WPI-CS-TR-98-2 January 1998

Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings

by

Amy J. Lee
Andreas Koeller
Anisoara Nica
Elke A. Rundensteiner

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings *

Amy J. Lee†, Andreas Koeller‡, Anisoara Nica†, and Elke A. Rundensteiner‡

(†) Department of EECS
University of Michigan, Ann Arbor
Ann Arbor, MI 48109-2122
{amylee|anica}@eecs.umich.edu

(‡)Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280
{koeller|rundenst}@cs.wpi.edu

Abstract

The problem of rewriting queries has been heavily explored in recent years, including in work on query processing and optimization, semantic query refinement in decentralized environments, the rewriting of queries using views, and view maintenance. Previous work has made the restricting assumption that the rewritten query must be *equivalent* to the initially given query. We now propose to relax this assumption to allow for query rewriting in situations where *equivalent* rewritings may not exist – yet alternate *not necessarily equivalent* query rewritings may still be preferable to users over not receiving any answers at all. Our approach is based on a preference model, an extension of SQL called E-SQL, that captures the intention of the query by how much deviation from the original query would still be acceptable to the user. In this paper, we introduce an analytical model of query rewritings that incorporates measures of quality of a query in addition to the commonly studied measures of costs (query performance). Quality is modeled as a function of the divergence from the intended view extent, both in terms of the preservation of the information amount and the information type. Both quality and cost are integrated into one uniform model, called the QC-Model , to allow for a trade-off among these two measures. This model can be used to compare two alternate (even if not equivalent) rewritings, and thus to establish a ranking among a possibly large set of query rewritings. Our model is the first to allow for automatic selection of good solutions in environments with numerous non-equivalent query rewritings. In this paper, we also report experimental studies that characterize trends, correlations and independence among the different efficiency factors, and demonstrate the utility of the proposed QC-Model in terms of establishing a ranking among rewritings.

Keywords: Evolvable view environment, view synchronization and preservation, data warehouse, cost model, information descriptions, evolving information sources, rewriting views.

*This work was supported in part by the NSF NYI grant #IRI 94-57609. We would also like to thank our industrial sponsors, in particular, IBM and Informix.

1 Introduction

Advanced applications such as web-based information services, data warehousing, digital libraries, and data mining typically gather data from a large number of interconnected Information Sources (ISs) in an environment such as the World Wide Web [Wid95]. There is generally a large variety and number of autonomous ISs to be expected, having diverse data models, supporting different query interfaces and query processing capabilities, and even freely updating both their contents and their capabilities. In order to provide efficient information access in such environments, relevant data is often retrieved from several sources, integrated as necessary, and then assembled into a *materialized view*. Besides providing simplified and customized information access to customers without the necessary technical background, materialized views may also offer a higher availability — offering better query performance as all information can be retrieved from a single location.

However, one important and as of now not yet addressed problem for these applications is that current view technology only supports *static, a-priori-specified* view definitions – meaning that views are assumed to be specified on top of a fixed environment. Once the underlying ISs change their capabilities, the views derived from them may become undefined. This new problem is in contrast to work on incremental view maintenance which addresses changes at the data but not at the schema level [ZGMW96, ZWGM97] and to recent work on view redefinition [GMR95, MD96].

In the *EVE* (Evolvable View Environment) project [LNR97b], we began to tackle this new problem [LNR97a, NLR97, RLN97, NLR98]. We have proposed Evolvable SQL (E-SQL), an extension of SQL that allows a view definer to specify preferences about what information is essential and what information can be replaced or discarded in a materialized view when the base tables providing this information change or become unavailable. Another contribution of our work is the Model for Information Source Description (MISD) that allows us to express semantic relationships and overlaps between ISs, while making minimal assumptions about the data model and capabilities of the ISs. Based on this framework, the *EVE* system is able to maintain materialized views (data warehouses) as underlying ISs change their schemas. This process, which we call *view synchronization*, is accomplished by adapting view queries according to the schema changes that the underlying ISs undergo.

When a view is synchronized with a capability change, there are typically numerous possible new view queries *(rewritings)* that preserve the original query to some degree, depending on how the view was specified using E-SQL and what meta data about ISs is known (i.e., has been recorded using the MISD). Each of these new view queries will in general preserve a different amount and different types of information, which for the purpose of this paper we will term the *quality* of the view. Also, each new view query will cause different *view maintenance costs*, since data will have to be collected from different ISs. We contend that, with these two dimensions, it is possible to *compare* different view queries (or *legal rewritings*) with each other, even if they are not equivalent (as commonly assumed for query rewritings in the context of query optimization [BLT86, GM95] or in more recent work on rewriting queries using views [LMS95, CKP95]).

Since a large number of different legal rewritings may exist for a capability change and a given view, we need a model that allows us to compare these rewritings, establish some (numeric) ranking among them indicating the relative preference of one solution over alternate ones, and to possibly allow us to identify the *best* solution. This question becomes important with a growing number of ISs to query data from, especially if these ISs are independent from each other and are prone to undergo capability changes during the lifetime of a materialized view (as is the case on the Internet). A typical example would be a service that queries information about flights and hotel reservations from several travel agencies on the WWW. Here, it is likely that one of the participants in the system (e.g., an airline company or a hotel chain) changes the type of services (queries) it supports. This would cause our algorithms to generate a number of suggestions for a new view query that preserve different amounts of information from the old view query and which would have to be compared against each other.

To solve this problem, we present the *QC-Model* in this paper which is capable of comparing different view rewritings (generated as replacements for an original view that has become invalid). It measures the *efficiency* of a query rewriting (view) in the two dimensions of *quality* and *cost*. Quality is a function of the view interface and the set of data being returned from a view query compared to the original view. Cost refers to *incremental view maintenance* costs and measures how expensive the new query would be for long-term maintenance of data in the data warehouse. The two dimensions, quality and cost, are composed of several relevant factors, such as interface and extent for the quality; transferred data, number of messages, and I/O cost for the cost. We have integrated these two measures into an efficiency model that lets us trade-off between the quality and cost of a view rewriting.

This QC-Model allows us to now easily compare different view queries based on some computable numeric efficiency score taking complex aspects of the solutions into account. Our model can thus be helpful to users by ranking all possible synchronization solutions in a linear order and allowing the user to make the final choice. Given that view definers in EVE can provide the system with preferences on view evolution (using E-SQL), the complete process of choosing among alternate solutions to view synchronization could even be made transparent to a view user.

Our experimental studies reported in this paper demonstrate the utility of the proposed efficiency model in terms of establishing a ranking among rewritings. Our studies succeed in identifying cases with trends and correlations among the different efficiency factors, as well as cases where the measures of cost and quality are independent from one another. The former findings then lead us to suggest possible heuristics for the optimization of the rewriting process itself. These heuristics may be utilized in the future to develop algorithms that find good legal rewritings without first having to compute values for a complete set of rewritten queries. This is however beyond the scope of this current work.

While we have initially developed this novel QC-Model in the context of our view synchronization problem of EVE, it is also likely to be useful for a number of different scenarios in the area of query reformulation such as query rewriting using views [LMS95], where several different ways of rewriting a query are conceivable, or data warehouse maintenance [AAS97, ZGMHW95], for which our cost measurement approach and optimization insights gained by our performance studies can be valuable.

This work builds partly upon the *EVE* project and is complementary to our previous work on *finding* and *generating* view rewritings [LNR97b]. It goes beyond the work in traditional query optimization [JK84, vdBK94, AAS97] since it is not concerned with finding the *best way* to execute a well-defined query, but with finding the *best (not necessarily equivalent) query* for a particular problem. It builds upon previous cost models of incremental view maintenance [BLT86, ZGMHW95, ZWGM97] but extends the QC-Model to now also incorporate a measure of divergence from the intended view specification.

In summary, the contributions of this paper are fourfold: First, it identifies the new problem of trade-offs of quality against cost for query rewriting in general and for view synchronization in particular and the need for an efficiency model for assessing and evaluating these measures. Second, we explore how to overall find *good* replacements for materialized view queries that become invalid due to schema changes of underlying base tables.

Third, we elaborate this by introducing a new measure of *quality* for such a replacement, and adopt existing cost models [ZGMHW95] for our purposes in order to establish an integrated efficiency model for view synchronization evaluation. The fourth contribution is an experimental evaluation of our findings that demonstrates the utility of our QC-Model .

The remainder of this paper is organized as follows: Section 2 gives an overview over related work. Section 3 introduces our EVE framework addressing the issue of view adaptation under capability changes and explains previous work that lead us to the research on the quality and cost issues discussed in this paper. Section 4 introduces our proposed overall solution strategy, whereas Sections 5 and 6 present a detailed analytic model of quality and performance (cost) trade-offs, respectively. Section 7 summarizes experiments we have run to demonstrate the utility of our approach. Section 8 discusses our conclusions.

2 Related Work

To our knowledge, we are the first to study the problem of view synchronization caused by capability changes of participating ISs. Most of the prior work on views in distributed database systems has focused on the problem of view maintenance (e.g., propagating data changes to the view) [GM95, QW97, CKL+96].

In [RLN97], we establish a taxonomy of view adaptation problems that identifies alternate dimensions of the problem space, and hence serves as a framework for characterizing and hence distinguishing our view synchronization problem from other (previously studied) view adaptation problems. In [LNR97a, LNR97b], we introduce the overall *EVE* solution framework, in particular the concept of associating evolution preferences with view specifications. We also present an algorithm for achieving view synchronization, called the Simple View Synchronization algorithm (SVS). SVS produces a set of view definitions that can all be used as legal rewritings for view queries under capability changes. A new algorithm that is more general and imporves on the current SVS algorithm is introduced in [NLR98]. This Complex View Synchronization algorithm (CVS) generates an even larger number of alternative legal rewritings, thus raising the need for an efficiency model as introduced. This current paper now addresses this need by establishing a model for systematically ranking otherwise incomparable solutions for view synchronization.

The problem of *query optimization* has been addressed for instance by Jarke et al. [JK84], van den Berg et al. [vdBK94], or Agrawal et al. [AAS97]. These works are concerned with optimizing a given query in order to execute it in an efficient way. View synchronization in the *EVE* environment encounters a different problem, namely we have to select a good (but not necessarily *equivalent*) query among several possible ones. One component of the desirability of a new view is of course the cost of view maintenance after IS data updates. For this, we hence are able to adopt and adapt a measure similar to traditional view maintenance costs [ZGMHW95], measuring different parameters in an incremental update environment. But for our purposes, we also have to look at the quality of different view definitions in comparison to the original query in terms of their degrees of preservation of interfaces and extents, a problem that does not occur in traditional query optimization.

For the problem of *incremental view maintenance*, a concept which we use in our performance studies, earlier work has been done by several other projects in the literature [CTL+96, GMS93]. Blakeley et al. [BLT86] are concerned with a centralized environment only. Also, they have looked at incremental view maintenance assuming non-concurrent updates (updates are sufficiently spaced to not interfere with each other, each update reaches the data warehouse before the next update is executed at any of the base relations). Lately, work on concurrent updates has been done. Based on the concept of updates interfering with each other due to long transmission times between base relations and the data warehouse, these works attack increasingly complex scenarios of handling concurrent updates by collecting update information in queues and handling them in batches. Zhuge et al. [ZGMHW95, ZWGM97] introduce the ECA algorithm for incremental view maintenance and report on findings on the cost of their algorithm, but in a different environment from ours (a single information source is assumed). They also give a taxonomy for different *levels of correctness* for view maintenance algorithms. A second paper by the same authors ("Strobe", [ZGMW96]) extends their findings towards multi-source information spaces, but does not incorporate any performance model or cost studies. Agrawal et al. [AAS97] propose the SWEEP-algorithm, which can ensure consistency of the data warehouse in a larger number of cases compared to the Strobe family of algorithms. Finally, [ZGMHW95] contains a performance study similar to ours. However, their work is limited to a comparison between traditional view recomputation and incremental view maintenance algorithms, and does not compare quality and cost between different rewritings for a query, while the latter is the topic of our work.

To determine the *quality* of a view rewriting, we need to estimate sizes of overlapping view extents in order to determine how much information is retained by a new query and how much meaningless new data is introduced. This in some way parallels the concept of *precision and recall* used in the field of information retrieval [RJB89], although it is set in an entirely different context. Information retrieval generally does not deal with selecting subsets of tuples from a typed relation nor with combining such relation fragments via joins into larger result tuples. Rather, the work on precision and recall establishes measures of how well boolean queries perform on textual documents in terms of term similarities and counts.

Much research has been done on query reformulation using materialized views. For example, Levy et al. [LRU96, LMS95, SDJL96] consider the problem of replacing an original query with a new expression containing materialized view definitions such that the new query is *equivalent* to the old one. To the best of our knowledge, there is no work done in this context of query reformulation using views with the goal of generating queries without *equivalence* (for example, the new reformulated query could be a subset of the original query). This approach to query reformulation [LMS95] has some similarities with our view synchronization process, but again it is set in a different environment and has different goals. Namely, we have extended the idea of query reformulation by using a well-defined query language E-SQL to specify constraints on query reformulation, thus, when in compliance with those constraints, we allow the view redefinitions to be for example a subset or a superset of the original view. Furthermore, we introduce the concept of ranking alternative view redefinitions through the QC-Model presented in this paper.

3 Review of the EVE Project

In this section, we will review the concepts of the *Evolvable View Environment* (*EVE*) [LNR97b] as needed for the remainder of this paper. Our *EVE*-system provides a solution for the problem of capability changes in distributed networks of information systems (Figure 1).

Major concepts of this architecture are [LNR97b]:

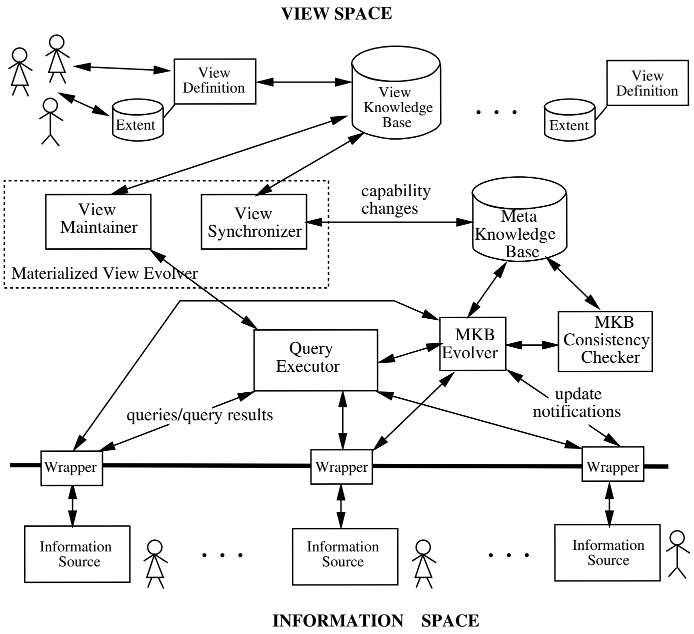

Figure 1: The Framework of the Evolvable View Environment (*EVE*).

