

Improving Query Evaluation with Approximate Functional Dependency Based Decompositions

Chris M. Giannella, Mehmet M. Dalkilic, Dennis P. Groth,
Edward L. Robertson *

Department of Computer Science, Indiana University
Bloomington, IN 47405 USA

{cgiannel, dalkilic, dgroth, edrbtsn}@cs.indiana.edu

Abstract

We investigate how relational restructuring may be used to improve query performance. Our approach parallels recent research extending semantic query optimization (SQO), which uses knowledge about the instance to achieve more efficient query processing. Our approach differs, however, in that the instance does not govern whether the optimization may be applied; rather, the instance governs whether the optimization yields more efficient query processing. It also differs in that it involves an explicit decomposition of the relation instance. We use approximate functional dependencies as the conceptual basis for this decomposition and develop query rewriting techniques to exploit it. We present experimental results leading to a characterization of a well-defined class of queries for which improved processing time is observed.

Keywords: query processing, approximate functional dependency, relational decomposition, query rewriting.

1 Introduction

A powerful feature of relational query languages is that identities of relational algebra may be used to transform query expressions to enhance efficiency of evaluation. Some transformations are always valid, but whether they enhance or degrade efficiency depends upon characteristics of the data. Other transformations are such that their very validity depends on characteristics of the instance. This paper takes a characteristic of the second sort, namely functional dependency (FD), recasts it to a characteristic of the first sort, and investigates resulting implications on query evaluation. For a certain set of queries, we characterize certain cases, including some surprising ones, that yield substantial decreases in query execution time. A wider range of cases is covered in [5]. Thus, this work is suggestive rather than definitive, in that it indicates the value of an approach rather than providing a full characterization of applicable results.

As noted, an FD is a characteristic of an instance. FDs used in database design hold because a declared constraint is enforced, while query optimization may also use “discovered” FDs. The difficulty with using discovered functional dependencies is that they are brittle, so that a minor change to the instance may mean that a dependency no longer holds. Previous work (described below) has addressed this difficulty by trying to handle the situations in which a functional dependency may break. Our work, on the other hand, uses a more supple notion, approximate functional

*All authors were supported by NSF Grant IIS-0082407

dependency (AFD), which bends but does not break as the instance changes. The notion of AFD applies to any instance, parameterized only by “degree of approximation.” To our knowledge, this represents the first use of AFDs in query optimization. AFDs are so ubiquitous in this work that explicit mention of them disappears from discussion, replaced by a relational decomposition (details given later) which, among other things, captures the “degree of approximation” in a simple manner.

Query optimization as considered here involves modifying a query such that semantics is preserved but performance is enhanced. The first step is to replace the name of the decomposed relation by an expression that recovers the table from the decomposition. One would hope that off-the-shelf query evaluators could optimize the rewritten query, but unfortunately our experiments have failed to bear this out. Thus we have defined query rewrite rules that apply specifically to HV decomposed relations. As with other query rewriting, application of these rules may be blocked in a particular query, just as the standard “push down selects” rules is blocked when the select condition spans both branches of a join.

In order to appraise the decomposition and rewriting rules, we performed a number of experiments using synthetic and real-world data. Synthetic data allowed control of the “degree of approximation” of the AFD being tested. Real-world data was used to validate the synthetic data generation model. These experiments showed that rewriting yielded substantial performance improvements (on queries where they were not blocked, of course). Most surprisingly, these improvements occurred not only in cases where the AFD was quite close to an FD but also in cases that were far from an FD.

We envision our approach as only the beginning of a larger investigation of similar restructurings and rewritings. There are two different features that may vary in this investigation: classes of queries where rewritings may apply or be blocked and characteristics of instances that may suggest different restructurings. For example, multi-valued or approximate multi-valued decompositions are likely candidates.

The remainder of the paper is as follows. The next section provides a brief overview of the work leading up to our approach. Section 3 presents basic definitions and provides the theoretical background and results for the paper. Included in this are examples of two different query rewriting techniques. The results of experiments designed to test the effectiveness of the rewriting techniques are presented and discussed in Section 4. The paper ends with conclusions and future work.

2 Previous work: SQO and dependencies

Two separate trains of research led toward the work reported in this paper: a long-running and substantial effort in query optimization and a more recent interest in AFDs.

Semantic query optimization (SQO) began some two decades ago and was discovered independently by King[14, 15] and Hammer *et al.*[9]. SQO is the use of “semantic knowledge for optimizing queries...”[3, 12, 33]. Some researchers proposed that a query be rewritten prior to being passed to the query engine. The query is rewritten (to an equivalent query) according to a set of rewrite rules with the idea that the rewritten query will execute faster. One such rule allows sub-queries to be merged into the outer block (thereby eliminating the sub-query). The use of rewrite rules in this fashion has been implemented in Starburst and IBM DB2 ([27, 28]).

Some researchers used additional information such as integrity constraints (ICs, *e.g.* a primary key)[29] to rewrite queries [24, 25, 30, 32, 33, 34]. For example, Paulley and Larson [24, 25] rewrite to eliminate unnecessary group by and distinct operations.

