `s` be true of pairs of cities connected by a non-stop flight. In this case, `query` is the set of all triples `X`, `Y`, `Z` where `X` and `Y` are cities in the same country connected by a one-stop train ride for which there is a non-stop flight from `Y` to `Z`. And, in this case, the view `v` is the set of all pairs of cities in the same country that are connected by a one-stop train ride.

Existing research on database reformulation describes various methods for finding reformulations of this sort. This research includes *selecting* pre-existing views to materialize [1, 2, 4, 9, 11–14] and *inventing* additional views of the base data to afford more opportunities for materialization [5–8].

SAC'19, April 8-12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04.
https://doi.org/xx.xxx/xxx_x

For the purposes of this paper, the most interesting result in this work is Chirkova's *Bounding Theorem*, introduced in [5]. 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*. Next, we describe a reformulation algorithm that runs in singly exponential time in the size of the query (not the database instance); 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). We then describe a few ways to improve the performance of the reformulation algorithm. We show that the the problem of computing optimal reformulations of a conjunctive query is isomorphic to the Steiner Tree problem [10], a well known NP-complete problem. This guarantees that it is not possible in all cases to eliminate the exponential search involved in computing optimal reformulations. Finally, we mention some limitations of our results and suggest some directions for future work.

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) :- p(X,Y) & q(Y)
```

Semantically, a rule is a reverse implication. It is a statement that the head of the rule is true whenever the subgoals 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 [5], 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

Once again, consider the query from Section 1.

```
query(X,Y,Z) :- p(X,A) & q(X,A) & r(A,Y) & s(Y,Z)
```

Since the definition of `query` contains four subgoals, the Bounding Theorem assures us that we need consider only those views that can be defined with four or fewer subgoals. Unfortunately, the theorem does not place any restrictions on the patterns of arguments in these subgoals, and there are many such patterns. In this case, we might need to consider views defined with various combinations of the following subgoals: `p(X,X)`, `p(X,Y)`, `p(Y,X)`, `p(X,A)`, `p(A,X)`, `p(X,Z)`, `p(Z,X)`, `p(Y,Z)`, `p(Z,Y)`, `q(X,X)`, `q(X,Y)`, `q(Y,X)`, `q(X,A)`, `q(A,X)`, `q(X,Z)`, `q(Z,X)`, `q(Y,Z)`, `q(Z,Y)`, and so forth. While some combinations of these subgoals are identical to others under variable renaming, there are many that are not.

By contrast, the Subgoal Theorem (stated below) 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. In this case, we need to consider only view definitions with combinations of `p(X,Y)`, `q(X,A)`, `r(A,Y)`, and `s(Y,Z)` as subgoals.

Theorem 4.1 (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 $q :- p_1, \ldots, p_m$, and consider a reformulation $q' :- v'_1, \ldots, v'_k$, where $v'_i :- p_{i1}, \ldots, p_{im_i}$. 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 $p_{11}, \ldots, p_{1m_1}, \ldots, p_{k1}, \ldots, p_{km_k}$. 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 $q'' :- v''_1, \ldots, v''_k$ where the definition of each view v''_i is replaced by the result of applying the containment mapping to v'_i.

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. □

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.

```
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)
```

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.)

$$\texttt{Y2} \rightarrow \texttt{Y}$$
$$\texttt{Z2} \rightarrow \texttt{Z}$$

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

```
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)
```

Of course, in this case, the reformulation has no practical value. However, the example shows how a candidate reformulation with

new subgoals can be replaced by one consisting entirely of subgoals of the original query.

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 5.1 (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. □

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 of the original query 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.

Step 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.

Step 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 by performing containment tests as described in [3]. 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.

Step 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 two subgoals contain variables that are not used in the head.

```
query(X,Y,Z) :- 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.

```
query(X,Y,Z) :- v1(X,A) & v2(X,A) & v3(Y,Z)
v1(X,A) :- p(X,A) & q(X,A) & r(Y,Z)
v2(X,A) :- p(X,A) & q(X,A) & r(Y,Z)
v3(Y,Z) :- p(X,A) & q(X,A) & r(Y,Z)
```

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 an still preserve equivalence, and so we drop that occurrence as well. By comparing this reformulation to other equivalent reformulations, we find that this reformulation is as good as or better than any other.

```
query(X,Y,Z) :- v1() & v2(X) & v3(Y,Z)
v1() :- p(X,A) & q(X,A) & r(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)
```