- **IS Registration**
  All information sources (IS) participating in the system register themselves with EVE through the Meta Knowledge Base (MKB). The ISs are assumed to be autonomous in their actions, yet semi-cooperative in the sense that it is possible to establish certain facts (constraints) about their data and relationships of data between ISs. They also show a certain level of cooperation, in the sense that we assume that the EVE system is notified when a data or capability change in one of the ISs occurs.

- **Meta Knowledge Base (MKB)**
  Meta information about participating ISs is stored in the MKB. The MKB consists primarily of information about semantic interrelationships observed between different ISs registered in the system (cf. Section 3.2 for the Model of Information Source Description).

- **View Knowledge Base**
  The view knowledge base stores information about views defined over the ISs by different users. These views are augmented with a user preference model about view evolution (cf. Section 3.1 for Evolvable SQL).

- **View Synchronization**
  When underlying ISs change their *schema* (not just their *data*, as other projects typically assume), existing *view queries* have to be adapted in order to keep providing information to their users. This goal is accomplished by *EVE* by *synchronizing* views with the schema changes of underlying ISs (cf. Section 3.3).

Important features of our approach are an extension of SQL to allow for expressing preferences in queries, a model to specify relations between information sources, and an algorithm to keep up views under schema changes of underlying ISs. These three features will now be introduced.

3.1 Establishing Preferences for the Evolution of SQL Views — E-SQL

E-SQL or *Evolvable-SQL* is an extension of SQL that allows the view definer to express preferences for view evolution [LNR97b]. A user defining a view can specify what information is indispensable, what information is replaceable by (well defined) similar information from other ISs and whether a changing view extent is acceptable under certain circumstances. This enables the *EVE* system to better evolve a view under a schema change of one of the underlying ISs. The following extension to the original SQL SELECT-FROM-WHERE syntax incorporates user evolution preferences:

$$\begin{array}{ll}
\textsf{CREATE VIEW} & V\ (B_1,\ldots,B_m)\ \ (\mathcal{VE}=\mathcal{VE}_V)\ \textsf{AS} \\
\textsf{SELECT} & R_1.A_{1,1}(\mathcal{AD}=\mathcal{AD}_{1,1},\mathcal{AR}=\mathcal{AR}_{1,1}),\ldots,R_1.A_{1,i_1}(\mathcal{AD}=\mathcal{AD}_{1,i_1},\mathcal{AR}=\mathcal{AR}_{1,i_1}),\ldots, \\
 & R_n.A_{n,1}(\mathcal{AD}=\mathcal{AD}_{n,1},\mathcal{AR}=\mathcal{AR}_{n,1}),\ldots,R_n.A_{n,i_n}(\mathcal{AD}=\mathcal{AD}_{n,i_n},\mathcal{AR}=\mathcal{AR}_{n,i_n}) \\
\textsf{FROM} & R_1(\mathcal{RD}=\mathcal{RD}_1,\mathcal{RR}=\mathcal{RR}_1),\ldots,R_n(\mathcal{RD}=\mathcal{RD}_n,\mathcal{RR}=\mathcal{RR}_n) \\
\textsf{WHERE} & C_1(\mathcal{CD}=\mathcal{CD}_1,\mathcal{CR}=\mathcal{CR}_1)\ AND\ \ldots\ AND\ C_k(\mathcal{CD}=\mathcal{CD}_k,\mathcal{CR}=\mathcal{CR}_k)
\end{array} \tag{1}$$

Figure 2: Syntax of a Generic E-SQL View Definition.

The set $\{B_1,\ldots,B_m\}$ corresponds to the local names given to attributes preserved in the view V, the set $\{A_{s_{j,1}},\ldots,A_{s_{j,i_j}}\}$ is a subset of the attributes of relation R_j with $j=1,\ldots,n$; C_i with $i=1,\ldots,k$, are primitive clauses defined over the attributes of relations in the FROM clause. A primitive clause has one of the following forms: $(<attribute-name> \theta <attribute-name>)$ or $(<attribute-name> \theta <value>)$ with $\theta \in \{<,\leq,=,\geq,>\}$. All parameters $\mathcal{VE},\mathcal{AD},\mathcal{AR},\mathcal{RD},\mathcal{RR},\mathcal{CD}$, $\mathcal{CR}$ and their respective values are defined as given in Figure 3. For view components that have their evolution parameter values omitted, the default value is given in column 3 of the table.

Evolution Parameter		Domain	Default
Attribute-	dispensable ($\mathcal{AD}$)	*true/false* (attribute is dispensable/indispensable)	false
	replaceable ($\mathcal{AR}$)	*true/false* (attribute is replaceable/non-replaceable)	false
Condition-	dispensable ($\mathcal{CD}$)	*true/false* (condition is dispensable/indispensable)	false
	replaceable ($\mathcal{CR}$)	*true/false* (condition is replaceable/non-replaceable)	false
Relation-	dispensable ($\mathcal{RD}$)	*true/false* (relation is dispensable/indispensable)	false
	replaceable ($\mathcal{RR}$)	*true/false* (relation is replaceable/non-replaceable)	false
View-	extent ($\mathcal{VE}$)	$\approx$: no restriction on the new extent $\equiv$: new extent is equal to old extent $\supseteq$: new extent is superset of old extent $\subseteq$: new extent is subset of old extent	$\equiv$

Figure 3: View Evolution Parameters of E-SQL Language.

A typical E-SQL query looks like this:

$$\begin{array}{ll}
\textsf{CREATE VIEW} & \text{Asia-Customer } (\mathcal{VE}=\text{“}\supseteq\text{”})\ \textsf{AS} \\
\textsf{SELECT} & \text{Name, Address, Phone } (\mathcal{AD}=true,\ \mathcal{AR}=true) \\
\textsf{FROM} & \text{Customer C } (\mathcal{RR}=true), \text{FlightRes F} \\
\textsf{WHERE} & \text{(C.Name = F.PName) AND (F.Dest = ‘Asia’) } (\mathcal{CD}=true)
\end{array} \tag{2}$$

with all evolution parameters set to false omitted, as this is the default.

3.2 Describing Information Sources — MISD

MISD, our *Model for Information Source Description*, allows a variety of heterogeneous ISs to participate in *EVE*. This is accomplished by expressing semantic relationships between these ISs using constraints. As the wrapper of each IS translates the capabilities of its underlying IS into a common set of primitives, MISD provides a common model to describe relationships and constraints between different ISs[1]. Figure 4 shows some of the constraints supported in our system. These descriptions are collected in a Meta Knowledge Base (MKB) (see Figure 1), forming an information pool that is critical in finding appropriate replacements for view components when view definitions become undefined.

Name	Syntax
Type Integrity Constraint	$\mathcal{TC}_{R.A_i} = (R(A_i) \subseteq A_i(Type_i))$
Join Constraint	$\mathcal{JC}_{R_1,R_2} = (C_1 AND \cdots C_l)$
Partial/Complete Constraint	$\mathcal{PC}_{R_1,R_2} = (\pi_{\bar{A_1}}(\sigma_{\mathcal{C}(\bar{B_1})} R_1)\ \theta\ \pi_{\bar{A_2}}(\sigma_{\mathcal{C}(\bar{B_2})} R_2))$

Figure 4: Possible Types of Semantic Constraints for IS Descriptions.

The basic units of information available in each of the ISs are described as follows:

$$IS.R(A_1, \ldots, A_n). \tag{3}$$

The domain types of the attributes A_i are described using *type integrity constraints*, denoted by $A_i(Type_i)$. A *join constraint* between two relations R_1 and R_2, denoted as $\mathcal{JC}_{R_1,R_2}$, states that tuples in R_1 and R_2 can be meaningfully joined if the join condition, i.e., a conjunction of primitive clauses, is satisfied. A generic join constraint, defined as

$$\mathcal{JC}_{R_1,R_2} = (C_1\ AND\ \cdots\ AND\ C_l) \tag{4}$$

where $C_1, \ldots, C_l$ are primitive clauses over the attributes of R_1 and R_2 states that $R_1 \bowtie_{C_1 \wedge C_2 \wedge \ldots \wedge C_n} R_2$ is a meaningful way to join R_1 and R_2.

A *partial/complete* ($\mathcal{PC}$) constraint between two relations R_1 and R_2 states that a (horizontal and/or vertical) fragment of R_1 is semantically contained or equivalent to a (horizontal and/or vertical) fragment of R_2 at all times. *EVE* makes use of the $\mathcal{PC}$ constraints to decide if an evolved view extent is equivalent to, a subset of, or a superset of the initial view extent. A generic $\mathcal{PC}$ constraint between two relations R_1 and R_2 is specified as follows:

$$\mathcal{PC}_{R_1,R_2} = \left(\ \pi_{A_{i_1},\ldots,A_{i_k}}(\sigma_{\mathcal{C}(A_{j_1},\ldots,A_{j_l})} R_1)\ \ \theta\ \pi_{A_{n_1},\ldots,A_{n_k}}(\sigma_{\mathcal{C}(A_{m_1},\ldots,A_{m_t})} R_2)\ \right) \tag{5}$$

where $A_{i_1}, \ldots, A_{i_k}, A_{j_1}, \ldots, A_{j_l}$ are attributes of R_1; $A_{n_1}, \ldots, A_{n_k}, A_{m_1}, \ldots, A_{m_t}$ are attributes of R_2; $\mathcal{TC}(R_1.A_{i_s}) = \mathcal{TC}(R_2.A_{n_s})$, for s = 1, ...,k; and θ is $\{\subseteq, \supseteq, \equiv\}$ for the partial ($\subseteq$ and $\supseteq$) or complete ($\equiv$) information constraint, respectively.

[1]In our first implementation of EVE, this is a SQL-based relational type model, although extended object models used by other projects will be employed in future versions.

3.3 View Synchronization

The *EVE* system employs several algorithms for keeping up views under schema changes of underlying ISs i.e., for achieving view synchronization [LNR97b, NLR98]. Once a view is defined, *EVE* tracks schema changes in all participating ISs for this view. Once affected view queries are identified, the view synchronizer then tries to find relations or attributes as replacements for currently deleted information from an IS by exploring meta information stored about the information space, such as join constraints or $\mathcal{PC}$-constraints.

The E-SQL evolution preferences are then used to determine whether the adapted view is considered to be acceptable by the user. If a view query can be found that preserves a "sufficient" part of the information that the old view query originally retrieved, then the view is considered *synchronized* with the new state of the information space. Such a rewritten query is called a *view rewriting*, and if it fulfills certain criteria of correctness [LNR97b], it is called a *legal rewriting*.

Schema changes supported in our current system are the ones commonly found in commercial systems, such as *delete-attribute, add-attribute, change-attribute-name, delete-relation, add-relation* and *change-relation-name.*

4 Our Approach for Ranking Legal Rewritings: An Integrated Analytic Model for Quality and Cost

In the *EVE*-system described in the previous section, several possible solutions for the rewriting of a view query are to be expected for each schema change. Depending on the degree of redundancy in the information space, the view synchronizer may find a large and possibly exponentially (over the size of the information space) growing number of legal rewritings for an affected view. Each legal rewriting may preserve a different combination of attributes (of the original view) and may be specified on disparate base relations with different cardinalities at different sites and even computed differently, e.g., with different joins compared to the original view definition. Ideally, we would like to preserve the original view fully in terms of both view interface and view extent, and at the same time be able to maintain the materialized view in an economical way.

But since this goal cannot always be reached, we introduce measures of evaluating the efficiency of a view rewriting that is close to the "intent" of the initial query (although not necessarily equivalent) and also has low long-term view maintenance costs. We express the efficiency of a query rewriting in terms of its *quality* and *cost.* The first measure is the degree of divergence of quality (i.e., information preservation) of the new view with respect to the old view (See Section 5). The second measure takes the long term view maintenance cost associated with the legal rewriting into account (See Section 6). In this paper, we will demonstrate that both quality and cost are very different for different rewritings, which makes it important to select a *good* rewriting out of all legal ones for a particular view query and schema change. Our paper focuses on how to define *efficiency* , i.e., an efficiency model for query rewritings (which we will refer to as the QC-Model), and how to compute this efficiency for a given query rewriting.

Note that all traditional view maintenance strategies conform to the notion that the rewritten query is *equivalent* to the original one. On the contrary, we relax the assumption that a query has to be replaced by an equivalent one and introduce instead the concept of *non-equivalent rewritings* with a notion of "legality" that is expressed by our preference model (E-SQL). Furthermore, the view results in *EVE* are materialized and used for an extensive period of time. This means more care has to be given to long-term maintenance cost as opposed to one-time view recomputation cost.

As an example, a legal rewriting may be able to preserve all the attributes in the old view, but only return a small subset of the original results (compared to the original view result). Another legal rewriting may not be able to preserve all the attributes, but may preserve the original results (when projecting both the new and the old views at the common subset of attributes). Similarly, a legal rewriting may preserve information to a higher degree (i.e., have a higher quality) but at the price of also a higher expected view maintenance cost. An ideal legal rewriting should preserve the original view interface and generate the same set of results without introducing any surplus tuples, and it should be efficiently maintainable in the long run. If these ideal goals cannnot be reached, we try to (1) assess the divergence from this ideal and assign this divergence a normalized numerical value, (2) estimate (normalized) long-term view maintenance cost, and (3) establish a ranking among otherwise non-comparable query rewritings by combining these two measures into one in order to compare different view rewritings.

Our proposed model gives a user of our *EVE*-system a tool to express the importance of these factors by assigning weights to them. These weights are called the *trade-off parameters* in our model. A legal rewriting ranks high and is chosen by our system, if it shows low divergence from the original view and has low maintenance cost. The *EVE*-system will recommend to replace the affected view by the legal rewriting which ranks the highest in this measurement system. Our system can also show the other legal rewritings to the user in the order of their numeric efficiency ranking. Alternatively, a user could also tune the legal rewriting selection result by setting the tradeoff parameters, if desired. A validation of the utility of our proposed QC-Model based on experimental studies can be found in Section 7.

5 Efficiency Model: Quality of a Legal Rewriting

5.1 Information Preservation in Rewritings

The information returned by a view is of great importance to the view end-users. Thus in this section, we evaluate the set of legal rewritings generated by the view synchronizer in terms of the amount of information preserved in the rewritings.