Fundamentally different from this use of declared constraints is that of discovering information

about the *instance* itself that can be used in SQO. For example, several researchers have incorporated rules discovered from the data to rewrite queries [1, 10, 31, 36]. For example, Bell [1] uses discovered FDs to eliminate group by and distinct operations. Recently, work by Godfrey *et al.*[6, 7] demonstrates that instance knowledge yields significant, positive results in SQO. They use the concept of a soft constraint (SC), which reflects knowledge about the state of the database. Thus, SCs are weaker than traditional ICs in that they do not impose any conditions upon the database. The role they play, then, is not to ensure integrity, but to “semantically characterize the database”[7]. Godfrey *et al.* introduce two classes of SCs, absolute soft constraints (ASC) and statistical soft constraints¹ (SSC). ASCs hold completely and absolutely during the current state of the database. In contrast, SSCs do not hold completely. An obvious advantage of an ASC is that, when it holds, it can be incorporated in SQO, since, for the time it holds true, it functions essentially like an IC. ASCs can be applied to SQO for the purposes of: 1. query rewriting; 2. query plan parameterization; and 3. cardinality estimation. An advantage of an SSC is that it need not be checked against every update to verify whether it holds—rather, every so often the SSCs must be brought up to date. While Godfrey *et al.* shows how SSCs can be useful for cardinality estimation, they do not show how SSCs can be used for query rewriting.

Godfrey *et al.* then describe how SCs would be incorporated into an RDBMS (since no current system is available): 1. discovery, 2. selection, 3. maintenance. For ASCs, Godfrey *et al.* focus on both checking when an ASC is violated and maintaining ASCs. Because of their tenuous nature *i.e.*, being state-dependent, considerable care must be given to both checking and maintaining ASCs – a difficult task. This is, in fact, the “Achilles heel” of ASCs. A natural question arises from the work of Godfrey *et al.*: how can SSCs be used, if at all, for query rewriting?

While the body of work deriving from SQO is substantial, a second, more recent body of work concerning AFDs was even more significant in the genesis of this paper. The notion of a functional dependency was originally introduced as an IC for use in database design. However, more recently, research has been conducted with the view point that FDs represent interesting patterns existent in the data. In this setting, FDs are not regarded as declared constraints. Researchers have investigated the problem of efficiently discovering FDs that hold in a given instance [11, 13, 16, 18, 20, 21, 23, 35]. Researchers have also considered the concept of an FD “approximately holding” in an instance and have developed measures to characterize the “degree of approximation” [2, 4, 8, 16, 17, 19, 22, 26]. The measure proposed by Kivinen and Mannila [16], g_3 , correlates with the idea of “correction” that we use. Huhtala *et al.* [11] develop an algorithm for efficiently discovering all AFDs in a given instance whose g_3 approximation measure is below a user specified threshold.

3 Definitions and Theoretical Results

In this section we describe the theoretical results that form the basis of our work. We describe two query rewriting techniques. The first technique is guaranteed to always preserve correctness. The second technique is guaranteed to preserve correctness only on a special class of queries described later. Then, we describe two hypotheses about how our rewriting techniques affect query evaluation time.

We do not give proofs in this paper. Rather, we describe intuitively why the results work and illustrate with examples. Rigorous proofs can be given, but, in our opinion, do not illuminate the ideas.

¹The term “statistical” is meant to connote that the soft constraint holds true for some, if not all, of the data and not that any probabilistic techniques are used.

3.1 Basic Notation

We assume the reader is familiar with the basic concepts of relational database theory (see [29] for a review). In what follows, we fix a relation symbol R with schema $\{A, B, C\}$. Our results can easily be generalized to apply to schema with any number of attributes, however, for simplicity, we stick with $\{A, B, C\}$. Since tables, in practice, may contain repeats, we develop our theoretical foundations with bags. Whenever we write “relation instance” or “instance” or “relation” we mean a bag and not necessarily a set.

We also make the distinction between those relational algebra (RA) operators that return sets and those that return bags. The ones that return sets (*i.e.* the ones that are duplicate removing) are denoted in the standard way: Π , σ , \bowtie , δ , \cup , and $-$ (projection, selection, natural join, renaming, union, and minus, respectively). Their bag counterparts are denoted: $\hat{\Pi}$, $\hat{\sigma}$, $\hat{\bowtie}$, $\hat{\delta}$, $\hat{\cup}$, and $\hat{-}$. We call the relational algebra over all operators (bag and set) the *bag relational algebra (bag RA)*.

Let $\mathbf{s}_1, \mathbf{s}_2$ be instances over some schema S . We say that \mathbf{s}_1 and \mathbf{s}_2 are *set equivalent*, written $\mathbf{s}_1 \equiv \mathbf{s}_2$ if $\Pi_S(\mathbf{s}_1) = \Pi_S(\mathbf{s}_2)$. In other words, \mathbf{s}_1 and \mathbf{s}_2 are equal once duplicates have been removed. Given, Q , a bag RA expression involving R , and E , another bag RA expression, let $Q[R \leftarrow E]$ be the result of replacing all occurrences of R by E . Note that the result may contain schema conflicts.

Example 1 Let $Q := \Pi_{A,B}(\sigma_{C=0}(R))$. Let $E := R_1 \hat{\cup} \hat{\Pi}_{A,B}(R_2)$ where R_1 has schema $\{A, B\}$ and R_2 has schema $\{A, B, C\}$. Note that E has no schema conflicts. The schema of E is $\{A, B\}$. By definition $Q[R \leftarrow E]$ is $\Pi_{A,B}(\sigma_{C=0}(E))$. This expression has a schema conflict since $\sigma_{C=0}$ is being applied to E which has schema $\{A, B\}$.

Consider another example. Let $Q' := \Pi_{A,B}(R \cup S)$ (S has the same schema as R , $\{A, B, C\}$). By definition $Q'[R \leftarrow E]$ is $\Pi_{A,B}(E \cup S)$. This expression has a schema conflict since E has schema $\{A, B\}$ while S has schema $\{A, B, C\}$. \square

3.2 Horizontal-Vertical Decompositions

Let \mathbf{r} be an instance of R . If the functional dependency $A \rightarrow B$ holds in \mathbf{r} , then \mathbf{r} may be decomposed vertically as $\mathbf{r}_{AB} = \Pi_{A,B}(\mathbf{r})$, $\mathbf{r}_{AC} = \hat{\Pi}_{A,C}(\mathbf{r})$. This decomposition enjoys the property of being join lossless: $\mathbf{r} = \mathbf{r}_{AB} \hat{\bowtie} \mathbf{r}_{AC}$.

