ON OPTIMAL CONSTRAINT DECOMPOSITION, MONITORING, AND MANAGEMENT IN DISTRIBUTED ENVIRONMENTS

by

Samuel E. Varas G.

A Dissertation Submitted
to the Graduate Faculty
of
George Mason University
in Partial Fulfillment of
the Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

_____	Larry Kerschberg, Dissertation Director
_____	Alex Brodsky, Dissertation Director
_____	Daniel Barbará
_____	Daniel Menascé
_____	Edgar H. Sibley
_____	Carl M. Harris, Associate Dean for Graduate Studies and Research
_____	Lloyd J. Griffiths, Dean, School of Information Technology and Engineering

Date: _____ Summer, 1998
George Mason University
Fairfax, Virginia

On Optimal Constraint Decomposition, Monitoring, and Management in Distributed Environments

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy Information Technology at George Mason University.

By

Samuel E. Varas G.
B.S. Industrial Engineering, University of Chile, Chile, 1989
M.S. Industrial Engineering, University of Chile, Chile, 1989

Directors: Dr. Larry Kerschberg and Dr. Alex Brodsky
School of Information Technology and Engineering

Summer 1998
George Mason University
Fairfax, Virginia

Dedication

I would like to dedicate this work to my parents, Carmen and Samuel, whom always taught me to give my best effort, and to my wife Paula, and our daughter Paula Antonia who understood me and shared their life with me.

Acknowledgments

I would like to thank my advisors Dr. Larry Kerschberg and Dr. Alex Brodsky for their support and exceptional guidance and encouragement to understand new paradigms. I would also like to thank the other members of my dissertation committee, Dr. Daniel Barbará, Dr. Daniel Menascé, and Dr. Edgar H. Sibley for their helpful comments and support during the research and writing of this dissertation.

I would like to thank the Industrial Engineering Department, of the Physical Sciences and Mathematics School, of the University of Chile for its support. I would also like to thank the Government of Chile for the Scholarship "Beca Presidente de la República" granted to me.

I would also like thank to the Defense Advanced Research Project Agency (DARPA) and the Advanced Logistics Program under contract number N00600-96-D-3202 for supporting, in part, this dissertation.

Table of Contents

List of Figures

Abstract

ON OPTIMAL CONSTRAINT DECOMPOSITION, MONITORING, AND MANAGEMENT IN DISTRIBUTED ENVIRONMENTS

Samuel E. Varas G.
George Mason University, 1998
Dissertation Directors: Dr. Larry Kerschberg and Dr. Alex Brodsky

This dissertation addresses three distributed database management problems: integrity constraint management, optimal view materialization, and quasi-view optimal decomposition. First, it considers the problem of decomposing global integrity constraints in a distributed database. Decompositions are performed in order to save communication and other distributed processing costs, since if during a local update the corresponding local constraint is satisfied, no distributed global constraint checking is necessary. This dissertation addresses the problem of deriving the best possible decompositions, both during database design and at update time. It formulates a generic powerful framework for finding optimal decompositions for a range of design and query-time scenarios. It also provides a comprehensive solution for the family of unrestricted linear constraints. Linear (integrity) constraints are widely used in (distributed) applications such as: resource allocation, ticket reservations, financial transactions, and logistics.

The comprehensive optimization-based solution includes: (1) reducing the problem to one of mathematical programming, (2) developing effective algorithms for it, and (3) providing a distributed a protocol to manage updates and constraint decompositions, while guaranteeing desirable properties of consistency, availability, and optimality. Second, the optimal view materialization problem is addressed, where for set of views (queries), one must decide which additional (intermediate) views should be materialized in order to reduce the computational effort of maintaining the views updated. It introduces a generic optimization framework to decide optimally those (intermediate) materialized views. It uses expression-DAG (Directed Acyclic Graph) as a mechanism to represent equivalent view evaluation plans, and shows that the optimal view materialization problem, under certain objective function conditions, is equivalent to finding a constrained shortest path in an expression-DAG. For those cases where the optimal solution is an expression-DAG path, a linear-time algorithm is presented. Third, the concept of quasi-view (a view with explicit update conditions) is extended considerably in this dissertation. The problem of deciding on the optimal quasi-view decomposition is addressed. This problem reduces to the optimal view materialization and constraint decomposition problems. Although the quasi-view decomposition problem is not a separable one, a solution strategy is presented in terms of the view materialization and constraint decomposition problems.

Chapter 1

INTRODUCTION

Decentralized and distributed architectures pose unique problems for database management systems, particularly in the area of constraint management. Distributed architectures can be characterized by: (1) autonomous and distributed data sources connected by loosely-coupled networks, (2) an increasing number of users with complex requirements, and (3) large amounts of multimedia data and information being gathered, cataloged and stored. Distributed database solutions assigned most of the work to local processing, thereby saving on communication costs. However, this decentralization requires additional costs in coordination and control to ensure proper system behavior.