The information preserved in a view can be discussed in terms of two aspects, namely the attributes preserved in the view interface and the view extent returned by the query. Ideally, we would like to replace an affected view V by a legal rewriting V_i that *fully preserves* the original view. That is, (1) V_i preserves the original view interface, although some information may be taken from other information sources (denoted by $\text{Attr}(V_i) = \text{Attr}(V)$), and (2) it returns the same set of tuples as the original query on the original information space (denoted by $\text{Ext}(V_i) = \text{Ext}(V)$). Otherwise, V_i is said to *diverge* from the original view V. A legal rewriting V_i is *less preferred* than another legal rewriting V_j in terms of the amount of information preservation (denoted by $V_i <_{IP} V_j$), if $\text{Attr}(V_i) \subseteq \text{Attr}(V_j)$ and $\pi_{Attr(V_i) \cap Attr(V_j)}(V_i) \subseteq \pi_{Attr(V_i) \cap Attr(V_j)}(V_j)$. That is to say, the "closer" a legal rewriting is to the original view in terms of information preservation, the more preferred it is.

Example 1 *Let a view V be defined as follows:*

$$\begin{array}{ll} \textsf{CREATE VIEW} & V\ (\mathcal{VE} = `\supseteq')\ \textsf{AS} \\ \textsf{SELECT} & A,\ B\ (\mathcal{AD} = true, \mathcal{AR} = true), \\ & C\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \textsf{FROM} & R \\ \textsf{WHERE} & R.A > 10 \end{array} \tag{6}$$

Assume the attribute R.C is deleted from its site. Let's further assume that the view synchronizer fails to find any appropriate substitute for R.C. Therefore, R.C is dropped from V, and we get a legal rewriting V_1 as follows:

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_1\ (\mathcal{VE} = \text{`}\supseteq\text{'})\ \textsf{AS} \\ \textsf{SELECT} & A,\ B\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \textsf{FROM} & R \\ \textsf{WHERE} & R.A > 10 \end{array} \tag{7}$$

Since R.B in V_1 is dispensable (its attribute-dispensable parameter $\mathcal{AD}$ is set to true), another legal rewriting V_2 of V is obtained by dropping the attribute R.B as well:

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_2\ (\mathcal{VE} = \text{`}\supseteq\text{'})\ \textsf{AS} \\ \textsf{SELECT} & A \\ \textsf{FROM} & R \\ \textsf{WHERE} & R.A > 10 \end{array} \tag{8}$$

Here, V_2 is less preferred *than V_1 in terms of information preservation ($V_2 <_{IP} V_1$), because $Attr(V_2) = \{A\} \subseteq \{A,B\} = Attr(V_1)$ and $\pi_{Attr(V_i)\cap Attr(V_j)}(V_i) = \pi_{Attr(V_i)\cap Attr(V_j)}(V_j)$ with duplicates removed. (Rewriting V_1 preserves "more" than rewriting V_2.)*

Note that not all legal rewritings can be ranked as easily as in the example shown above. In the following, first we show an example that demonstrates that there is no simple way to order legal rewritings, then we present our solution approach for ranking various legal rewritings.

Example 2 *Let the view V be defined as follows:*

$$\begin{array}{ll} \textsf{CREATE VIEW} & V\ (\mathcal{VE} = \text{`}\approx\text{'})\ \textsf{AS} \\ \textsf{SELECT} & A\ (\mathcal{AD} = true, \mathcal{AR} = true), \\ & B\ (\mathcal{AR} = true), \\ & C\ (\mathcal{AD} = true, \mathcal{AR} = true), \\ & D\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \textsf{FROM} & R\ (\mathcal{RD} = true, \mathcal{AR} = true) \end{array} \tag{9}$$

Assume the relation R is deleted from its site. Let us assume that the view synchronizer finds two relations S and T as appropriate substitutes for R. The tuples in each of these relations and the view extent of the original view V are shown in Figures 5(a) and 5(b). We assume the following two legal rewritings [2]:

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_1\ (\mathcal{VE} = \text{`}\approx\text{'})\ \textsf{AS} \\ \textsf{SELECT} & A\ (\mathcal{AD} = true, \mathcal{AR} = true), \\ & B\ (\mathcal{AR} = true) \\ \textsf{FROM} & S\ (\mathcal{RD} = true, \mathcal{AR} = true) \end{array} \tag{10}$$

[2] Since the attributes preserved in V are all dispensable, our system can generate a whole spectrum of legal rewritings out of these two legal rewritings by dropping a proper subset of the view components at a time. However, we do not list all these legal rewritings one by one, because these unlisted legal rewritings are inferior to the legal rewritings listed in queries (10) and (11) in terms of information preservation.

R

A	B	C	D
1	3	1	1
1	3	2	3
1	6	3	5
2	6	4	7
2	9	5	9
2	9	6	0

S

A	B	E
1	3	7
2	6	0
1	6	9
2	8	5

T

B	C	D	F
6	4	7	7
6	4	6	4
3	1	1	2
7	6	7	6
8	1	7	1
6	3	5	5
8	2	7	9

(a) Base Relations R, S, and T.

V

A	B	C	D
1	3	1	1
1	3	2	3
1	6	3	5
2	6	4	7
2	9	5	9
2	9	6	0

(b) Original View: V

V_1

A	B
1	3
2	6
1	6
2	8

(c) View Rewriting: V_1

V_2

B	C	D
6	4	7
6	4	6
3	1	1
7	6	7
8	1	7
6	3	5
8	2	7

(d) View Rewriting: V_2

$V_1 \cap V$

A	B
1	3
2	6
1	6

Surplus tuples in V_1 but not in the common attribute set of V.

A	B
2	8

(e) Comparing V_1 with V.

$V_2 \cap V$

B	C	D
3	1	1
6	3	5
6	4	7

Surplus tuples in V_2 but not in the common attribute set of V.

B	C	D
6	4	6
7	6	7
8	1	7
8	2	7

(f) Comparing V_2 with V.

Figure 5: *Different Amounts of Information Are Preserved in Legal Rewritings.*

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_2\ (\mathcal{VE}\ =\ \text{`}\approx\text{'})\ \textsf{AS} \\ \textsf{SELECT} & B\ (\mathcal{AR}\ =\ true), \\ & C\ (\mathcal{AD}\ =\ true, \mathcal{AR}\ =\ true), \\ & D\ (\mathcal{AD}\ =\ true, \mathcal{AR}\ =\ true) \\ \textsf{FROM} & T\ (\mathcal{RD}\ =\ true, \mathcal{AR}\ =\ true) \end{array} \tag{11}$$

In this example, V_1 is able to preserve two attributes A and B, while V_2 is able to preserve three attributes B, C, and D. Therefore, from the point of view of information preservation on the view interface, V_2 is superior to V_1. However, as shown in Figures 5(e) and (f), V_1 and V_2 each preserve three tuples if the common set of attributes is considered between (V_1 and V) and (V_2 and V), respectively. But V_1 generates one surplus tuple that was not in the original view V while V_2 returns four surplus tuples that were not in V. Thus, from the viewpoint of information preservation on the view extent, V_1 seems to be superior to V_2. Therefore, our system must trade off the pros and cons between the view interface and view extent preservation in order to be able to rank these potential rewritings of V in some linear order.

5.2 Information Preservation on the View Interface

The attributes in a view interface can be classified into four categories according to their *(attribute-dispensable, attribute-replaceable)* parameter values (see Figure 6). Each row represents one type of category to which a preserved attribute may belong. Figure 6 has three columns: column one shows the values for the *(attribute-dispensable, attribute-replaceable)* parameters, column two the preservation requirement, and column three the weight. Since any legal rewriting must preserve all the indispensable attributes of the original view V, independent of whether the attributes can be taken from other information sources, all attributes in categories 3 and 4 must be preserved in the final view interface. Thus, we do not assign weights to attributes in categories 3 and 4, and we do not include these attributes in our discussion any further.

Four Categories for Preserved Attributes		
Category (dispensable, replaceable)	Preservation Requirement	Weight
C1: $(true, true)$	–	w_1
C2: $(true, false)$	–	w_2
C3: $(false, true)$	must stay	n/a
C4: $(false, false)$	must stay	n/a

Figure 6: Four Categories for Preserved Attributes.

However a view rewriting is still legal if it omits attributes in categories 1 or 2. In effect, different legal rewritings may preserve different combinations of attributes in these two categories. Therefore, we define weights w_1 and w_2 $(0 \leq w_1, w_2 \leq 1)$ for a view definer to set as needed. We will discuss effects of changing weight parameters in the evaluation section (Section 7), while in this section we now assume one fixed setting of w_1 and w_2. When w_1 and w_2 are not explicitly specified by the user, EVE uses the default values set in the system $((w_1, w_2) = (0.7, 0.3))$. The default settings have the property $w_1 > w_2$ [3]. This represents the fact that EVE is in favor of preserving the replaceable attributes (i.e., attributes in category 1). A view having replaceable attributes may be evolved further as more schema changes occur (as our experimental evaluation in Section 7 confirms, whereas having relatively many non-replaceable attributes (i.e., attributes in category 2) has a negative effect on the further evolvability of a view query. In other words, it is harder to find good legal rewritings for a view if its view elements are non-replaceable.

5.3 Information Preservation on View Extent

We now introduce the notation for *common subset of attributes* and *common-subset-of-attributes-equivalence* which we will need in the remainder of this section.

Definition 1 ***Common Subset of Attributes of*** V ***with respect to*** V_i***.***
Let V *and* V_i *be two relations, such that* $Attr(V) \cap Attr(V_i) \neq \emptyset$*. We use* $V^{\pi(V_i)}$ *to denote the projection of relation* V *on the common attributes of* V *and* V_i*. That is,* $V^{\pi(V_i)} = \pi_{Attr(V) \cap Attr(V_i)} V$*. Similarly,* $V_i^{\pi(V)}$ *is defined as* $\pi_{Attr(V) \cap Attr(V_i)} V_i$*.*

Besides considering the attributes preserved in the legal rewritings, the *sets of tuples* returned by the queries will also have an impact on the user's satisfation with a view rewriting V_i. When the view interfaces of a legal rewriting V_i and the original view V are not the same, the extent preservation evaluation is done by comparing tuples on the common subset of attributes only [4]. When V_i and V have different view interfaces, we say V_i is *common-subset-of-attributes-equal* to V, denoted by $V_i =_\pi V$, if their projections on the common subset of attributes are equal.

[3] Note that the absolute values of w_1 and w_2 are not as important as their relative values, since the measure will be normalized later.

[4] When the view interfaces of V_i and V are the same, the extent comparison is done as usual.

Definition 2 ***Common-Subset-of-Attributes Equivalence.***
$V =_\pi V_i$, *iff*

(I) $\forall t \in V, \exists\, t_i \in V_i$ *s.t.* $t[Attr(V) \cap Attr(V_i)] = t_i[Attr(V) \cap Attr(V_i)]$. *That is,* $V^{\pi(V_i)} \subseteq V_i^{\pi(V)}$.

(II) $\forall t_i \in V_i, \exists\, t \in V$ *s.t.* $t_i[Attr(V) \cap Attr(V_i)] = t[Attr(V) \cap Attr(V_i)]$. *That is,* $V_i^{\pi(V)} \subseteq V^{\pi(V_i)}$.

Condition (I) examines whether the legal rewriting V_i preserves all the tuples in the original view V with respect to the common subset of attributes (with duplicates removed). Condition (II) investigates whether there are surplus tuples in V_i, i.e., tuples in V_i but not in V with respect to the common subset of attributes. Other set operations between the view extents of V_i and V can be similarly defined on the common subset of attributes of V_i and V. We summarize the operations and their semantics in Figure 7.

Set Operator	Semantics
$V =_\pi V_i$	$\forall\, t \in V_i, \exists\, t_i \in V$ s.t. $t[Attr(V) \cap Attr(V_i)] = t_i[Attr(V) \cap Attr(V_i)]$ and $\forall\, t_i \in V_i, \exists\, t \in V$ s.t. $t_i[Attr(V) \cap Attr(V_i)] = t[Attr(V) \cap Attr(V_i)]$
$V_i \subseteq_\pi V$	$\forall\, t_i \in V_i, \exists\, t \in V$ such that $t_i[Attr(V) \cap Attr(V_i)] = t[Attr(V) \cap Attr(V_i)]$
$V \cap_\pi V_i$	$\{z \mid \exists\, t \in V \wedge \exists\, t_i \in V_i,\ z = t[Attr(V) \cap Attr(V_i)] = t_i[Attr(V) \cap Attr(V_i)]\}$
$V \setminus_\pi V_i$	$\{z \mid \exists\, t \in V, z = t[Attr(V) \cap Attr(V_i)] \wedge \nexists\, t_i \in V_i,\ z = t_i[Attr(V) \cap Attr(V_i)]\}$

Figure 7: Set Operators on the Common Subset of Attributes of V and V_i.

Intuitively, we would like to choose a legal rewriting such that Conditions (I) and (II) are both satisfied. If it is not possible to find a legal rewriting that satisfies both conditions, we choose a legal rewriting that produces a view extent as *close* as possible to the original one. Some legal rewritings may have a larger number tuples in V preserved, but at the same time generate many extra tuples that were not in V. On the other hand, some legal rewritings may preserve less tuples in V, but also generate less surplus tuples. In the next section, we discuss how to generate a *good* rewriting according to the user's preference by making a choice which tries to have both conditions (I) and (II) satisfied to an as large degree as possible.

5.4 Metric of Quality: Degree of Divergence ($\mathcal{DD}$)

We will now present the metric of quality, i.e., the *degree of divergence* $\mathcal{DD}$, that we define to appraise the quality of legal rewritings of a view. In this section, we first discuss information preservation in terms of the view interface, then we explain our findings on information preservation in terms of the view extent. Finally, we unify the discussions into one quality measure – the *Degree of Divergence* of a legal rewriting from the original view.