If $A \rightarrow B$ does not hold in \mathbf{r} , then \mathbf{r} may still be decomposed. But we must first horizontally decompose \mathbf{r} into two disjoint, non-empty relation instances whose union is \mathbf{r} , such that $A \rightarrow B$ holds in one of these relation instances. This instance is then vertically decomposed. The result is a *horizontal-vertical decomposition* of \mathbf{r} . The next set of definitions makes this concept precise.

$\mathbf{r}^c \subseteq \mathbf{r}$ is said to be an $A \rightarrow B$ *correction* for \mathbf{r} if $A \rightarrow B$ holds in $\mathbf{r} \hat{-} \mathbf{r}^c$. Often we omit mention of $A \rightarrow B$ and \mathbf{r} when clear from context and only say that \mathbf{r}^c is a correction. Given correction, \mathbf{r}^c , the horizontal-vertical decomposition (HV decomposition) induced is \mathbf{r}'_{AB} , \mathbf{r}'_{AC} , and \mathbf{r}^c where \mathbf{r}' is $\mathbf{r} \hat{-} \mathbf{r}^c$, \mathbf{r}'_{AB} is $\Pi_{A,B}(\mathbf{r}')$, and \mathbf{r}'_{AC} is $\hat{\Pi}_{A,C}(\mathbf{r}')$. As noted, this corresponds to the g_3 measure of [16].

Consider the instance, \mathbf{s} , in Figure 1. Clearly, $A \rightarrow B$ does not hold. Let $\mathbf{s}^c = \{(1, 3, 1), (2, 1, 1)\}$. Since $A \rightarrow B$ holds in $\mathbf{s}' = \{(1, 2, 1), (2, 1, 1)\}$, then \mathbf{s}^c is a correction. The HV decomposition induced is depicted in Figure 2. Notice that $\mathbf{s} = (\mathbf{s}'_{AB} \hat{\bowtie} \mathbf{s}'_{AC}) \hat{\cup} \mathbf{s}^c$. Moreover, in this case, $\hat{\Pi}_{A,B}(\mathbf{s}) = \mathbf{s}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{s}^c)$. The first observation points to the lossless property of HV decompositions. The second observation points to another property that will later be shown important: if C is not needed in a query involving \mathbf{s} , then the join can be eliminated from the decomposition.

In the previous example, \mathbf{s}^c is not minimal since a smaller correction can be found. For example, $\mathbf{s}^c = \{(1, 3, 1)\}$ is also a correction. The HV decomposition induced is depicted in Figure 3. Notice that $\mathbf{s} = (\mathbf{s}'_{AB} \hat{\bowtie} \mathbf{s}'_{AC}) \hat{\cup} \mathbf{s}^c$. Moreover, $\hat{\Pi}_{A,B}(\mathbf{s}) \equiv \mathbf{s}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{s}^c)$ (but equality does not hold). As

A	B	C
1	2	1
1	3	1
2	1	1
2	1	1

Figure 1: An instance, \mathbf{s} , over schema $\{A, B, C\}$.

\mathbf{s}^c			\mathbf{s}'_{AB}		\mathbf{s}'_{AC}	
A	B	C	A	B	A	C
1	3	1	1	2	1	1
2	1	1	2	1	2	1

Figure 2: Induced HV non-minimal decomposition, \mathbf{s}^c , \mathbf{s}'_{AB} , \mathbf{s}'_{AC} .

in the previous example, the first observation points to the lossless property of the decomposition, and the second observation points to the property that if C is not needed in queries involving \mathbf{s} , then the join can be eliminated from the decomposition. However, this example shows that the property must be weakened to set equality rather than equality (see Theorem 1).

The point of these last two examples was (i) to illustrate two important properties of HV decompositions, and (ii) to point out that these properties do not depend on the correction being minimal. Our query rewriting techniques rely on these properties. If these properties were dependent on the decomposition being minimal, then maintaining the decomposition in the presence of updates would be difficult. But, these properties are not dependent on the correction being minimal. Hence, we gain much greater maintenance flexibility. Nonetheless, maintenance is still a difficult issue. Due to space constraints, we defer description of a simple method for maintaining the decomposition to an extended version of this paper [5]. The performance of the maintenance method is not analyzed in [5]; this is left as future work.

In short, the fundamental properties of HV decompositions needed for our query rewriting techniques are the following.

Theorem 1

1. $\mathbf{r} = (\mathbf{r}'_{AB} \hat{\bowtie} \mathbf{r}'_{AC}) \hat{\cup} \mathbf{r}^c$.
2. $\hat{\Pi}_{A,B}(\mathbf{r}) \equiv \mathbf{r}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{r}^c)$.

In part 2, if $\hat{\cup}$ were replaced by \cup , then the result could be strengthened to equality ($=$) rather than set equality (\equiv). However, the merit of stating part 2 as done above will be seen later when we discuss our second rewriting technique. In that setting, it will not matter for correctness whether

\mathbf{s}^c			\mathbf{s}'_{AB}		\mathbf{s}'_{AC}	
A	B	C	A	B	A	C
1	3	1	1	2	1	1
			2	1	2	1
					2	1

Figure 3: Induced HV minimal decomposition, \mathbf{s}^c , \mathbf{s}'_{AB} , \mathbf{s}'_{AC} .

we use equality or set equality, but for efficiency, not removing duplicates in the union will be an advantage.