Finally, in step 3 we check each view to see whether it can be eliminated. In this case, we can remove `v1`, but we must keep the two other view definitions, resulting in the following reformulation.

```
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)
```

Theorem 6.1 (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. □

7 IMPROVEMENTS

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

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 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 last reformulation from Section 6 would lead to the equivalent but simpler reformulation shown below.

```
query(X,Y,Z) :- v2(X) & v3(Y,Z)
v2(X) :- p(X,A) & q(X,A)
v3(Y,Z) :- r(Y,Z)
```

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`.

```
query(X,Y) :- r1(X,A) & r2(A,Y) & r3(X,B) & r4(B,Y)
```

In fact, it is possible to show that *any rewriting* of a query using projection views is a reformulation if for every pair of head variables X and Y such that X and Y are connected with the respect to the query, the variables X and Y are also connected with respect to the rewriting by a unique path. We can use this result in step 3 of our reformulation algorithm to filter the list of reformulations to be checked for equivalence. We call such rewritings as *dpp-transforms*.

Lemma 7.1. *Every dpp-transform of a query is a reformulation.*

Proof. We prove the above lemma using induction on the size of the path between head variables. The scenario where a supplied query consists of only one head variable case is uninteresting because the materializing the query itself does not violate the linear storage constraint, and is the optimal reformulation. Therefore, we focus on the case where a supplied query has two head variables.

Base case. In this case, there exists a shortest path of length one between the head variables, say, X and Y, i.e., there exists an atom of the form $r(X, Y)$. We can materialize the query without violating storage constraint. If there exists a shortest path of length two, e.g. $q(X, Y)$:- $v_1(X, A)$, $v_2(A, Y)$, $v_3(X, B)$ $v_4(B, Y)$, we may either drop A or B to get an equivalent reformulation.

Inductive hypothesis. If the shortest path between the head variables is of length $\leq n$, the lemma holds, and we may eliminate variables preserving a unique path, and have an equivalent reformulation.

Inductive step. Suppose the shortest path between the head variables, say, X and Y is of length $n + 1$. Let N_y be the set of nodes = $\{A_1, A_2, \ldots, A_k\}$ that are directly connected to Y in such paths. Consider a rewriting of the dpp-transform where every view of the form $v(Y, \bar{A}, \bar{Z})$ where $\bar{A}(\subseteq N_y)$, is replaced by a view $t_v(Y, A_1, A_2, \ldots, A_k, \bar{Z})$. Clearly, this rewriting is equivalent to the dpp-transform. The shortest path between X and Y in the rewriting is n. Therefore, we can apply our induction hypothesis to prove the lemma.

We can extend our proof to queries with more than two variables by applying induction on the number of head variables. □

8 COMPLEXITY

A straightforward computational analysis of our reformulation algorithm described in Section 6 shows that it runs in time that is exponential in the size of the input query (not the database instance). This running time is dominated by the containment tests performed in step 2 of the algorithm.

While the pruning heuristics described in Section 7 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 [10].

In the following, we focus on queries with binary subgoals and prove that the time taken to compute optimal dpp-transforms is NP-hard in the size of the query.

Theorem 8.1. *For conjunctive queries with binary subgoals, the time taken to compute an optimal dpp-transform is NP-hard in the number of subgoals and free variables in the query.*

Proof. We prove NP-hardness by showing that the problem of computing an optimal dpp-transform is *isomorphic* to the Steiner Tree problem, which is NP-complete [10]. Given an undirected graph $G(V, E)$, a subset of the vertices R called terminal nodes, the Steiner Tree is there a subtree $G'(V', E')$ of G such that $R \subseteq V'$, and there does not exist a subtree $G''(V'', E'')$, where $R \subseteq V''$ and $| E'' | < | E' |$.

Given an instance of a Steiner Tree problem, we *reduce* it to an instance of finding an optimal dpp-transform as follows.