This dissertation investigates and develops optimal solutions for three distributed database management problems: integrity constraint management, optimal view materialization, and optimal quasi-view decomposition. These problems are interrelated in that the solution techniques use results obtained for other problems as shown in Figure 1.1. The arrow in Figure 1.1 denote the "uses" relationship, e.g., the solution

to the "Distributed Constraint Management" problem uses the results and machinery developed to solve the "Optimal Constraint Decomposition" problem. The next section describes a general problem characterization and provides an overview of the contributions for each problem addressed, the complete explanation is found at the respective chapters. Finally, Section 1.2 presents the overall organization for this dissertation.

1.1 Problem Characterization

The problems addressed in this dissertation are interrelated, but each problem can be addressed individually. Figure 1.1 presents the relation among the problems. The research begins by providing a framework for distributed constraint management. Next, it is shown that under certain conditions, one may use the framework to obtain optimal constraint decompositions. The problem of optimizing materialized views is an independent problem. Finally, it is shown that the optimal quasi-view materialization problem involves the novel use of both optimal constraint decomposition and optimal view materialization.

Now, each problem is characterized and an overview of the contributions is provided.

1.1.1 Optimal Integrity Constraint Management

Traditional protocols used to manage global distributed constraints (e.g., two-phase commit) incur enormous overhead and have limitations. In order to reduce those costs and limitations, the idea of local constraint verification has been studied in

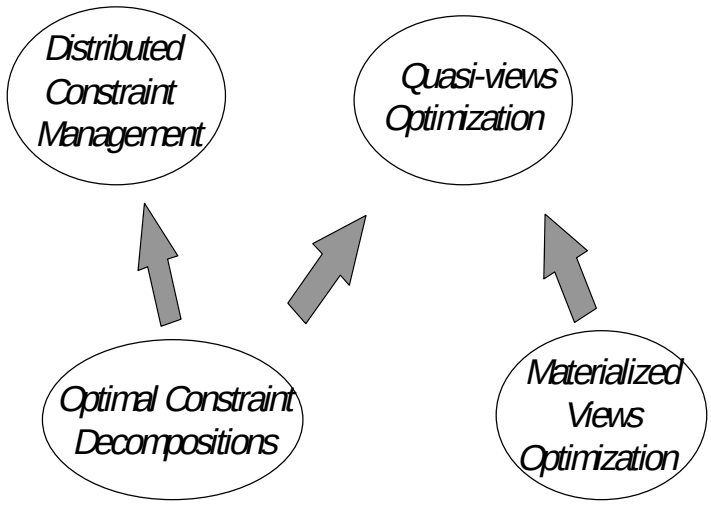

Figure 1.1: Problems Addressed

[BGM92, GM91, GW93, Maz93, Qia89, SV86, GSE+97, Huy97]. This idea decomposes a global constraint into a set of local ones, such that they provide a sufficiency test of the global constraint, i.e., if the local constraints are satisfied, then the global constraint is also satisfied. Furthermore, those sufficiency tests allow for autonomous operation, which is important when some sites are unavailable or the network is partitioned. This problem is important but has not been addressed effectively: (1) most of the previous work has been oriented to finding feasible tests, but they have not addressed the problem of finding the *optimal* one, (2) sufficiency tests require frequent re-definition, because certain database instances may satisfy global constraints, while violating current sufficiency tests, and (3) this dynamic re-definition of sufficiency tests requires a protocol to coordinate and guarantee correct system operation. This moti-

vates the research presented in this thesis.

The first problem addressed in this dissertation corresponds to optimal constraint decomposition. Here a global constraint Ω is decomposed into a set of optimal localized constraints $C_1, \ldots, C_M$, such that they constitute sufficient satisfiability tests for Ω, under various decomposition scenarios. In general, this optimization problem is intractable, since there is no bounded characterization for such decompositions. However, this dissertation proposes a characterization of feasible solutions within a general optimization framework. Moreover, for the case where Ω is represented by a set of general linear constraints, a comprehensive solution is provided which reduces the decomposition problem to a standard mathematical programming problem, and efficient algorithms are provided.

The second problem addressed corresponds to that of distributed constraint management, i.e., the protocol to manage updates and effect concurrent linear integrity constraint decompositions. Even though this problem has been addressed before in [BGM92], this dissertation extends considerably its results by providing a protocol that guarantees properties of consistency, safety, last-resort update refusal, and optimality under communication and site failures. The protocol makes use of previous machinery to decompose and re-decompose a global constraint.

1.1.2 Optimal View Materialization

The evaluation of materialized views (queries) may require considerable computational effort because some materialized views (1) can share some intermediate results (views) with other materialized views, or (2) are complex enough to justify some intermediate pre-computed views. This optimal view materialization problem can be formulated as follows: Given a set of materialized views (queries) $\mathcal{V}$, defined over a set of base-relations or views $\mathcal{R}$, then, one must decide which additional (intermediate) views $\mathcal{V}^*$ should be materialized in order to optimize some criterion (e.g., maintenance costs, response time, etc.), while a set of materialization constraints is satisfied (e.g., maintenance time, available storage, budget, etc.). In general, this problem is NP-hard [Gup97], because, for a given set of possible additional views, it corresponds to selecting a subset of elements, in which the number of subsets is exponential in the number of additional views.