3.3 Query Rewriting: Technique I

HV decompositions, in a sense, “expose” structural information about the instance. Our basic idea is to rewrite queries using the decomposition such that structural information is exposed to the DBMS query evaluator. Our thinking is that the optimizer could use this structure to improve query evaluation. Theorem 1, part 1 provides the foundation of our first rewriting technique (illustrated by the following example). Consider an example query, Q :

```
Select Distinct R1.A, R1.B
From R as R1
Where R1.A = 0.
```

Expressed in bag RA terms, $Q := \Pi_{A,B}(\sigma_{A=0}(R))$. If the HV decomposition \mathbf{r}'_{AB} , \mathbf{r}'_{AC} , \mathbf{r}^c is kept in the database, then, by Theorem 1, part 1, we have $Q(\mathbf{r}) = \Pi_{A,B}(\sigma_{A=0}((\mathbf{r}'_{AB} \bowtie \mathbf{r}'_{AC}) \hat{\cup} \mathbf{r}^c))$. So, Q can be rewritten as Q_1 :

```
Select Distinct R1.A, R1.B
From ((Select RAB.A as A, RAB.B as B, RAC.C as C
      From RAB, RAC
      Where RAB.A=RAC.A)
      Union All
      (Select A,B,C
      From Rc)) as R1
Where R1.A = 0.
```

This technique of query rewriting preserves correctness on any SQL query, *i.e.* the rewritten query is well-formed (no schema conflicts) and, when handed to the DBMS query evaluator, produces exactly the same result as the rewritten query. If R occurs more than once (*e.g.* “ R as R_2 ”), then each occurrence of R is replaced as above. We call this *Rewriting Technique I*.

3.4 Query Rewriting: Technique II

In the previous subsection, Q was rewritten as Q_1 . However, Q has properties that allow further rewriting. First observe that attribute C does not appear in the output schema of Q and is not used elsewhere in the query. As a result, only the attributes A and B are needed; C can be projected out: $Q(\mathbf{r}) = Q(\hat{\Pi}_{A,B}(\mathbf{r}))$. Hence, we have $Q(\mathbf{r}) = \Pi_{A,B}(\sigma_{A=0}(\hat{\Pi}_{A,B}((\mathbf{r}'_{AB} \bowtie \mathbf{r}'_{AC}) \hat{\cup} \mathbf{r}^c)))$.

Now by Theorem 1 part 2, we have $Q(\mathbf{r}) \equiv \Pi_{A,B}(\sigma_{A=0}(\mathbf{r}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{r}^c)))$. But since $\Pi_{A,B}$ appears at the top level of Q (hence duplicates are removed from the output), then we may replace set equality by equality: $Q(\mathbf{r}) = \Pi_{A,B}(\sigma_{A=0}(\mathbf{r}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{r}^c)))$. So, Q may be rewritten as Q_2 :

```
Select Distinct R1.A, R1.B
From ((Select RAB.A as A, RAB.B as B
      From RAB)
      Union All
      (Select Rc.A as A, Rc.B as B
      From Rc)) as R1
Where R1.A = 0.
```

The decomposition join has been eliminated and we expect that Q_2 , when handed to the DBMS query evaluator, will evaluate faster than Q_1 . Moreover, we would expect that Q_2 will evaluate faster than Q if \mathbf{r}'_{AB} and \mathbf{r}^c are not large relative to \mathbf{r} . If R occurs more than once in Q , then replace each occurrence as above. We call this *Technique II*.

3.4.1 Application of Technique II

In the previous subsection, Q was rewritten as Q_1 and then further rewritten as Q_2 . While rewriting as Q_1 (Technique I) always preserves correctness, rewriting as Q_2 (Technique II) does not. We would like to isolate syntactic properties of Q that guarantee that Technique II preserves correctness.

Since Technique II replaces R by an expression whose schema is $\{A, B\}$, then the original query cannot involve C or else the rewritten query may have schema conflicts. The top bag RA expression in Example 1 illustrates how such a schema conflict can arise. Moreover, if the original query involves union or minus, then schema conflicts can also arise. The bottom RA expression in Example 1 illustrates how schema conflicts can arise in the presence of union. However, even when the original query does not involve C , the rewritten query may not produce the same output (but is set equivalent to the original). Consider the following example query, which is the same as query Q in subsection 3.3 except that the “Distinct” has been removed.

```
Select R1.A, R1.B
From R as R1
Where R1.A = 0.
```

When applying Technique II to this query, the rewritten query may not be equivalent because of duplicates in the output. For a similar reason, aggregates cannot be supported by Technique II.

We give a result (Corollary 1) that defines a general class of queries over which Technique II is guaranteed to preserve correctness. First, though, we state a theorem that defines a general class of bag RA expressions over which Technique II is guaranteed to preserve correctness *up to set equivalence*. Corollary 1 falls out immediately by restricting the class further to those which have Π at their top level.

Let Q be any bag RA expression. Let E be the bag RA expression $R'_{AB} \hat{\cup} \hat{\Pi}_{AB}(R^c)$ where R'_{AB} is a relation symbol over schema $\{A, B\}$ and R^c is a relation symbol over schema $\{A, B, C\}$. Let Q_2 be $Q(R \leftarrow E)$.

Theorem 2 *If Q does not involve union or minus, and the output schema of Q does not contain C , and C does not appear in Q , then Q_2 is well-defined and $Q(\mathbf{r}) \equiv Q_2(\mathbf{r}'_{AB}, \hat{\Pi}_{A,B}(\mathbf{r}^c))$.*

We say that a bag RA expression Q is *top-level distinct* if it is of the form $\Pi_{\dots}(Q')$. A top-level distinct expression always returns a set.

Definition 1 *We say that a bag RA expression, Q , is join elimination re-writable (JE-rewritable) if (i) Q is top-level distinct, (ii) Q does not involve union or minus, (iii) The output schema of Q does not contain C , and (iv) C does not appear in Q (i.e. is not part of any projection, selection, or renaming condition, and no join has C among its join attributes).*

Take note that the first example in subsection 3.3 is the SQL of the JE-rewritable, bag RA expression $\Pi_{A,B}(\sigma_{A=0}(\mathbf{r}))$. Theorem 2 implies the following result that defines the class of bag RA expressions over which Technique II preserves correctness.