5.4.1 Degree of Divergence on the View Interface ($\mathcal{DD}_{attr}(V_i)$)

Let V be a view and V_i ($i \geq 1$) a legal rewriting of V under the capability change $\mathcal{CC}$. Let Attr(V_i) ($i \geq 0$) be the attributes specified in the SELECT clause of V_i. As mentioned earlier, all indispensable attributes in V must be preserved in every legal rewriting V_i for it to be considered legal. So the indispensable attributes do not have to be included in our discussion. An attribute A in the view interface of V_i, $A \in Attr(V_i)$, has two boolean properties *attribute-dispensable* $\mathcal{AD}(A)$ and *attribute-replaceable* $\mathcal{AR}(A)$ corresponding to their evolution parameters set in the E-SQL definition of the view V_i. Thus, the attributes in $Attr(V_i)$ are classified according to the following rules:

$$
\begin{aligned}
A_i^1 &= \{A \mid A \in Attr(V_i) \ \wedge \mathcal{AD}(A) = true \wedge \mathcal{AR}(A) = true\} \\
A_i^2 &= \{A \mid A \in Attr(V_i) \ \wedge \mathcal{AD}(A) = true \wedge \mathcal{AR}(A) = false\}
\end{aligned}
$$

We define the quality of the view interface of a view V_i as:

$$
Q_{V_i} = \ |A_i^1| \cdot w_1 + |A_i^2| \cdot w_2, \ \mathit{for} \ i \geq \ 1 \tag{12}
$$

where $|A_i^j|$ is the number of attributes of V_i that fall into the category j (i.e., the cardinality of the set A_i^j). The quality of the original view V is defined likewise and denoted by Q_V.

The (normalized) degree of divergence of V_i from V in terms of the view interface, denoted by $DD_{attr}(V_i)$, can then be defined as:

$$
DD_{attr}(V_i) = \begin{cases} 0 & \text{if } Q_V = 0 \\ \frac{Q_V - Q_{V_i}}{Q_V} & \text{otherwise} \end{cases}
$$

This is a measure of *distance* of the interface of a view rewriting from the original view interface. $Q_V = 0$ means that the attributes contained in the original view V are all indispensable (since indispensable attributes do not have weights and are not considered in this computation). In this case, if V is evolvable and a legal rewriting V_i is found, then V_i must preserve the indispensable attributes entirely. That is, $Q_{V_i} = Q_V$ (i.e., $DD_{attr}(V_i) = 0$). However, when there are one or more attributes in Attr(V) that are dispensable, then $Q_V > 0$ and $DD_{attr}(V_i)$ is computed as defined above. When V_i does not preserve any of the dispensable attributes, then $Q_{V_i} = 0$ and $DD_{attr}(V_i) = 1$. So in terms of the view interface, a legal rewriting V_i is preferred to the legal rewriting V_j if $DD_{attr}(V_i) < DD_{attr}(V_j)$.

Example 3 *We look at the view and legal rewritings defined in Example 1. In that example, Attr*$(V) = \{A, B, C\}$, *and the two attributes B and C fall into category 1. Therefore,* $Q_V = 2 \cdot w_1$. *The legal rewriting* V_1 *preserves the attribute B (besides the indispensable attribute A). Therefore,* $Q_{V_1} = 1 \cdot w_1$. *On the other hand, the legal rewriting* V_2 *only preserves the indispensable but none of the dispensable attributes. Therefore,* $Q_{V_2} = 0$. *Intuitively,* V_2 *diverges more from* V *than* V_1 *does in terms of view interface preservation. Thus,* V_1 *is preferred to* V_2 *as indicated by* $0.5 = DD_{attr}(V_1) < DD_{attr}(V_2) = 1$.

5.4.2 Degree of Divergence on the View Extent ($\mathcal{DD}_{ext}(V_i)$)

The view extent of a legal rewriting V_i may diverge from the original view extent of V in two aspects:

D1. The (relative) number of tuples from the original view V that are not preserved in the new view V_i, denoted by

$$
DD_{ext\_D1}(V_i) = 1 \ - \ \frac{|V_i \cap_\pi V|}{|V^{\pi(V_i)}|} \ = \ \frac{|V \setminus_\pi V_i|}{|V^{\pi(V_i)}|} \tag{13}
$$

D2. The (relative) number of surplus tuples in the new view V_i that are not in the original view V, denoted by

$$
DD_{ext\_D2}(V_i) = \frac{|V_i \setminus_\pi V|}{|V_i^{\pi(V)}|} \ = \ \frac{|V_i^{\pi(V)}| - |V \cap_\pi V_i|}{|V_i^{\pi(V)}|} \tag{14}
$$

with $V^{\pi(V_i)}$ and $V_i^{\pi(V)}$ as defined in Definition 1.

Note that, intuitively, the number of tuples that are *not preserved* (**D1**) has to be related to the size of the original view extent $|V^{\pi(V_i)}|$, whereas the number of extra tuples coming into the new view (**D2**) must be seen relative to the size of the *new* view extent $|V_i^{\pi(V)}|$. Figure 8 shows the four possible cases of the relationship between the original view V and a view rewriting V_i (none, either or both of **D1**, **D2** can be empty sets).

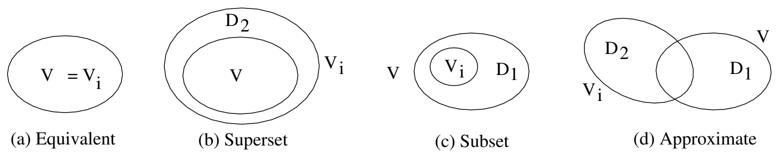

Figure 8: Divergence of New View Extent V_i from Original View Extent V.

The total extent divergence of V_i from V is the weighed sum of $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D2}(V_i)$, denoted by $\mathcal{DD}_{ext}(V_i)$, and defined as follows:

$$\begin{aligned} DD_{ext}(V_i) &= \varrho_1 \cdot DD_{ext\_D1}(V_i) + \varrho_2 \cdot DD_{ext\_D2}(V_i) \\ &= \varrho_1 \cdot \frac{|V^{\pi(V_i)}| - |V_i \cap_\pi V|}{|V^{\pi(V_i)}|} + \varrho_2 \cdot \frac{|V_i^{\pi(V)}| - |V \cap_\pi V_i|}{|V_i^{\pi(V)}|} \\ &= 1 - \frac{(\varrho_1 |V_i^{\pi(V)}| + \varrho_2 |V^{\pi(V_i)}|) \cdot |V_i \cap_\pi V|}{|V^{\pi(V_i)}||V_i^{\pi(V)}|} \end{aligned} \tag{15}$$

where $\varrho_1,\ \varrho_2$ are the trade-off parameters between $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D2}(V_i)$ ($\varrho_1,\ \varrho_2 \geq 0$ and $\varrho_1 + \varrho_2 = 1$). Remember that we compare the common subset of attributes in the view extents of V_i and V only, with duplicates removed first. Again, the view definer is given a chance to set the trade-off parameters. If the view definer does not set the parameters explicitly, then the default setting $(\varrho_1, \varrho_2) = (0.5, 0.5)$ is used.

When the view definer sets the view-extent parameter $\mathcal{VE}$ = '$\equiv$', no view extent divergence is allowed, i.e., $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D1}(V_i)$ must be both zero. Therefore, we do not discuss this case further. When the view-extent parameter $\mathcal{VE}$ = '$\supseteq$', the first number ($DD_{ext\_D1}(V_i)$) has to be zero for a rewriting to be legal (since all objects of V must also appear in the new view V_i). Therefore, we only have to compute the second measure in order to know the efficiency of V_i in terms of view extent information preservation [5]. On the contrary, when the view-extent parameter $\mathcal{VE}$ = '$\subseteq$', the second measure returned is always zero. Therefore, we only need to compute the first measure [6]. Whenever the view-extent parameter $\mathcal{VE}$ ='$\approx$', both numbers have to be computed (and for meaningful results we should have $0 < \varrho_1, \varrho_2 < 1$). For $\mathcal{VE}$ = '$\supseteq$' and $\mathcal{VE}$ = '$\subseteq$', none of the expensive set intersection operations is required. For these cases, the degree of divergence can be computed by counting the numbers of tuples in the legal rewriting V_i and the original view V (since either $V \subset_\pi V_i$ or $V_i \subset_\pi V$, so the size of the intersection is equal to the size of the smaller relation).

When $\mathcal{VE}$ = '$\subseteq$', then $V_i \subseteq_\pi V$. The expression of $DD_{ext}(V_i)$ becomes:

[5] Since the value of ϱ_1 is not relevant in this case, one could set $\varrho_1 = 0$ and $\varrho_2 = 1$ for all "superset"-views.

[6] Similarly, this case can be supported by having $\varrho_1 = 1$ and $\varrho_2 = 0$

$$
\begin{aligned}
DD_{ext}(V_i) &= \varrho_1 \cdot DD_{ext\_D1}(V_i) + \varrho_2 \cdot DD_{ext\_D2}(V_i) \\
&= \varrho_1 \cdot \frac{|V \setminus_\pi (V_i \cap_\pi V)|}{|V^{\pi(V_i)}|} + \varrho_2 \cdot \underbrace{\frac{|V_i \setminus_\pi (V \cap_\pi V_i)|}{|V_i^{\pi(V)}|}}_{0 \text{ because } V_i \subseteq_\pi V} \\
&= \varrho_1 \cdot \frac{|V^{\pi(V_i)}| - |V_i \cap_\pi V|}{|V^{\pi(V_i)}|} \\
&= \varrho_1 \cdot \frac{|V^{\pi(V_i)}| - |V_i^{\pi(V)}|}{|V^{\pi(V_i)}|}
\end{aligned} \tag{16}
$$

When $\mathcal{VE}$ = '$\supseteq$', then $V_i \supseteq_\pi V$. The expression of $DD_{ext}(V_i)$ becomes:

$$
\begin{aligned}
DD_{ext}(V_i) &= \varrho_1 \cdot DD_{ext\_D1}(V_i) + \varrho_2 \cdot DD_{ext\_D2}(V_i) \\
&= \varrho_1 \cdot \underbrace{\frac{|V \setminus_\pi (V_i \cap_\pi V)|}{|V^{\pi(V_i)}|}}_{0 \text{ because } V_i \supseteq_\pi V} + \varrho_2 \cdot \frac{|V_i \setminus_\pi (V \cap_\pi V_i)|}{|V_i^{\pi(V)}|} \\
&= \varrho_2 \cdot \frac{|V_i \setminus_\pi (V \cap_\pi V_i)|}{|V_i^{\pi(V)}|} \\
&= \varrho_2 \cdot \frac{|V_i^{\pi(V)}| - |V^{\pi(V_i)}|}{|V_i^{\pi(V)}|}
\end{aligned} \tag{17}
$$

5.4.3 Size Estimation of Overlapping View Extents

In order to compute degrees of divergence using Equation (15), we need to know how many tuples are common to both the view extents of V_i and V (we need to determine the size of the intersection $V_i \cap_\pi V$) [7]. With this number and the number of tuples in both the original view extent V and the new view extent V_i, we can compute a degree of divergence $DD_{ext}(V_i)$ for the view V_i. We will now discuss how one could estimate the size of the intersection $V_i \cap_\pi V$. To help to derive the size of the overlapping view extents, we consider the following example:

Example 4

CREATE VIEW	V ($\mathcal{VE}$ = '$\approx$') *AS*	
SELECT	$R.A$ ($\mathcal{AR}$ = $true$),	
	$S.B$	
FROM	R ($\mathcal{RR}$ = $true$),	(18)
	S	
WHERE	$R.A = S.A$ ($\mathcal{CR}$ = $true$)	

with the constraints (in the MKB):

1. Three tables R, S, and T are defined as $R(A)$, $S(A,B)$, and $T(A,B)$, respectively.
2. $\mathcal{JC}_{\mathbf{R,\ S}} = (R.A = S.A)$
3. $\mathcal{JC}_{\mathbf{S,\ T}} = (S.A = T.A)$

[7] As mentioned earlier, this computation is only necessary in the case of $\mathcal{VE}$ = "$\approx$"

After the capability change *"delete-relation R"*, one possible rewriting for this view would be [LNR97b]

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_1\ (\mathcal{VE} = \text{`}\approx\text{'})\ \textsf{AS} \\ \textsf{SELECT} & T.A\ (\mathcal{AR}\ =\ true), \\ & S.B \\ \textsf{FROM} & T(\mathcal{RR}\ =\ true), \\ & S \\ \textsf{WHERE} & T.A = S.A\ (\mathcal{CR}\ =\ true) \end{array} \tag{19}$$

This rewriting replaces the dropped relation R with the relation T, replacing the one missing attribute $R.A$ in the view by the attribute $T.A$ and adapting the FROM and WHERE clauses accordingly. Starting from this example, we now demonstrate how to calculate the parameters for Equation (15). In our example, we replaced attribute $R.A$ with $T.A$. In order to do a meaningful estimation for an overlapping view extent, we need some information about the underlying relations, which can be specified for example using $\mathcal{PC}$-constraints [8].

Estimating Overlaps of Relations Using PC-Constraints

We now discuss the estimation of the size of overlapping relations in general terms. We will refer to these findings in the remaining discussion of Example 4. A $\mathcal{PC}$ constraint between two attributes in two different relations in the original and derived views, respectively, enables us to compute view extent overlaps between those two views. As a reminder, a $\mathcal{PC}$ constraint (for a replacement of attributes in a relation R by attributes in a second relation T) is of the following form (cf. Equation 5, page 8):

$$\mathcal{PC}_{R_1,R_2} \quad = \quad (\pi_{A_{i_1},\ldots,A_{i_k}}(\sigma_{\mathcal{C}_{R_1}(A_{j_1},\ldots,A_{j_l})}R_1)\ \ \theta\ \pi_{A_{n_1},\ldots,A_{n_k}}(\sigma_{\mathcal{C}_{R_2}(A_{m_1},\ldots,A_{m_t})}R_2))$$

In such a constraint, either the left, or the right, or both selection conditions $\mathcal{C}_{R_1}(A_{j_1},\ldots,A_{j_l})$, $\mathcal{C}_{R_2}(A_{m_1},\ldots,A_{m_t})$ can be conjunctions of primitive clauses or the tautologically true condition (i.e., always true). This gives us four different kinds of $\mathcal{PC}$ constraints to consider. With the relation in the $\mathcal{PC}$-constraint set to any of the three values $\theta \in \{\subseteq, \supseteq, \equiv\}$, we have twelve different kinds of $\mathcal{PC}$-constraints to consider when comparing old and new view extents (see Figure 9). The size of the overlapping parts of the underlying relations for the original and the evolved view can be computed for most of these kinds of constraints. For other cases we can find minimal boundaries for the intersection sizes. In order to determine intersection sizes, some statistical parameters about the underlying relations are necessary. In those cases where the selection conditions in the $\mathcal{PC}$ constraints are not the tautologically true condition we need to know the selectivity of those conditions. For the following, we assume that these selectivities σ_{R_1} and σ_{R_2} are known, and that we also know the cardinalities of the dropped relation and the relation used for replacement ($|R_1|$ and $|R_2|$, respectively) [9].