- We label the terminal nodes as $X_1, X_2, \ldots, X_{|R|}$. We label the edges where at least one of the end-points is a non-terminal node as $A_1, A_2, \ldots, A_{|E|}$.
- For every pair of edges A_i, A_j incident on a non-terminal node, we assign a relation, say $r_m(A_i, A_j)$.
- For every edge incident A_i on a terminal node X_j, we assign a relation, say $r_k(A_i, X_j)$.
- For every between two terminal nodes X_i, X_j, we assign a relation, say $r_l(X_i, X_j)$.
- Let ϕ be the conjunction of the above relations. The output dpp-transform instance is the following query.

$$q(X_1, X_2, \ldots, X_{|R|}) \text{ :- } \phi$$

Let q' be an optimal dpp-transform of q. We can leverage the definition of q' to construct the Steiner Tree of G as follows. We construct G' by removing the edges in G that correspond to the variables that are present in the definition of q, but missing in q'. Since q' is a dpp-transform, there exists a unique path between any two head variables X_i and X_j. As a result, there is a unique path in G' between any two nodes. Therefore, G' is a tree. G' must be a Steiner Tree; if not, then we could obtain a better dpp-transform by starting with the Steiner Tree and applying the reduction outlined above.

Given a conjunctive query q with binary subgoals, we *reduce* it to an instance of Steiner Tree problem as follows.

- We create a node for every variable in the query.
- For every subgoal, say, $r(X, Y)$, we create an edge between nodes in the graph corresponding to X and Y.
- The set of terminal nodes R is the set of nodes corresponding to the head variables in the query.

Consider a Steiner Tree of $G'(V', E')$ of G. We can use G' to construct an optimal dpp-transform q' of q as follows. We replace each subgoal $s_i(X, Y)$ of q using a view $v_i(X, Y)$ where v_i is defined as $v_i(X, Y)$:- body of q. We remove all views of the form $v_k(X_i, X_j)$ if there is no edge between the nodes corresponds X_i and X_j in G'. We drop all variables X_i such that the node corresponding to X_i does not appear in G'. Since G' is a tree, there is a unique path between any two head variables in q'. Hence, q' is a dpp-transform. Suppose, there is a different dpp-transform q'' with fewer free variables than in q', then we can construct a small subtree G'' that spans the terminal nodes R by applying the above reduction to q''.

Since the Steiner Tree problem is NP-complete [10], the problem of computing an optimal dpp-transform is NP-hard as well. □

Although the problem of computing an optimal reformulation of a conjunctive query is NP-hard, all is not lost. The running time of our reformulation algorithm from Section 6 is exponential in the size of the query but is independent of the size of the database instance which, in practice, is typically much larger than the query size.

9 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 [15]. 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 [7, 8]). (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] Serge Abiteboul and Oliver M. Duschka. 1998. 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*. 254–263. http://doi.acm.org/10.1145/275487.275516

[2] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. 1997. 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.* 156–165. http://www.vldb.org/conf/1997/P156.PDF

[3] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*. 77–90. http://doi.acm.org/10.1145/800105.803397

[4] Rada Chirkova. 2002. 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*. 159–168. http://doi.acm.org/10.1145/543613.543634

[5] Rada Chirkova and Michael R. Genesereth. 2000. Linearly Bounded Reformulations of Conjunctive Databases. In *Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*. 987–1001. https://doi.org/10.1007/3-540-44957-4_66

[6] Rada Chirkova and Michael R. Genesereth. 2000. 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*. 144–163. https://doi.org/10.1007/3-540-44914-0_9

[7] Rada Chirkova and Michael R. Genesereth. 2005. Database Reformulation with Integrity Constraints (extended abstract). *CoRR* abs/cs/0506026 (2005). arXiv:cs/0506026 http://arxiv.org/abs/cs/0506026

[8] Rada Chirkova and Michael R. Genesereth. 2009. Equivalence of SQL queries in presence of embedded dependencies. In *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*. 217–226. http://doi.acm.org/10.1145/1559795.1559829

[9] Rada Chirkova, Alon Y. Halevy, and Dan Suciu. 2001. 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*. 59–68. http://www.vldb.org/conf/2001/P059.pdf

[10] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[11] Himanshu Gupta. 1997. Selection of Views to Materialize in a Data Warehouse. In *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*. 98–112. https://doi.org/10.1007/3-540-62222-5_39

[12] Himanshu Gupta and Inderpal Singh Mumick. 1999. 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*. 453–470. https://doi.org/10.1007/3-540-49257-7_28

[13] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. 1995. 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*. 95–104. http://doi.acm.org/10.1145/212433.220198

[14] Kenneth A. Ross, Divesh Srivastava, and S. Sudarshan. 1996. 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*. 447–458. http://doi.acm.org/10.1145/233269.233361

[15] Yehoshua Sagiv and Mihalis Yannakakis. 1980. Equivalences Among Relational Expressions with the Union and Difference Operators. *J. ACM* 27, 4 (1980), 633–655. http://doi.acm.org/10.1145/322217.322221