Corollary 1 *If Q is JE-rewritable, then Q_2 is well-formed and $Q(\mathbf{r}) = Q_2(\mathbf{r}'_{AB}, \hat{\Pi}_{A,B}(\mathbf{r}^c))$.*

The SQL queries equivalent to the JE-rewritable, bag RA expressions are the queries on which Technique II is guaranteed to preserve correctness. We call these *JE-rewritable queries*.

3.5 Hypotheses

We have developed two query rewriting techniques: $Q \rightarrow Q_1$ (Technique I), $Q \rightarrow Q_2$ (Technique II). The first technique is guaranteed to preserve correctness for any query. The second is only guaranteed to preserve correctness for JE-rewritable queries. Q_1 may evaluate faster than Q when handed to the DBMS query evaluator, because Q_1 exposes more of the structure of \mathbf{r} , thereby allowing the optimizer to possibly take advantage of this structure. We arrive at our first hypothesis. Let $time(Q)$ denote the time required by the DBMS query evaluator to evaluate Q (likewise, define $time(Q_1)$).

Hypothesis 1 (Rewriting Technique I) *Given query Q , $time(Q_1) < time(Q)$.*

We expect Q_2 to evaluate faster than Q_1 because of the elimination of the join in the HV decomposition. Moreover, we expect Q_2 to evaluate faster than Q when \mathbf{r}'_{AB} and \mathbf{r}^c are small relative to \mathbf{r} . Let $|\mathbf{r}|$, $|\mathbf{r}'_{AB}|$, and $|\mathbf{r}^c|$ denote the number of tuples in \mathbf{r} , \mathbf{r}'_{AB} , and \mathbf{r}^c , respectively. Let $adom(A, \mathbf{r}'_{AB})$ denote the active domain of A in \mathbf{r}'_{AB} (likewise define $adom(A, \mathbf{r})$). By definition of HV decompositions, $|\mathbf{r}'_{AB}| = |adom(A, \mathbf{r}'_{AB})|$. We use $\frac{|\mathbf{r}|}{|adom(A, \mathbf{r}'_{AB})| + |\mathbf{r}^c|}$ to quantify the size of \mathbf{r}'_{AB} and \mathbf{r}^c relative to \mathbf{r} . For example, if $|adom(A, \mathbf{r}'_{AB})|$ is two percent of $|\mathbf{r}|$ and $|\mathbf{r}^c|$ is twenty percent of $|\mathbf{r}|$, then $|\mathbf{r}|$ is 4.55 times as large as $|\mathbf{r}'_{AB}| + |\mathbf{r}^c|$.

Hypothesis 2 (Rewriting Technique II) *Given JE-rewritable query Q , if $\frac{|\mathbf{r}|}{|adom(A, \mathbf{r}'_{AB})| + |\mathbf{r}^c|} > 1$, then $time(Q_2) < time(Q)$. Moreover, as $\frac{|\mathbf{r}|}{|adom(A, \mathbf{r}'_{AB})| + |\mathbf{r}^c|}$ increases, then $time(Q) - time(Q_2)$ also increases.*

4 Experimental Results

Datasets for the experiments were generated randomly, controlling for the size of the relation, the size of the correction, and the size of the active domains for A and B . The order of tuples was permuted to avoid long sequences of tuples in the dataset with the same A and B value. The table used for the join query was generated from \mathbf{r} by projecting out the unique B values and adding a description field, which resulted in the schema $S = \{B, D\}$.

Fixing the size of the active domains provides the benefit of controlling for the selectivity of the queries that select rows based on a constant. The constant used for these queries was the value representing the median frequency. We fixed the size of $adom(B, \mathbf{r})$ to be 100 for each experiment. We considered three sizes for $adom(A, \mathbf{r})$: 100, 1000, and 10000. The results for each of these sizes were similar, so we report the case where $adom(A, \mathbf{r}) = 1000$. The other sizes, as well as a more thorough description of the synthetic data generation procedure, are reported in [5].

For all experiments, the size of the relation was 500,000 tuples. The size of the correction ranged from 0–90% in 10% increments. Note that the 0% correction represents a case where the functional dependency $A \rightarrow B$ holds. The decompositions generated for the experiments were minimal, so $|adom(A, \mathbf{r})| = |adom(A, \mathbf{r}'_{AB})|$ in each case.

The datasets were stored as tables in an Oracle (Version 8.05) database running on Sun UltraSparc 10 server running Solaris 7 equipped with 256 MB of RAM. No indexes were generated for any of the tables. However, statistics were generated for all tables, allowing for cost-based optimization.

Queries were executed from a Java 1.3 application using the JDBC thin client interface provided by Oracle. Each query was executed 5 times, with the mean completion time to return the final row in the result reported. The standard deviations we observed were small and are omitted from this report. We used the “NOCACHE” optimizer directive on each query to avoid reusing the cache.²

4.1 Testing Hypotheses

We tested Hypothesis 1 on the following query:

```
Select Distinct R1.A,R1.B,R1.C
From R as R1
Where R1.A=constant
```

rewritten using Technique I as:

```
Select Distinct R1.A,R1.B,R1.C
From (Select RAB.A,RAB.B,RAC.C
      From RAB,RAC Where RAB.A=RAC.A
      Union All
      Select A,B,C From Rc) as R1
Where R1.A=constant
```

The results show that the rewritten query performs worse than the original query (due to space constraints, detailed results are omitted, see [5]). Consequently, it does not appear that exposing the structure to the optimizer yielded any benefits. Interestingly, the worst performance occurs in the case of a perfect functional dependency. We conclude that our hypothesis is incorrect. We believe that the reason for failure is directly related to the join in the rewritten query. A closer examination of the query plans makes clear why we did not experience an improvement. The plan generated for the rewritten query did not push the selects into the query. Instead, the original relation was materialized and then scanned to get the answer.