Now we can evaluate the different cases of $\mathcal{PC}$ constraints. It turns out that in many of the cases the size of the overlapping set is determined by the smaller of the two relations (all cases with exact inclusion of one relation in another). In most other cases, the exact overlap cannot be determined since the $\mathcal{PC}$ constraint given is not sufficient to determine all tuples in the intersection. A graphical representation of the findings is given in Figure 9. In the figure, each row represents $\mathcal{PC}$ constraints of a certain type as described above. For example, the first row labeled no/no represents $\mathcal{PC}$ constraints with the selection conditions on both sides of the constraint

[8] Note that if the view extent parameter $\mathcal{VE}$ is *not* set to '$\approx$', $\mathcal{PC}$ constraints are *necessary* to even determine if a rewriting is legal. But even in the '$\approx$'-case, $\mathcal{PC}$ constraints help to estimate overlapping view extents.

[9] These parameters are computed by many commercial database management systems and are stored in a data dictionary.

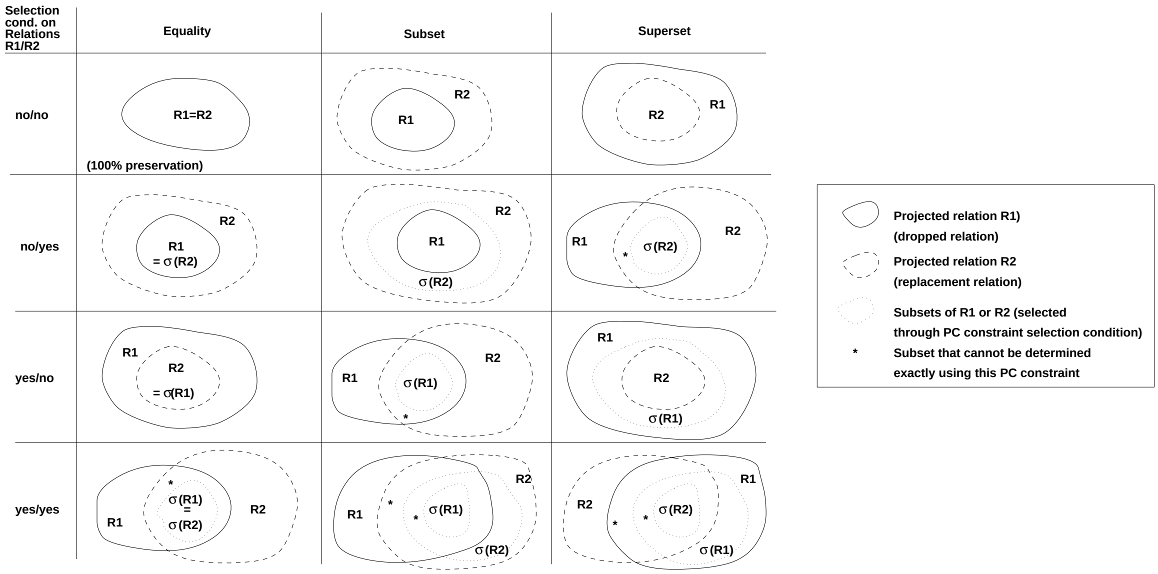

Figure 9: Determining overlapping extent sizes.

being true (i.e., no select conditions). The objects $R1$ and $R2$ represent projections of the relations mentioned in the $\mathcal{PC}$ constraint (R1 $= \pi_{A_{i_1},\dots,A_{i_k}}(R_1)$, R2 $= \pi_{A_{n_1},\dots,A_{n_k}}(R_2)$). Note that in five cases, the size of the overlap can only be estimated so the final result for the *view* overlap may have a larger statistical error. The subsets that cause these inexact results are marked by an asterisk (*) in the picture. Figure 10 summarizes the findings in an algebraic way.

Type	$\theta = \equiv$	$\theta = \subseteq$	$\theta = \supseteq$
$\mathcal{C}_{R_1} = true \wedge \mathcal{C}_{R_2} = true$ ("no/no")	$\|R1\| = \|R2\|$	$\|R1\|$	$\|R2\|$
$\mathcal{C}_{R_1} = true \wedge \mathcal{C}_{R_2} \neq true$ ("no/yes")	$\|R1\| = \sigma_{R_2}\|R2\|$	$\|R1\|$	$\approx \sigma_{R_2}\|R2\|$
$\mathcal{C}_{R_1} \neq true \wedge \mathcal{C}_{R_2} = true$ ("yes/no")	$\|R2\| = \sigma_{R_1}\|R1\|$	$\approx \sigma_{R_1}\|R1\|$	$\|R2\|$
$\mathcal{C}_{R_1} \neq true \wedge \mathcal{C}_{R_2} \neq true$ ("yes/yes")	$\approx \sigma_{R_1}\|R1\| = \sigma_{R_2}\|R2\|$	$\approx \sigma_{R_1}\|R1\|$	$\approx \sigma_{R_2}\|R2\|$

Figure 10: Estimating $|R1 \cap_\pi R2|$: Intersection size for different types of $\mathcal{PC}$ constraints.

In those cases in which the exact size of the overlapping extents cannot be determined from the $\mathcal{PC}$ constraint, the approximations compute a *minimal* value for the intersection. In a well defined information space, all information about overlaps of relations should be covered by $\mathcal{PC}$ constraints. If this is the case, only the "exact" overlap cases will occur when finding replacements.

Computation of $|V \cap_\pi V_1|$ for Example 4

With these calculations on the underlying relations, we can now try to determine the approximate size of the actual *view* on top of these relations. The size of a view can be estimated by looking at its view definition and determining how the view is computed from the underlying relations. For example, the size of our example view rewriting (19) can be estimated as

$$|V_1| \approx js_{T,S} \cdot |T| \cdot |S|$$

since the view has a join over these two tables T and S. $js_{T,S}$ is the join selectivity for a join over T and S. The size of the overlapping view extent can be estimated similarly. In our example (capability change *delete-relation*), tuples from an old relation are replaced with similar tuples from a new relation. So the size of the overlap is computed by the size of the overlap between the original and replacing relations (cf. Figure 9), joined with any other relation that appears in the view query. In our example, relation R is replaced by relation T. The size of the original view V is determined by

$$|V| \approx js_{R,S} \cdot |R| \cdot |S|$$

So the size of the overlap between the the extent of the original view V and the extent of view rewriting V_1 is approximated by:

$$|V \cap_\pi V_1| \approx js_{T,S} \cdot |R \cap_\pi T| \cdot |S|$$

The size of $R \cap_\pi T$ can be estimated as introduced above from the relation sizes and a $\mathcal{PC}$ constraint between these relations.
If no $\mathcal{PC}$ constraints are used (that is possible in the case of $\mathcal{VE}$ =‘$\approx$’), the size of the intersection of two relations cannot be determined. In this case, we use 0 (zero) as an approximation for the size of the overlapping part, since without a $\mathcal{PC}$ constraint between two relations we have to assume that these relations do not overlap.

5.4.4 Total Degree of Divergence

With the findings of this section, we now define the total degree of divergence of V_i from V as:

$$DD(V_i) = \varrho_{attr} \cdot DD_{attr}(V_i) + \varrho_{ext} \cdot DD_{ext}(V_i), \ \ where\ \varrho_{attr}, \varrho_{ext} \geq 0 \ \ and\ \varrho_{attr} + \varrho_{ext} = 1. \tag{20}$$

6 Efficiency Model: View Maintenance Cost of a Legal Rewriting

6.1 View Maintenance Basics

We assume that data content updates on the base relations, e.g., inserts or deletes of tuples to/from the base relations, take place more frequently than capability changes in the information space. Therefore, we choose to rank the legal rewritings by their *long term* view maintenance costs [10]. A legal rewriting is considered to be *preferred* if its expected view maintenance costs are low compared to other legal rewritings. We further assume that a conventional incremental view maintenance algorithm similar to the one specified in [ZGMHW95] is used to bring the view extent up-to-date right after the IS data is updated. We assume the IS data updates are sufficiently spaced from each other, so concurrent data updates are not considered in this paper. Considering concurrent updates would significantly complicate this portion of our analytical model, but we expect that it would not have a large enough impact on our findings to justify this extra effort.

Now, we briefly introduce the view maintenance algorithm used for keeping the view extent up-to-date after one data content change. The cost for multiple updates can then be computed by summing over all individual costs. For the following, we assume a view V that references relations $R_{1,0}, R_{1,1}, \ldots, R_{1,n_1}, \ldots, R_{i,1}, \ldots, R_{i,n_i}, \ldots, R_{m,1}, \ldots, R_{m,n_m}$ residing in m ISs (see Figure 11). Let $n = 1 +$

[10] The cost for recomputing the original view extent after a view re-definition is a one-time cost. Thus we do not judge the legal rewriting on this one-time view update cost.

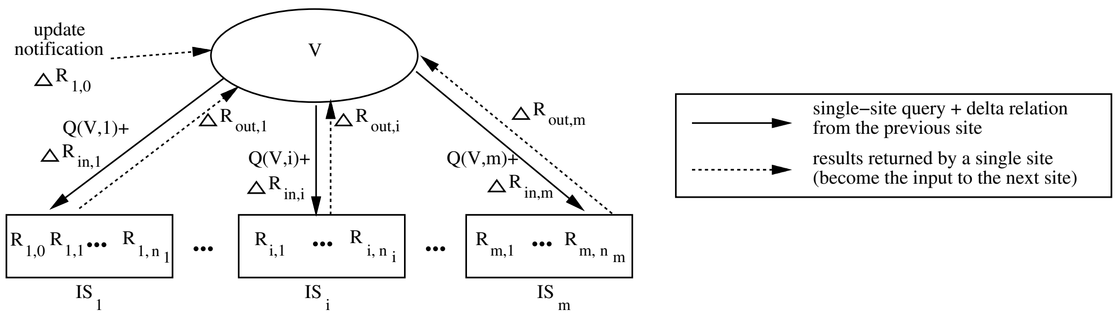

Figure 11: View Maintenance Process.

$\sum_{i=1}^{m} n_i$ be the total number of relations in the information space that are referred to in the view (that is, including the relation where the data update originated). Further assumptions are:

- the ISs are fully cooperative and are able to join their local relations with incoming delta-relations.
- the data warehouse is not doing any joins.
- the partial results are sent along to the next IS (for all ISs).
- the maintenance is done in a non-parallel way.
- the relations referenced in the view are joined in an order that does not make it necessary to again query data from an IS already visited[11].
- each source may have more relations that are not used by the view definition V, but these relations are not relevant to our computation and thus are not shown in Figure 11.

Without loss of generality, we assume $IS_1.R_{1,0}$ changes its data content, and there are n_1 other relations in the same source referenced in V. After the change is completed, IS_1 notifies EVE about the data change by sending the update information $\Delta R_{1,0}$ to the view site. Upon receiving this data update notification, EVE decides which views are affected by it. Then our view maintainer brings the view extents of these affected views up-to-date by executing the following view maintenance algorithm.

Algorithm 1 ***View-maintenance(V):***

```
begin
  1. Delta = data content update at source 1 by relation R(1,0)
  2. for (i = 1; i <= m; i++) {
  3.     view maintainer sends appropriate single site query Q(V,i) with Delta to source i;
  4.     source i sends the query results, a new Delta, back to view maintainer;
  5. }
  6. view maintainer updates the view extent of V
end
```

[11] If this is the case, we can treat such a physical IS as two or more logical ones and count it more than once in our computation, which would then generate the correct results for the cost estimates

Note that the delta relation ($\Delta R_{out,i}$) sent back by the information source IS_i to the warehouse becomes the delta relation ($\Delta R_{in,i+1}$) sent along with the single site query Q(V,i+1) to the next information source IS_{i+1} [12]. In order to compute the join at information source IS_{i+1}, the tuples of the delta relation $\Delta R_{in,i+1}$ are created as a new relation at the IS which is then joined with the local relations.

For the estimation of the view maintenance costs of the legal rewritings, similar to prior work, we assume the following database statistics:

1. The cardinality (number of tuples) of each relation R is known and denoted as $|R|$.
2. The size of each attribute $R.A$ is known as $s_{R.A}$ (and registered in the MKB). From this information, we can estimate the size of any set of attributes in a query sent to an IS and the size of the results returned by an IS.
3. The join selectivity (js) is the percentage of tuples in a relation that would join (with an equijoin condition[13]) with tuples in the other relation. For simplicity, we assume the join selectivity is a constant for any two relations across the information sources (and is also registered in the MKB).
4. We assume there is a local selection condition for each relation involved in a view definition. Similar to joins, we further assume all the operators of the local condition are equality-based (in order to have a constant local selectivity σ_{IS_i})[14].
5. $|R|$ and js do not change significantly as updates occur.
6. K = the number of bytes per physical block.

Similar to previous work [ZGMHW95], we now introduce three major cost factors (for a single data content update) for a particular legal rewriting: the number of messages exchanged, the number of bytes transferred, and the I/O cost at the local ISs. Then, we present our workload model that is used to compare the view maintenance costs of different legal rewritings.

6.2 Cost Factor Based on Number of Messages Exchanged (CF_M)

The number of messages exchanged between the information space and the view site for a single base data update, denoted as CF_M, is in the range $[0, 2m]$ (with m denoting the number of information sources involved in the view). To be more specific:

$$CF_M = \begin{cases} 0 & \text{if } m = 1 \text{ and } n_1 = 0 \\ 2 & \text{if } m = 1 \text{ and } n_1 > 0 \\ 2 \cdot (m-1) & \text{if } m > 1 \text{ and } n_1 = 0 \\ 2 \cdot m & \text{otherwise} \end{cases}$$

n_1 is the number of relations in the update-generating IS *besides* the relation where the update occured. The best case $CF_M = 0$ occurs when there is only one relation referred to in the view V (or when V is self-maintainable as discussed in [GJM96]. Self-maintainability is out of the scope of this paper, so we do not discuss it any further.). Note that when there is only one relation in IS_1 referred to in V ($n_1 = 0$), then no query needs to be sent to IS_1.

[12] Note that if there is only a single relation at IS_1 referred in V, then the view maintainer does not need to send a query to IS_1.

[13] To simplify we assume all the relations are joined by some equijoin conditions.

[14] The reason for this assumption is to keep the current dicussion simple. The model can be extended to handle all comparison operators.