Rewriting Technique II, when applicable, does not produce queries requiring the join. As a result, we expect that the performance of queries rewritten with Technique II will be clearly faster than those rewritten with Technique I. Now we test whether queries rewritten with Technique II are faster than the original queries.

To test hypothesis 2, the following four JE-rewritable queries were used.

1. `Select Distinct R1.A,R1.B From R as R1 Where R1.A=constant`
2. `Select Distinct R1.A,R1.B From R as R1 Where R1.B=constant`
3. `Select Distinct R1.A,R1.B,S.D From R as R1,S Where R1.B=S.B`
4. `Select Distinct R1.A,R1.B From R as R1`

Each query was rewritten as follows:

²This directive specifies that blocks are placed in the least recently used part of the LRU list in the buffer cache when a full table scan is performed. This was necessary when repeating a query to avoid misleading results.

1. `Select Distinct R1.A,R1.B`
`From (Select A,B From RAB Union All Select A,B From Rc) as R1`
`Where R1.A=constant`
2. `Select Distinct R1.A,R1.B`
`From (Select A,B From RAB Union All Select A,B From Rc) as R1`
`Where R1.B=constant`
3. `Select Distinct R1.A,R1.B,S.D`
`From (Select A,B From RAB Union All Select A,B From Rc) as R1, S`
`Where R1.B=S.B`
4. `Select Distinct R1.A,R1.B`
`From (Select A,B From RAB Union All Select A,B From Rc) as R1`

Figure 4 shows the results for Queries 1, 2, and 3. For Queries 1 and 2, we see that the rewritten queries perform better than the original query when the size of the correction is small. As the size of the correction increases, the performance of the rewritten queries degrades in a linear fashion. Queries 3 and 4 exhibited similar behavior. See the extended version of the paper for details.

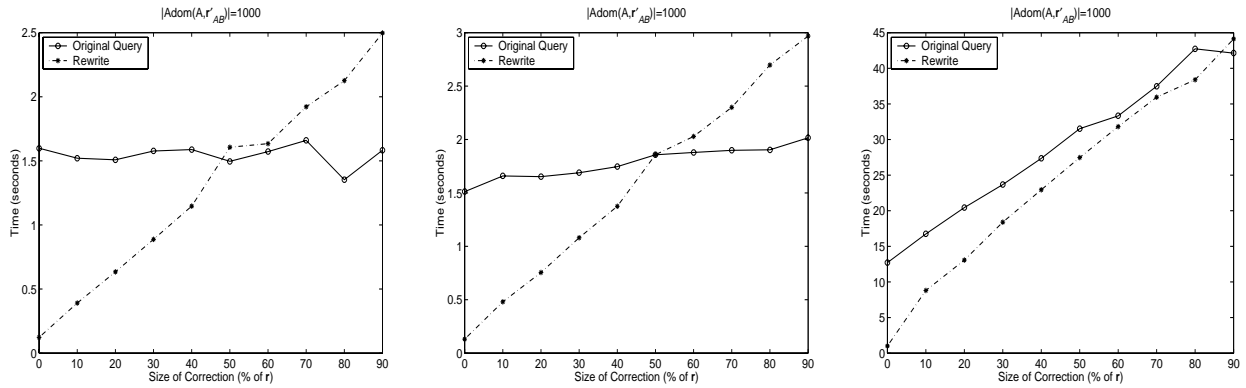


Figure 4: Comparing the mean execution time for Queries 1, 2, and 3 to the size of the correction.

4.2 Discussion

Hypothesis 1 was incorrect due to the cost of the join introduced by the decomposition and the fact that the optimizer did not take advantage of the decomposition. For example, as shown in the queries below, the optimizer could have pushed selects into the decomposition. The first query is the original query rewritten with Technique I used to test hypothesis 1. The second query exploits the structure of the decomposition by pushing selects as deep as possible, which is one of the most basic query optimization strategies. In these experiments the query optimizer did not take advantage of the decomposition.

1. `Select Distinct R1.A,R1.B,R1.C`
`From (Select RAB.A,RAB.B,RAC.C From RAB,RAC Where RAB.A=RAC.A`
`Union All Select A,B,C From Rc) as R1`
`Where R1.A=constant`

```

2. Select Distinct R1.A,R1.B,R1.C
   From (Select RAB.A,RAB.B,RAC.C From RAB,RAC
        Where RAB.A=constant and RAC.A=constant and RAB.A=RAC.A
        Union All Select A,B,C From Rc Where A=constant) as R1

```

Queries that were able to use Technique II outperformed the original queries until the size of the correction exceeded 50% - a certainly robust technique. We were pleasantly surprised that the breaking point was so high. After all, it seems intuitive that the decomposition will perform better when the AFD is close to an FD. It is surprising though, that the AFD $A \rightarrow B$ in the original relation can be significantly far from being an FD and result in better performing queries. Observe that for our experimental datasets, the size of the active domain does not materially affect the relative performance of the queries.

In addition to experiments against synthetic data, we tested Technique II on U.S. Census data (see [5] for details). These results demonstrate that the technique was beneficial when applied to real-world data. In fact, the rewritten queries performed better against the census data than against the synthetic data. However, more experiments against real data are necessary to determine the general applicability of the technique.