6.3 Cost Factor Based on Bytes of Data Transferred (CF_T)

Considering an information space consisting of n relations $R_1, \ldots, R_n$ in m information sources $IS_1, \ldots, IS_m$, we can derive a general computation for the number of bytes transferred (assuming the algorithm described earlier in the Section 6.1). This computation assumes that one inserted/deleted tuple is sent from IS_1 to the view site, which is the initial delta relation (the first line in Equation 21). Then this delta relation is sent down to the information source IS_1 to join with other relations in IS_1 referred to in the view query (the first term in the second line), and the resulting new delta relation (the second term in the second line) is sent back to the view site. The same process iterates through all the information sources referred to in the view to build up the delta relation that contains the tuples affected by the data update.

In summary, the number of bytes transferred can thus be approximated by:

$$\begin{aligned}
CF_T &= \underbrace{s_{R_{1,0}}}_{update\ notification} \\
&+ \underbrace{s_{R_{1,0}}}_{\Delta R_{in,IS_1}} + \underbrace{\sigma_{IS_1} \cdot J_{IS_1} \cdot s_{\Delta R_{out,IS_1}}}_{\Delta R_{out,IS_1}} \\
&+ \underbrace{\sigma_{IS_1} \cdot J_{IS_1} \cdot s_{\Delta R_{out,IS_1}}}_{\Delta R_{in,IS_2}} + \underbrace{\sigma_{IS_1}\sigma_{IS_2} \cdot J_{IS_1}J_{IS_2} \cdot s_{\Delta R_{out,IS_2}}}_{\Delta R_{out,IS_2}} \\
&+ \cdots \\
&+ \underbrace{(\sigma_{IS_1} \cdot \ldots \cdot \sigma_{IS_{m-1}})(J_{IS_1} \cdot \ldots \cdot J_{IS_{m-1}}) s_{\Delta R_{out,IS_{m-1}}}}_{\Delta R_{in,IS_m}} + \underbrace{(\sigma_{IS_1} \cdot \ldots \cdot \sigma_{IS_m})(J_{IS_1} \cdot \ldots \cdot J_{IS_m}) s_{\Delta R_{out,IS_m}}}_{\Delta R_{out,IS_m}}
\end{aligned} \tag{21}$$

where s_R is the size (sum of the length of attributes in bytes) of the relation R or intermediate query result R, σ_{IS_i} is the average selectivity for the selection conditions used in the single-site query to source IS_i [15], and J_{IS_i} is the estimated size of the resulting join relation $R_{i,1} \bowtie R_{i,2} \bowtie \cdots \bowtie R_{i,n_i}$ returned by the source IS_i. That is, $J_{IS_i} \approx js^{n_i} \cdot |R_{i,1}| \cdot |R_{i,2}| \cdot \ldots \cdot |R_{i,n_i}|$, with js^{n_i} being the average join selectivity for this IS, raised to the power of the number of relations in this IS.

If all selectivities (σ), join selectivities (js), relation cardinalities ($|R|$), and tuple sizes (s) of the relations are assumed to be the same for all Rs, we can simplify the above summation as follows:

$$CF_T = 2s + 2\sum_{j=1}^{m-1} \left(\sigma^j \cdot (|R| \cdot js)^{n_R(j)} \cdot s(1 + n_R(j))\right) + \sigma^j \cdot (|R| \cdot js)^{n_R(j)} \cdot s(1 + n_R(m)) \tag{22}$$

with $n_R(k) = \sum_{i=1}^{k} n_i$.

6.4 Cost Factor Based on I/O ($CF_{I/O}$)

In this section we use the total number of estimated input and output operations performed by local ISs in order to process incremental view maintenance for each legal rewriting (in the information space) as a criterion to rank the legal rewritings. Intuitively, a legal rewriting is *preferred* if it requires less I/O-operations (resources) from the overall information space to keep its view extent up-to-date in the long run.

---

[15] Since we assume there is one local condition for each relation residing in IS_i, $\sigma_{IS_i} = \sigma_{R_{i,1}} \cdot \sigma_{R_{i,2}} \cdot \ldots \cdot \sigma_{R_{i,n_i}}$ for $1 \le i \le m$.

Let $CF_{I/O,IS_i}$ be the number of estimated I/Os at the information source IS_i. $CF_{I/O,IS_i}$ is the sum of the I/Os of the relations that reside at source IS_i, i.e., incorporating the I/O-costs of all relations at IS_i. Then the total number of I/Os in the information space, denoted as $CF_{I/O}$, is the sum of the I/Os at all m sources, i.e.,

$$CF_{I/O} = \sum_{i=1}^{m} CF_{I/O,IS_i} \tag{23}$$

$CF_{I/O}$ is influenced by many parameters, such as the number of relations referred to in the query, the tuple sizes of the relations, the cardinalities of the relations, the join selectivity factors of the joined relations, the index structures available for each join attribute, the size of local buffers available to the information sources and the view site, and the join methods available to each of the ISs. For a more detailed discussion, the reader is referred to Appendix A.

6.5 Total View Maintenance Cost for a Single Data Update

The total view maintenance cost of a view V with respect to a single data update can now be defined as:

$$Cost(V) = CF_M \cdot cost_M + CF_T \cdot cost_T + CF_{I/O} \cdot cost_{I/O} \tag{24}$$

where $cost_M, cost_T$, and $cost_{I/O}$ are the unit prices for sending a message, transferring a data block, and performing a disk I/O, respectively.

6.6 Workload Model

On the one hand, different legal rewritings of a view may make use of information from different information sources. And, on the other hand, a particular data update only affects the views that refer to this data. Therefore, it is not sufficient to compare various legal rewritings with respect to the same set of data updates. Instead, we have the following choices for a workload model for our system:

M1. We assume the number of updates of a relation is proportional to the number of tuples in the relation. This assumption is equivalent to data updates happening to p percent of a relation's tuples within a given time frame. So a view V is facing a total of $p \cdot \sum_{i=1}^{m}(|R_{i,1}| + |R_{i,2}| + \cdots + |R_{i,n_i}|)$ updates per time unit [16].

M2. We assume each relation R has a constant number of updates per time unit u, independent of the size or the location of R. If an information source IS_i has n_i of its relations used by V, then V is facing a total of $u \cdot \sum_{i=1}^{m} n_i$ data updates per time unit.

M3. We assume each information source is facing a constant number of data updates u. In this case, view V would face a total of $m \cdot u$ updates.

M4. We assume each legal rewriting is affected by a constant number u (but different sets) of data content updates. In this case, we would have to make further assumptions as to the distribution of these updates over all ISs, e.g., we could assume that data updates happen equally for each view element in the view.

[16] Remember that we have m ISs and each IS_i has n_i relations for $1 \leq i \leq m$.

Note that after selecting a specific workload model, we can compute the total view maintenance costs, COST(V_i), for the updates within a certain time unit. If we assume that there are k legal rewritings for an affected view, the total cost of legal rewriting V_i can be normalized as follows:

$$COST^*(V_i) = \frac{COST(V_i) - min_{1\leq j\leq k}(COST(V_j))}{max_{1\leq j\leq k}(COST(V_j)) - min_{1\leq j\leq k}(COST(V_j))} \tag{25}$$

This gives us a view maintenance cost between 0 and 1 that we can trade off against the view quality (Section 5).

6.7 Overall Efficiency of a Legal Rewriting

The *overall efficiency* of a legal rewriting can now be computed as

$$QC(V_i) = 1 - (\varrho_{quality} \cdot \mathcal{DD}(V_i) + \varrho_{cost} \cdot COST^*(V_i)) \tag{26}$$

with $0 \leq \varrho_{quality}, \varrho_{cost} \leq 1$ and $\varrho_{quality} + \varrho_{cost} = 1$. With both quality and cost normalized, this number will be between 0 and 1. An efficiency of 0 means a legal rewriting that preserves no information (which renders it "useless"), An efficiency of 1 would identify a "perfect" legal rewriting preserving the complete view interface and all tuples at no cost. Since the incremental view maintenance cost will never be zero, an efficiency of 1 can only be reached if the costs are given no consideration, i.e., if $\varrho_{cost} = 0$.

7 Experimental Evaluation

We now set out to verify the validity of our proposed *QC-Model* and get a deep understanding of the interplay between quality and maintenance costs through a number of experiments. These experiments were conducted in the context of our *EVE* system, i.e., all rewritings of a view were being generated by our synchronization algorithm [LNR97a, NLR98]. However, as our system is not fully instrumented with our QC-Model yet, the reported measures are computed using our algebraic findings instead of measuring them directly from an implemented system. This means, that in this section, we evaluate the *QC-Model* rather than the complete *EVE* system.

7.1 Experiment 1: "Survival" of a View

The number of capability changes a view can "survive"[17] depends on its evolution parameter settings and the degree of data redundancy in the information space. In general, when the evolution parameters are set to *dispensable* and *replaceable* and when data is amply duplicated in the information space, then the view has a higher chance to survive in an evolving environment. As an example, we assume an attribute $R.A$ referred to in a view is replaceable. When $R.A$ is deleted from its site, *EVE* will be able to replace R with its replica from another information site. On the other hand, if $R.A$ is non-replaceable, then even if there is a replica of $R.A$ in the information space, *EVE* will not be able to replace $R.A$ in the future. Therefore, when salvaging an affected view definition, *EVE* gives a higher priority to view components with their evolution parameters set to *replaceable*. Employing this strategy, *EVE* has a higher chance to keep the view alive in the future. Theoretically, if there is a high number of data replicas (i.e., larger than the number of relevant *delete*–capability-changes) in the information space and the view components are replaceable, then a view could be kept alive indefinitely. This observation is supported by the following experiment in our *EVE* system. Let's assume a view is defined as follows:

[17] The view can be evolved by our algorithm and stay valid after these capability changes.

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_0\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \textsf{AS} \\ \textsf{SELECT} & R.A\ (\mathcal{AD} = true, \mathcal{AR} = true), \\ & R.B\ (\mathcal{AD} = true) \\ \textsf{FROM} & R\ (\mathcal{RR} = true) \end{array} \tag{27}$$

We now assume that $R.A$ is deleted by its information provider. Further, we assume two relations, $S(A, C)$ and $T(A, D)$, in other information sources are related to the relation R. We have two $\mathcal{PC}$ constraints defined in the MKB: $\mathcal{PC}_{R,S} = (\pi_A(R) \equiv \pi_A(S))$ and $\mathcal{PC}_{R,T} = (\pi_A(R) \subseteq \pi_A(T))$.
Then, there are three alternative ways to evolve V_0:

- One possible solution is to drop the attribute $R.B$, since it is nonreplaceable but dispensable, and then replace $R.A$ with an appropriate attribute from either relation S or T. Using this strategy, we get the two legal rewritings V_1 and V_2.

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_1\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \textsf{AS} \\ \textsf{SELECT} & S.A\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \textsf{FROM} & S\ (\mathcal{RR} = true) \end{array} \tag{28}$$

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_2\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \textsf{AS} \\ \textsf{SELECT} & T.A\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \textsf{FROM} & T\ (\mathcal{RR} = true) \end{array} \tag{29}$$

- Another possible solution is to simply drop the attribute $R.A$ from V_0, since it is dispensable. We get V_3 as a legal rewriting:

$$\begin{array}{ll} \textsf{CREATE VIEW} & V_3\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \textsf{AS} \\ \textsf{SELECT} & R.B\ (\mathcal{AD} = true) \\ \textsf{FROM} & R\ (\mathcal{RR} = true) \end{array} \tag{30}$$

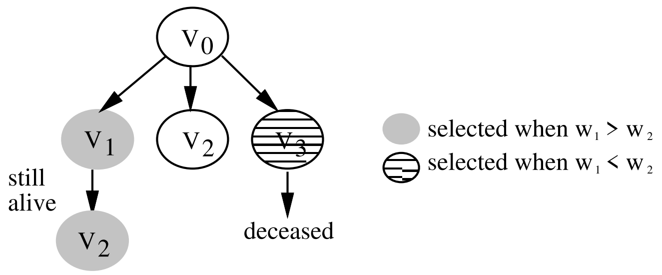

Figure 12: The Life Span of Legal Rewritings.

Ignoring the view extent quality factor for the time being, if $w_1 > w_2$ (that is, if we give a larger weight to replaceable attributes than to non-replaceable attributes), then EVE would choose V_1 and V_2 over V_3 (attribute A is in category 1, and attribute B is in category 2; see Section 5.2 for a definition of these categories and weight factors). On the other hand, if $w_2 > w_1$, then EVE may choose V_3 over the other two legal rewritings (see Figure 12). Assume EVE chooses V_1 to rewrite V_0. If S gets deleted later, then the view can still be evolved by rewriting it into V_2. On the other hand, if V_3 were to be chosen for this first rewriting, then any further capability change in the information space will cause the view to become undefined. This supports the default setting of $w_1 > w_2$ for the QC-Model .

7.2 Experiment 2: Ratio between Relations and ISs

In this experiment, we study the relationships between the number of ISs involved in a view and the view maintenance cost attributed to the view. We conduct this experiment by varying the number of ISs involved in a view, while fixing all other parameter settings, such as the number of relations referred in a view, the selectivity and the join selectivity. The main purpose of this experiment is to investigate whether it is beneficial to retrieve all information from as few sites or as many sites as possible.

For this experiment, we assume that six relations are used in the view. That implies there are at most six information sources involved in the view (each relation resides in a different information source) and at least one information source (all six relations are in one site). The system parameters for our experiment are summarized in Table 1.

Name	Meaning	Default Value
n	Total number of relations in the information space	6
m	Total number of information sites referred in a view	{1,6}
$\lvert R_i \rvert$	Cardinality of the relation R_i, for all i	400
s_{R_i}	Tuple size of the relation R_i, for all i	100
σ	Selectivity of a local condition	0.5
js	Join selectivity factor	0.005
bfr	Blocking factor	10

Table 1: List of System Parameters.

# Sites (m)	Relation Distribution (# of relations in each site)
1	(6)
2	(1,5), (2,4), (3,3), (4,2), (5,1)
3	(1,1,4), (1,2,3), (1,3,2), (1,4,1), (2,1,3), (2,2,2), (2,3,1), (3,1,2), (3,2,1), (4,1,1)
4	(1,1,1,3), (1,1,2,2), (1,1,3,1), (1,2,1,2), (1,2,2,1), (1,3,1,1), (2,1,1,2), (2,1,2,1), (2,2,1,1), (3,1,1,1)
5	(1,1,1,1,2), (1,1,1,2,1), (1,1,2,1,1), (1,2,1,1,1), (2,1,1,1,1)
6	(1,1,1,1,1,1)

Table 2: Relation Distribution across Information Sources.