5 Conclusion and Future Work

In this paper we investigated an approach (paralleling recent work extending SQO) to improve query evaluation. Our approach is based on decomposing relation instances with respect to AFDs and rewriting queries to take advantage of these decompositions (HV decompositions). The primary idea is that the semantic information contained in an AFD can be exposed by creating an HV decomposition of the relation instance. This information can then be exploited to speed up query evaluation: rewrite the query to use the decomposition instead of the original relation, then, issue the rewritten query to the DBMS query engine instead of the original query. Two things in particular should be pointed out about how our approach fits into the literature. First, it represents (to our knowledge) the first use of AFDs in query evaluation. Second, it addresses the question raised by the work of Godfrey *et al.* described in Section 2. Our primary motivation was not to specifically address the question. We discovered after we had obtained our results that they could be used to address the question.

We investigated two rewriting techniques. Technique I replaces all occurrences of the relation symbol by its decomposition. This technique is guaranteed to preserve correctness on all SQL queries. The motivation was that the optimizer can take advantage of the decomposition and produce a more efficient plan. Our experiments, however, point out that this was not the case. The introduction of the decomposition join caused the rewritten queries to run more slowly. Technique II replaces all occurrences of the relation symbol by the decomposition without the join. This technique is only guaranteed to preserve correctness on a special class of queries (JE-rewritable queries). However, our experiments show that queries rewritten with this technique tend to evaluate significantly faster than the original query, provided the correction was not too large.

Our results suggest an architecture designed around AFD-based decompositions. We have observed that Technique II can offer significant speed-up, but, it can only be applied to JE-rewritable queries. So, the original relation should be kept along with its HV decomposition. The preprocessor examines the query to see if it is JE-rewritable. If so, and if the correction is not too large, then the query is rewritten. Otherwise, the query is not rewritten and the original query is handed to the DBMS query engine. See Figure 5.

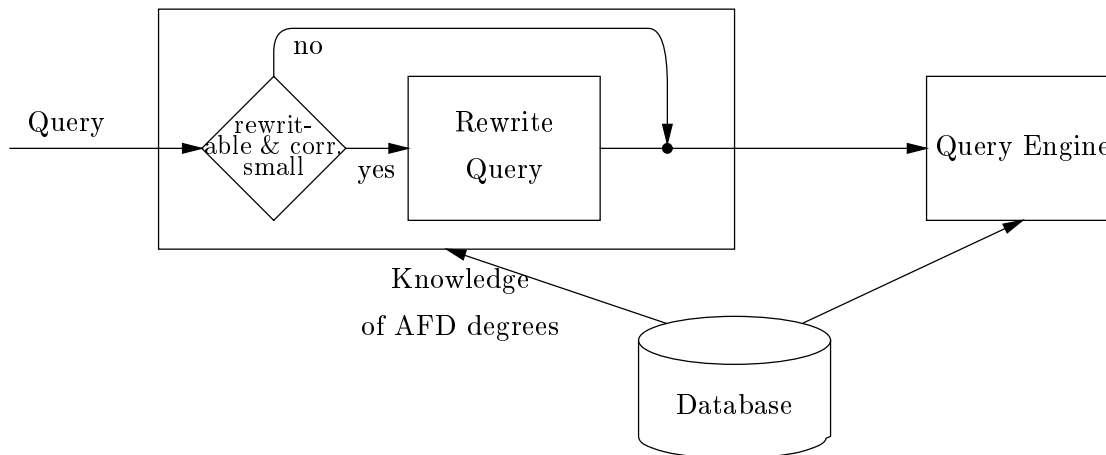


Figure 5: The preprocessor rewrites JE-rewritable queries provided that the correction is not too large.

There are a number of directions for future work. 1. Address the primary drawback to Technique II: it is guaranteed to preserve correctness only on JE-rewritable queries. In particular, modify our approach to handle aggregate operations like count and sum. This can be achieved by modifying the construction of \mathbf{r}'_{AB} to keep counts. The significant issue is then sharpened: how to rewrite queries to use the counts. 2. Investigate other rewritings in the AFD context. For example, an FD can allow elimination of outer join and group by operations. 3. Investigate the use of other decompositions. A natural candidate is a decomposition induced by approximate multi-valued dependencies, which creates two correction relations $\mathbf{r}^+, \mathbf{r}^-$. 4. Investigate methods for maintaining an HV decomposition. Does the extra time required for processing inserts and deletes eclipse the gains in query evaluation? Also, when should corrections be reorganized to minimize \mathbf{r}^c ? The $\mathbf{r}^+, \mathbf{r}^-$ decomposition mentioned above is intriguing because deletes are processed via \mathbf{r}^- .

In closing we point out that the primary purpose was to introduce the idea of exploiting AFDs in query evaluation via HV decompositions and demonstrate that the idea is fertile grounds for future work. We feel that we have achieved this goal.

References

- [1] BELL S. Deciding distinctiveness of query results by discovered constraints. *Proc. of the 2nd International Conf. on the Practical Application of Constraint Technology* (1996), 399–417.
- [2] CAVALLO R. AND PITTARELLI M. The theory of probabilistic databases. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)* (1987), pp. 71–81.
- [3] CHAKRAVARTHY U., GRANT J., AND MINKER J. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems* 15, 2 (June 1990), 162–207.
- [4] DALKILIC M. AND ROBERTSON E. Information dependencies. In *Proc. of the Nineteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)* (2000), ACM, pp. 245–253.
- [5] GIANNELLA C., DALKILIC M., GROTH D., AND ROBERTSON E. Using horizontal-vertical decompositions to improve query evaluation. Tech. rep., Computer Science, Indiana University, Bloomington, Indiana, USA, 2002.