With this experimental setting, there are six possible scenarios corresponding to the six rows of Table 2. Row 1 for example indicates that the data for the view is retrieved from a single IS, row 2 indicates it is from two ISs, and so on. Within each scenario, the relations may be distributed differently among the ISs. For example, when there are two information sources involved in a view, one relation can be retrieved from one IS and the other five relations from another IS – represented by (1,5) on the second row; two relations from one IS and the remaining four relations from another IS – represented by (2,4); and so on. Case (3,3) exhibits the most even distribution, since there is an equal number of relations in each IS. Cases (1,5) and (5,1) are the most skewed relation distributions. Note that (1,5) is different from the Case (5,1), because in this experiment we assume that data updates are initiated at the first IS. Therefore, (1,5) and (5,1) may incur different view maintenance costs. We first compute the view maintenance cost for every relation distribution in each scenario, then we compute the average view maintenance cost for that scenario. The results are shown in Figure 13.

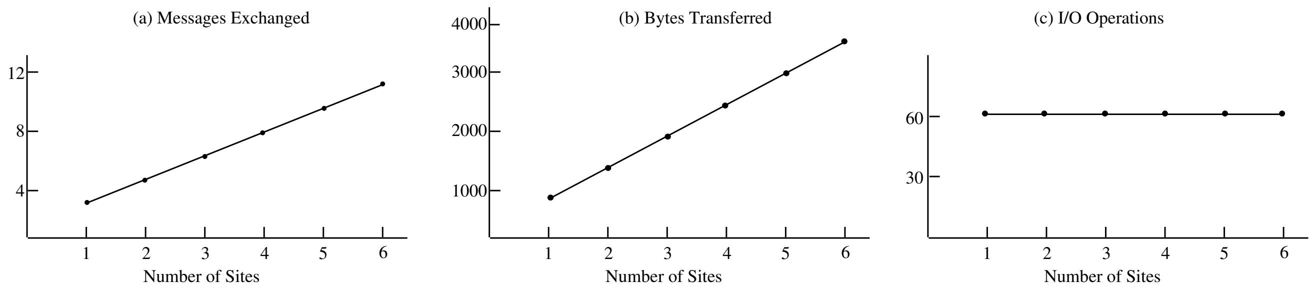

Figure 13: The Relationships between Numbers of ISs in the View and Three View Maintenance Cost Factors.

As shown in Figure 13, the number of messages exchanged and the number of bytes transferred between the view site and the information space both increase when the number of information sources involved in a view increases. That is, the view maintenance cost of a single data update increases when the number of information sources involved in a view goes up.

7.3 Experiment 3: Relation Distribution

Now we look at the previous experiment from another angle. Within each scenario (i.e., with a fixed number of information sources involved in a view) we study whether the relation distributions affect the view maintenance costs. Namely, we study whether the view maintenance costs are lower when the relations referred to in a view are evenly (or *uniformly*) distributed among the information sources or when they are unevenly distributed. The results are summarized in Figure 14. The graphs show the number of bytes transferred for three particular join selectivities. For each setting of js, we compare possible distributions of 6 relations in 2,3, and 4 information sources, respectively. The possibilities are listed in Table 2, in the chart we group the cases (i.e., $(1,5) \equiv (5,1)$).

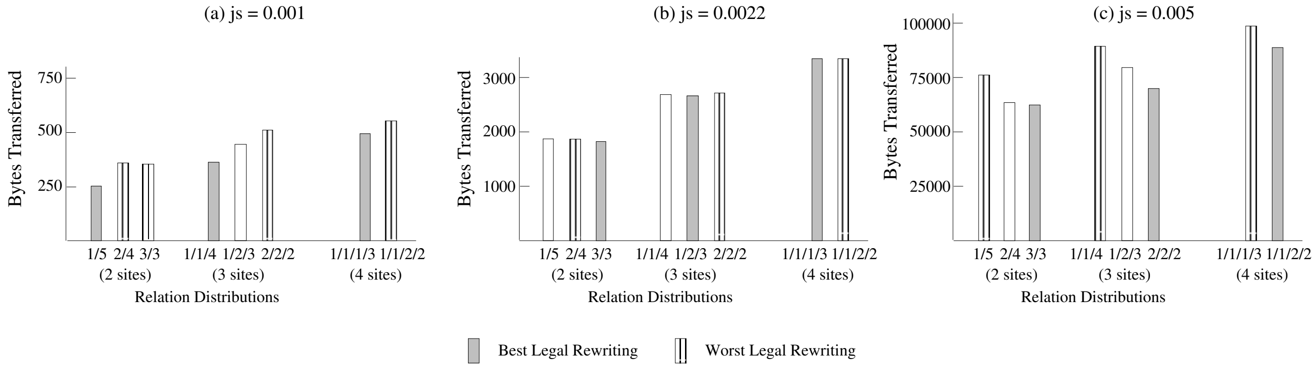

Figure 14: The Relationships between Evenness of Relation Distributions and View Maintenance Costs.

We find no apparent relationship between the view maintenance costs and the evenness of the relation distribution. Our experimental results show that when the number of tuples in the intermediate results during a query update grows fast (i.e., we have a relatively high average join selectivity js in our information space), it is beneficial to have a more evenly distributed relation allocation (see Figure 14(c) with $js = 0.005$). On the other hand, when the number of joined tuples of the delta relation does not grow as fast, it may be advantageous to have a more skewly distributed information space (see Figure 14(a) when $js = 0.001$). There are cases where the evenness of the relation distributions does not have a clear impact. However, as stated in Experiment 2, we have found a dependency of the view maintenance costs on the number of ISs involved. Therefore, minimizing the number of ISs involved in a view rewriting should have a higher priority over choosing a certain relation distribution among the ISs.

7.4 Experiment 4: Relation Cardinality

In this experiment, we study the relationship between the cardinalities of the substituted relations and the *overall efficiency* of the legal rewritings. We conduct this experiment by varying the cardinalities of the substituted relation while keeping all other parameter settings the same. Let us assume a view V is defined as follows:

$$\begin{array}{ll} \text{CREATE VIEW} & V\ (\mathcal{VE} = `\approx\text{'})\ \text{AS} \\ \text{SELECT} & \cdots,\ R_2.A\ (\mathcal{AR} = true), R_2.B\ (\mathcal{AR} = true), R_2.C\ (\mathcal{AR} = true) \\ \text{FROM} & R_1, R_2\ (\mathcal{RR} = true) \\ \text{WHERE} & \cdots \end{array} \tag{31}$$

Let us assume that relation R_2 is deleted by its information provider, and that there are five relations $S_1, \cdots, S_5$ in the information space that are identified by the view synchronizer to be appropriate substitutes for R_2. Five new views, $V_1 \ldots V_5$ can be defined that are formed by replacing relation R with the respective relation R_n. The cardinalities of R_2 and the substitute relations for our experiment are summarized in Table 3.

Site Name	Relation Name	Cardinality
IS_1	$R_2(A, B, C)$	4000
IS_2	$S_1(A, B, C)$	2000
IS_3	$S_2(A, B, C)$	3000
IS_4	$S_3(A, B, C)$	4000
IS_5	$S_4(A, B, C)$	5000
IS_6	$S_5(A, B, C)$	6000

Table 3: Cardinalities of R_2, $S_1, \cdots, S_5$.

We further assume that the following inter-relationships among these relations hold true: Relation S_1 is contained in relation S_2, denoted by a $\mathcal{PC}$ constraint: $\mathcal{PC}_{S_1,S_2} = (S_1 \subseteq S_2)$, S_2 in turn is contained in S_3, S_3 is equivalent to the deleted relation R_2, S_3 is contained in S_4, and S_4 contained in S_5 (i.e., $S_1 \subseteq S_2 \subseteq S_3 = R_2 \subseteq S_4 \subseteq S_5$). Therefore, replacing R_2 with S_i, for $1 \leq i \leq 5$, we get five alternate yet legal rewritings with different view extents and view maintenance costs[18]. Setting the system parameters to $w_1 = 0.7$, $w_2 = 0.3$, $\varrho_{D1} = 0.5$, $\varrho_{D2} = 0.5$, $\varrho_{attr} = 0.7$, $\varrho_{ext} = 0.3$, $cost_M = 0.1$, $cost_T = 0.7$, $cost_{I/O} = 0.2$, $\varrho_{quality} = 0.9$, and $\varrho_{cost} = 0.1$, we get the metrics

[18] Note that $\mathcal{VE} = `\approx\text{'}$ for this view as given in Equation 31

of quality and cost that are summarized in Table 4 (see also Case 1 in Figure 15). The other two cases in Table 4 and Figure 15 are obtained with ($\varrho_{quality} = 0.75$, $\varrho_{cost} = 0.25$) and ($\varrho_{quality} = 0.5$, $\varrho_{cost} = 0.5$), respectively.

Rewriting	DD_{attr}	DD_{ext}	DD	Cost (Normalized Cost)	$QC(V_i)$	Rating
V_1	0	0.25	0.075	842.3 (0)	0.9325	3
V_2	0	0.13	0.0375	1193.3 (0.25)	0.94125	2
V_3	0	0.00	0.00	1544.3 (0.5)	0.95	1
V_4	0	0.10	0.027	1895.3 (0.75)	0.898	4
V_5	0	0.17	0.045	2246.3 (1)	0.855	5

Table 4: Ranking of Legal Rewritings for Experiment 4. (Detailed Data for Case 1)

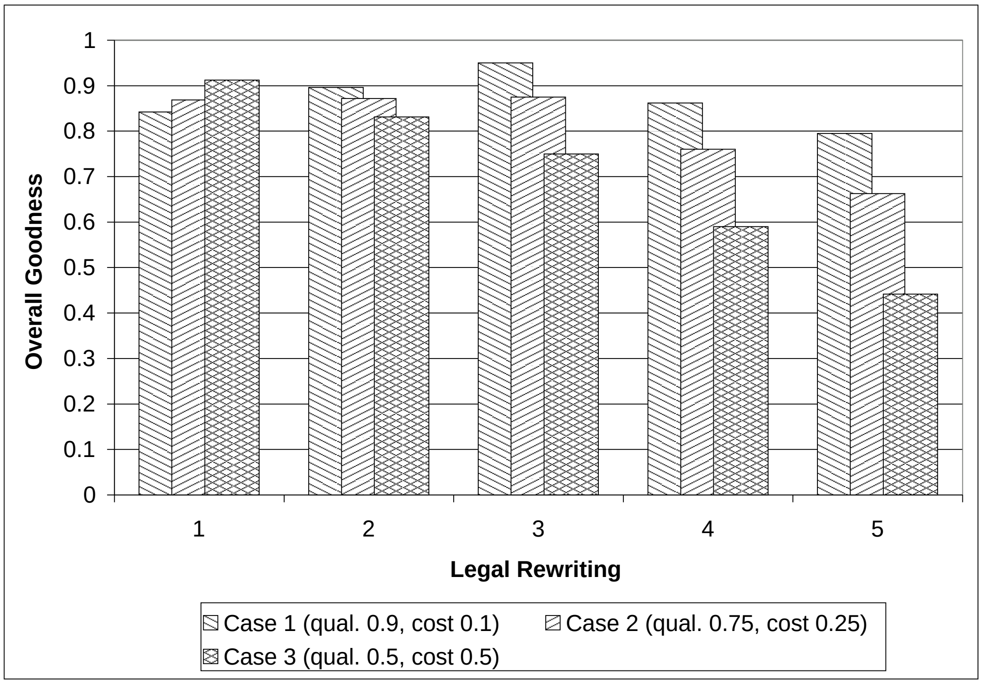

Figure 15: Results of Assessing Legal Rewritings for Experiment 4.

In Section 5 we postulated that the degree of divergence $DD(i)$ for a view rewriting V_i will be large for a relation whose size is very different from the size of the original relation, and vice versa. The cost of a legal rewriting will be larger, all other factors equal, with a growing size of the replaced relation(s). Trading off these two factors against each other will therefore lead to different results depending on how the trade-off parameters are set. Our experiment validates these findings.

For example, when the parameters are set to ($\varrho_{quality} = 0.9$, $\varrho_{cost} = 0.1$, Case 1), the QC-Model chose legal rewriting V_3 over the other four legal rewritings. Here, we give a high priority to the quality of the rewriting, which is best when the replacing relation comes as close as possible to the original relation, which is the case in legal rewriting V_3. The graph depicted in Figure 15 shows that the overall efficiency increases from legal rewriting V_1 until V_3 (because the size of the replacing relation approaches the size of the original relation), then becomes worse as the difference between the relation sizes grows bigger.

However, in Case 3, with ($\varrho_{quality} = 0.5$, $\varrho_{cost} = 0.5$), the cost has a larger impact on the overall efficiency of the legal rewriting. Since the cost is continuously increasing as the replacing relations get bigger (i.e., from legal rewriting V_1 to V_5), the overall efficiency of the rewritings decreases, so rewriting V_1 (with the smallest replacing relation) is chosen by our view synchronizer. Even in Case 2, the influence of the cost on the total result is large enough for V_1 to be selected as best legal rewriting.

Two observations we made from Figure 15 are:

- If we focus our attention on the legal rewritings V_3, V_4, and V_5 (labeled 3, 4, and 5 in Figure 15, rows 3 to 5 in Table 4), we can see that these rewritings are obtained by substituting the deleted relation R_2 by a *superset* relation. Among these three legal rewritings, V_3 is always ranked highest among the three in various parameter settings. This is because the degrees of divergence (fourth column in Table 4, labeled *DD*) as well as the view maintenance costs (fifth column, labeled Cost) go up when the cardinalities of the replaced relations go up. For these cases, the trade-off parameters have no influence on what rewriting is selected to be best. A consequence is that if we have only superset replacements at our disposal, the replacement that is closest to the original in terms of the relation size is also the smallest replacement and will always rank best among legal rewritings.

- If we focus on the legal rewritings V_1, V_2, and V_3 (labeled 1, 2, and 3 in Figure 15, rows 1 to 3 in Table 4), these rewritings are obtained by replacing the deleted relation R_2 with a *subset* relation. The degrees of divergence of the rewritings go down as the sizes of the replacement relations go up (column four in the table), but the view maintenance cost of the legal rewritings increases with the cardinality of the substituted relations (column five). Therefore, the overall efficiency of these rewritings depends on the trade-off parameters. For Case 1, V_3 is the best among the three. For Cases 2 and 3, i.e, when the view maintenance costs are weighted heavier, then V_1 is ranked higher by the efficiency model.

7.5 Experiment 5: Workload Models

While our previous experiment computed the view maintenance cost for a single data update, Experiment 5 considers the effect of the four different workload models M1, M2, M3 and M4 as defined in Section 6.6 for computing the view maintenance cost within a period of time.

Workload model M1 assumes a number of updates which is proportional to the size of a relation. In order to evaluate the influence of this workload model on our findings, we look again at Experiment 4, which compares five different legal rewritings with different relation sizes (cardinalities). With the cardinalities of the replacing relations growing from rewriting V_1 through V_5 (i.e., from left to right in Figure 15), the number of updates in this workload model grows proportionally. As an example, we assume a ratio of 1 update per 100 tuples. This gives us the situation in Table 5.

We observe that a workload model only influences the *cost* (and not the *quality*) of a legal rewriting, and in the case of M1, the influence is proportional to the relation size. However, since our model normalizes the cost factor before combining it into the overall efficiency measure, both the normalized cost factors and hence the final efficiency values are unchanged.

The workload model M2 assumes a constant number of updates per relation, i.e., a total number of updates proportional to the number of relations in a view. As there is no direct correlation between the number of relations in a view and the quality of the view, a general evaluation would not be very meaningful. However, assuming

Rewriting	DD	Cost	# of updates	Normalized Cost	$QC(V_i)$	Rating
V_1	0.075	842.3	20	0	0.9325	3
V_2	0.0375	1193.3	30	0.25	0.94125	2
V_3	0.00	1544.3	40	0.5	0.95	1
V_4	0.027	1895.3	50	0.75	0.898	4
V_5	0.045	2246.3	60	1	0.855	5

Table 5: Ranking of Legal Rewritings for Experiment 5. (Workload Model M1)

we could find other rewritings with the same quality but with a different number of relations, then the workload model M2 would encourage us to pick a legal rewriting with the fewest relations possible as the cost factor would be most reduced.

The workload model M3 assumes that there are n updates per information source per time unit. That is, if a view is specified over m information sources, then there is a total of $m \cdot n$ updates for the view per time unit. Extending Experiment 2, the results obtained from this experiment are shown in Table 6 and in Figure 16.

Rewriting	# sites	# updates	CF_M	CF_T	$CF_{I/O}$
V_1	1	10	30	8000	310
V_2	2	20	92	27200	620
V_3	3	30	186	57600	930
V_4	4	40	312	99200	1240
V_5	5	50	470	152000	1550
V_6	6	60	660	216000	1860

Table 6: Assessments of Legal Rewritings for Experiment 5.

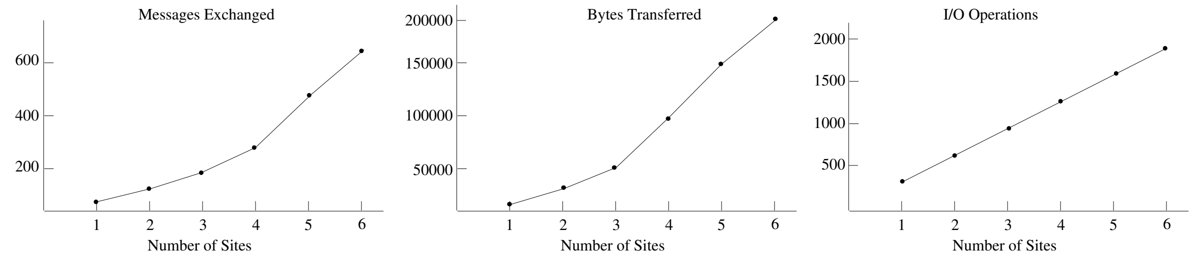

Figure 16: Results of Assessing Legal Rewritings for Experiment 5.

Under this experimental setup, our efficiency model would favor a legal rewriting that has the smallest possible number of information sources involved in the view definition to replace the affected view definition. Under this workload model, a small number of information sites has the advantage of a small number of updates, plus the overall efficiency is also better for rewritings with a smaller number of ISs.

Workload model M4 (i.e., having a fixed number of data updates for each legal rewriting) gives the same results as when considering a single data update, since a fixed number is multiplied with the single data update view maintenance cost for each scenario (as discussed in Experiment 2). Hence, it need not be considered any further here.

7.6 Heuristics

Based on the findings in Sections 5 and 6, we identify some heuristics that can help a view synchronizer to pick a good legal rewriting without having to compute all possible legal rewritings for a certain capability change.

As Equation (21) suggests and we now confirmed with Experiments 2 and 3, a view optimizer would prefer a legal rewriting with a smaller number of information sources and with relations with smaller cardinalities.

Experiment 4 supports the following heuristic: A view optimizer should choose a legal rewriting whose view component replacement is as close as possible to the original view component in terms of size. For example, when a relation R is deleted from its site, if two relations S and T are both legitimate replacements for R and we know $T \supseteq S \supseteq R$, then we would select relation S as the replacement since its size is closer to the size of the original relation.

Experiment 5 supports some findings on heuristics for different workload models (cf. also Section 6.6). For the workload model M1 (number of updates proportional to relation size), we would prefer a legal rewriting that refers to smaller relations, i.e., we would use the smallest relation that provides a satisfactory amount of information to the view user. For the models M1 and M2 (constant number of updates per relation), and M3 (constant number of updates per IS), we would aim to minimize the number of information sites referred in a view. Even in workload model M4 (globally constant number of updates for a legal rewriting) with the number of information sites fixed, we would minimize the number of joins or the number of primitive clauses in the WHERE clause.

Assuming that data updates are equally likely to occur at each relation (model M2), we can find an intuitive way to minimize the *number of messages* exchanged. All other parameters equal, the number of messages exchanged between the information space and the view site is minimized when the number of information sources referred in V is minimized, since no messages have to be sent between information sources when relations are located in the same IS. That is, a legal rewriting is chosen over other legal rewritings if the number of information sources involved in its view definition is smaller.

Lastly, we would choose a legal rewriting with a smaller number of relations referred to in the FROM clause, e.g., even if only one attribute is deleted from a relation R, we would replace R entirely if an appropriate relation can be found that already participates in the view definition (this is valid for all workload models).

8 Conclusion

View synchronization refers to the new and important problem of how to maintain views in dynamic distributed information systems [RLN97]. These issues become important as more and more diverse and autonomous database systems are incorporated into large data warehouses. Local schema changes at information sources participating in a data warehouse will generally cause a view in the warehouse to become invalid. This problem has been addressed by our previous work on the *EVE*-project [LNR97a, NLR97, LNR97b, NLR98, LKNR98].

In this work, we now focussed on performance issues raised by view synchronization. Since view evolution under capability changes of underlying data sources will generate a large number of possible *rewritings* for an original view query, it is necessary to compare these rewritings and identify the *best* solution to maintaining a view. A novel measure of *efficiency* is introduced in this paper that explores the two dimensions of *quality* and *cost* and leads to the definition of the QC-Model . This model can be used to establish a ranking among alternate legal query rewritings for an affected view definition. It turns out that a ranking is possible among seemingly incomparable solutions using the goodness model we developed, and that it is feasible to introduce *parameters* to trade off quality against cost (and also sub-dimensions of either against each other).

Since the number of query rewritings is potentially large, we also discussed heuristics that can be used to prune the search space for the best legal rewriting for a given view query, information space, and capability change. For instance, it is always preferable to select a legal rewriting that refers to a minimal number of information sources. Also, a rewriting that replaces a dropped relation with a relation similar in size is always preferred. The experiments that we ran using our system and varying different parameters of the information space (database size, number of information sources, "strictness" of view query definition in E-SQL) support our findings.

A first prototype of the EVE system is fully functional, and has been demonstrated at the IBM technology showcase during the CASCON '97 conference [LNR97a]. The results of this paper are currently being incorporated into our EVE prototype system which had previously simply picked the first legal view rewriting it discovered and not necessarily the best one.

Once the QC-Model has been implemented as evaluation module in the EVE system, we plan to instrument our view synchronizer tool to associate an efficiency ranking with all generated rewriting solutions. Other future work may be to conduct experimental studies to compare the cost portion of our QC-Model with the actual costs encountered by our system for incremental view maintenance. Lastly, an extension and elaboration of the heuristics identified in this current work may lead to the development of a novel heuristic view synchronization algorithm that instead of first generating all rewriting solutions and then ranking them, would be able to discard some of the search space early on.

Acknowledgments. The authors would like to thank students at the Database Systems Research Group at WPI for their interactions and feedback on this research. In particular, we are grateful to Yong Li and Xin Zhang for implementing several of the major components of the EVE system, including the MKB, the VKB, and the SVS algorithm.

References

[AAS97] D. Agrawal, A. El Abbadi, and A. Singh. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[BLT86] J. A. Blakeley, P.-E. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. *Proceedings of SIGMOD*, pages 61–71, 1986.

[CKL+96] L.S. Colby, A. Kawaguchi, D.F. Lieuwen, I.S. Mumick, and K.A. Ross. Supporting Multiple View Maintenance Policies. *AT&T Technical Memo*, 1996.

[CKP95] S. Chaudhuri, R. Krishnamurthy, and S. Potamianos. Optimizing Query with Materialized Views. In *Proceedings of IEEE International Conference on Data Engineering*, 1995.

[CTL+96] L.S. Colby, T.Griffin, L.Libkin, I.S.Mumick, and H.Trickey. Algorithms for Deferred View Maintenance. In *Proceedings of SIGMOD*, pages 469–480, 1996.

[GJM96] A. Gupta, H.V. Jagadish, and I.S. Mumick. Data Integration using Self-Maintainable Views. In *Proceedings of International Conference on Extending Database Technology (EDBT)*, 1996.

[GM95] A. Gupta and I.S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing*, 18(2):3–19, 1995.

[GMR95] A. Gupta, I.S. Mumick, and K.A. Ross. Adapting Materialized Views after Redefinition. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 211–222, 1995.

[GMS93] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining Views Incrementally. In *Proceedings of SIGMOD*, pages 157–166, 1993.

[JK84] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, pages 111–152, 1984.

[LKNR98] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Data Warehousing Evolution: Trade-offs between Quality and Cost. Technical Report WPI-CS-TR-98-2, Worcester Polytechnic Institute, Dept. of Computer Science, 1998.

[LMS95] A.Y. Levy, A.O. Mendelzon, and Y. Sagiv. Answering Queries Using Views. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 95–104, May 1995.

[LNR97a] A. J. Lee, A. Nica, and E. A. Rundensteiner. Keeping Virtual Information Resources Up and Running. In *Proceedings of IBM Centre for Advanced Studies Conference CASCON97, Best Paper Award*, pages 1–14, November 1997.

[LNR97b] A. J. Lee, A. Nica, and E. A. Rundensteiner. The EVE Framework: View Evolution in an Evolving Environment. Technical Report WPI-CS-TR-97-4, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[LRU96] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external processors. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 227–237, Montreal, Canada, 3–5 June 1996.

[MD96] M. Mohania and G. Dong. Algorithms for Adapting Materialized Views in Data Warehouses. *International Symposium on Cooperative Database Systems for Advanced Applications*, December 1996.

[NLR97] A. Nica, A.J . Lee, and E. A. Rundensteiner. View Synchronization with Complex Substitution Algorithms. Technical Report WPI-CS-TR-97-8, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[NLR98] A. Nica, A. J. Lee, and E. A. Rundensteiner. The CVS Algorithm for View Synchronization in Evolvable Large-Scale Information Systems. *To appear in Proceedings of International Conference on Extending Database Technology (EDBT'98)*, Valencia, Spain, March 1998.

[QW97] D. Quass and J. Widom. On-Line Warehouse View Maintenance. In *Proceedings of SIGMOD*, pages 393–400, 1997.

[RJB89] V. V. Raghavan, G. S. Jung, and P. Bollmann. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Office Information Systems*, pages 205–229, July 1989.

[RLN97] E. A. Rundensteiner, A. J. Lee, and A. Nica. On Preserving Views in Evolving Environments. In *Proceedings of 4th Int. Workshop on Knowledge Representation Meets Databases (KRDB'97): Intelligent Access to Heterogeneous Information*, pages 13.1–13.11, Athens, Greece, August 1997.

[SDJL96] D. Srivastava, S. Dar, H.V. Jagadish, and A.Y. Levy. Answering Queries with Aggregation Using Views. In *International Conference on Very Large Data Bases*, pages 318–329, 1996.

[vdBK94] C. A. van den Berg and M.L. Kersten. An Analysis of a Dynamic Query Optimization Schema for Different Data Distributions. In J. C. Freytag, D. Maier, and G. Vossen, editors, *Query Processing for Advanced Database Systems*, chapter 15, pages 449–473. Morgan Kaufmann Pub., 1994.

[Wid95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of International Conference on Information and Knowledge Management*, pages 25–30, November 1995.

[ZGMHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proceedings of SIGMOD*, pages 316–327, May 1995.

[ZGMW96] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *International Conference on Parallel and Distributed Information Systems*, December 1996.

[ZWGM97] Y. Zhuge, J. L. Wiener, and H. Garcia-Molina. Multiple View Consistency for Data Warehousing. In *Proceedings of IEEE International Conference on Data Engineering*, pages 289–300, 1997.

A Cost Factor Based on I/O ($CF_{I/O}$)

We make the following simplification assumptions for computing the number of I/O operations: (clustered or non-clustered) indexing is available for all the joined attributes, ample memory is available in each site, and each query optimizer at the information sites are able to select the best plan of whether to use indexing to retrieve the joined tuples or retrieve the entire relation, i.e., when the number of tuples in the delta relation is greater than the number of I/Os to retrieve the entire relation, the query optimizer will retrieve the entire relation. The number of I/Os required to retrieve the entire relation R_i is

$$IO_i = \left\lceil \frac{|R_i|}{bfr_{R_i}} \right\rceil . \tag{32}$$

with bfr_{R_i} being the blocking factor on relation R_i.

In order to compute the number of I/Os, we consider an update in relation $IS_1.R_{1,0}$. The tuple value is used to retrieve the joined tuples from the other relations. Depending on the number of matching tuples and if tuples can be retrieved in clusters, the number of I/O-operations for this join is within the boundaries

$$IO_i = \left[min\left(\left\lceil \frac{|R_i|}{bfr_{R_i}} \right\rceil , js^{i-1} \cdot \prod_{j=1}^{i-1} |R_j| \cdot \left\lceil \frac{js \cdot |R_i|}{bfr_{R_i}} \right\rceil \right) , min\left(\left\lceil \frac{|R_i|}{bfr_{R_i}} \right\rceil , js^{i} \cdot \prod_{j=1}^{i} |R_j| \right) \right] \tag{33}$$

where $1 \leq i \leq n$ and n is the total number of relations referred in the view definition V besides the updated relation R_0.

The total number of I/Os in order to bring the view extent up-to-date is: $\sum_{i=1}^{n} IO_i$.