- [6] GODFREY P., GRANT J., GRYZ J., AND MINKER J. *Logics for Databases and Information Systems*. Kluwer Academic Publishers, Boston, MA, 1998, pp. 265–307.
- [7] GODFREY P., GRYZ J., AND ZUZARTE C. Exploiting constraint-like data characterizations in query optimization. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data* (May 2001), pp. 582–592.
- [8] GOODMAN L. AND KRUSKAL W. Measures of associations for cross classifications. *Journal of the American Statistical Association* 49 (1954), 732–764.
- [9] HAMMER M. AND ZDONIK S. JR. Knowledge-based query processing. In *Proc. of the Sixth Intl. Conf. on Very Large Data Bases* (Montreal, Canada, Oct. 1980), pp. 137–147.
- [10] HSU C. AND KNOBLOCK C. Using inductive learning to generate rules for semantic query optimization. In *Advances in Knowledge Discovery and Data Mining* (1996), Fayyad U. and Piatetsky-Shapiro G., Ed.
- [11] HUHTALA Y., KÄRKKÄINEN J., PORKKA P., AND TOIVONEN H. Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings 14th International Conference on Data Engineering* (February 1998), IEEE Computer Society Press, pp. 392–401.
- [12] JARKE J., CLIFFORD J., AND VASSILIOU Y. An optimizing PROLOG front-end to a relational query system. In *Proc. of the ACM SIGMOD Conf.* (1984), pp. 296–306.
- [13] KANTOLA M., MANNILA H., RÄIHÄ K., AND SIIRTOLA H. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems* 7 (1992), 591–607.
- [14] KING J. QUIST—a system for semantic query optimization in relational databases. In *Proc. of the 7th International Conf. on Very Large Data Bases Cannes* (Cannes, France, Sept. 1981), IEEE Computer Society Press, pp. 510–517.
- [15] KING J. Reasoning about access to knowledge. In *Proc. of the Workshop on Data Abstraction, Databases, and Conceptual Modelling* (Jan. 1981), SIGPLAN Notices, pp. 138–140.
- [16] KIVINEN J. AND MANNILA H. Approximate dependency inference from relations. *Theoretical Computer Science* 149, 1 (1995), 129–149.
- [17] LEE T. An information-theoretic analysis of relational databases—part I: Data dependencies and information metric. *IEEE Transactions on Software Engineering SE-13*, 10 (October 1987), 1049–1061.
- [18] LOPES S., PETIT J., AND LAKHAL L. Efficient discovery of functional dependencies and Armstrong relations. In *Lecture Notes in Computer Science 1777 (first appeared in the Proceedings of the Seventh International Conference on Extending Database Technology (EDBT))* (2000), pp. 350–364.
- [19] MALVESTUTO F. Statistical treatment of the information content of a database. *Information Systems* 11, 3 (1986), 211–223.
- [20] MANNILA H. AND RÄIHÄ K. Dependency inference. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)* (1987), pp. 155–158.
- [21] MANNILA H. AND RÄIHÄ K. Algorithms for inferring functional dependencies. *Data & Knowledge Engineering* 12 (1994), 83–99.
- [22] NAMBIAR K. Some analytic tools for the design of relational database systems. In *Proceedings of the 6th International Conference on Very Large Databases (VLDB)* (1980), pp. 417–428.

- [23] NOVELLI N., CICCETTI R. FUN: an efficient algorithm for mining functional and embedded dependencies. In *Lecture Notes in Computer Science 1973 (Proceedings of the 8th International Conference on Database Theory (ICDT))* (2001), pp. 189–203.
- [24] PAULLEY G. *Exploiting Functional Dependence in Query Optimization*. PhD thesis, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Sept. 2000.
- [25] PAULLEY G. AND LARSON P. Exploiting uniqueness in query optimization. In *Proc. of the 10th ICDE* (1994), pp. 68–79.
- [26] PIATETSKY-SHAPIO G. Probabilistic data dependencies. In *Machine Discovery Workshop (Aberdeen, Scotland)* (1992).
- [27] PIRAHESH H., HELLERSTEIN J., AND HASAN W. Extensible/rule based query rewrite optimization in starburst. In *Proc. 1992 ACM-SIGMOD Int. Conf. Management of Data* (May 1992), pp. 39–48.
- [28] PIRAHESH H., LEUNG T.Y., AND HASAN W. A rule engine for query transformation in starburst and ibm db2 c/s dbms. In *Proc. 13th Int. Conf. on Data Engineering (ICDE)* (April 1997), pp. 391–400.
- [29] RAMAKRISHNAN R. AND GEHRKE J. *Database Management Systems 2nd Edition*. McGraw-Hill Higher Education, Boston, MA, 2000.
- [30] SAGIV Y. Quadratic algorithms for minimizing join in restricted relational expressions. *SIAM Journal of Computing* 12, 2 (May 1983), 316–328.
- [31] SHEKHAR S., HAMIDZADEH B., KOHLI A., AND COYLE M. Learning transformation rules for semantic query optimization: a data-driven approach. *IEEE Transactions on Knowledge and Data Engineering* 5, 6 (December 1993), 950–964.
- [32] SHENOY S. AND OZSOYGLU Z. A system for semantic query optimization. In *Proc. 1987 ACM-SIGMOD Int. Conf. Management of Data* (May 1987), pp. 181–195.
- [33] SHENOY S. AND OZSOYGLU Z. Design and implementation of a semantic query optimizer. *IEEE Transactions on Knowledge and Data Engineering* 1, 3 (Sept. 1989), 344–361.
- [34] SUN W. AND YU C. Semantic query optimization for tree and chain queries. *IEEE Transactions on Knowledge and Data Engineering* 6, 1 (February 1994), 136–151.
- [35] WYSS C., GIANNELLA C., AND ROBERTSON E. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *Lecture Notes in Computer Science 2114 (Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery)* (2001), pp. 101–110.
- [36] YU C. AND SUN W. Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Transactions on Knowledge and Data Engineering* 1, 3 (September 1989), 362–374.