This dissertation solves the optimal view materiazation problem by providing: (1) a mechanism to compactly represent intermediate views (queries) called expression-DAG (Direct Acyclic Graphs) is introduced and extended from [RSS96] [1], (2) an optimization framework is proposed to decide the set $\mathcal{V}^*$ based on a dynamic programming approach, and (3) when a complete path (i.e., expression-path or AND-path) characterizes the solution of this problem, a linear-time algorithm (in terms of the expression-DAG size) is presented.

[1] Expression-DAG and AND-OR graphs are used to represent intermediate views.

Note that this special case is found in important applications such as the case where the evaluation time is the critical variable, and therefore, although more intermediate views are materialized, the complete evaluation should be more efficient.

1.1.3 Optimal Decomposition of Quasi-Views

Many applications must support the monitoring of distributed data for the occurrence of critical events or complex conditions among data items. The concept of quasi-view (i.e., views with explicit re-materialization conditions called *refresh conditions*) has been introduced as a mechanism to refresh views based on the refresh conditions (monitoring conditions). The concept of quasi-view, introduced in [Sel94], is based on the notion of quasi-copies [ABGM90]. However, [Sel94, SK97] did not address the problem of multiple data sources and global events, such as those found in distributed environments.

The problem addressed in this dissertation is how a quasi-view is evaluated efficiently over a distributed environment. In general, this problem is complex because it requires the efficient coordination of the view materialization strategy with the refresh condition strategy. This dissertation proposes a general solution strategy, where the problem is reduced to the optimal view materialization and constraint decomposition problem, and for the special case of refresh conditions represented as disjunctive arithmetic linear constraints, the refresh condition part corresponds to a special case of the "compact split decomposition" problem formulated in Chapter 2.

1.2 Organization

Chapter 2 addresses both the optimal constraint decomposition and distributed constraint management problems. It proposes an optimization framework to find the best decomposition under multiple scenarios. For the general linear constraint case, a mathematical programming reduction and algorithm are provided. In addition a distributed protocol to manage global linear integrity constraints is presented. Chapter 3 addresses the problem of optimal view materialization, which is formulated as an optimization problem, and for some special cases (where a complete materialization path should be selected) a linear-time algorithm is proposed. Chapter 4 addresses the problem of optimal quasi-view decomposition. An optimization framework is proposed to decompose quasi-views, which is based on both the constraint decomposition and the view materialization problems. Finally, Chapter 5 presents the conclusions, a summary of contributions and suggestions for future research.

Chapter 2

OPTIMAL CONSTRAINT MANAGEMENT IN DISTRIBUTED DATABASES

2.1 Introduction

2.1.1 Local Verification of Global Integrity Constraints

Increasingly, enterprise-wide information systems are being built in distributed, heterogeneous environments. The prevalence of the Internet and the World Wide Web [BLCea94] allow designers to incorporate data and information from multiple sources. In those systems centralized control may be difficult, if not impossible, due to the autonomy of the local constituents, which indicates that highly autonomous federated distributed architectures [KGea96] are more appropriate. Often, workflow coordination for distributed systems involves constraint-based agreements, which can be viewed as global integrity constraints [JK97]. These global database integrity constraints are difficult to monitor, update and enforce in distributed environments, so that new,

distributed techniques and protocols are desirable.

To reduce the costs of distributed management of global constraints, the idea of local verification of global constraints was introduced and studied (e.g., [BGM92, GM91, GW93, Maz93, Qia89, SV86, GSE+97]). The idea is to decompose a global constraint into a set of local ones that will serve as a conservative approximation, that is, satisfaction of local constraints by a database instance guarantees satisfaction of the global constraint. Then, when a local site i is being updated, if the update satisfies its local constraint C_i, no global constraint checking is necessary. Thus, most of the work can be delegated to local processing, thereby saving communication and other distributed processing costs. The ability to perform updates autonomously is also very important in presence of site or network failures.

While the above-mentioned works have considered many aspects of local verification (see Related Work section), they have not addressed the problem of finding optimal constraint decompositions and distributed constraint-management protocols that achieve decomposition optimality along with maximal resource utilization. This is precisely the subject of this chapter.

To illustrate the problem of distributed constraint management we now consider an example of a distributed database for an application of logistics support for crisis management.

2.1.2 Crisis Management Scenario Example

Emergency service providers (e.g., fire fighters, medical personnel, military etc.) must be prepared to respond efficiently to crises such as floods, fires and earthquakes. In order to perform Crisis Management, the enterprise must find, coordinate, allocate, deploy and distribute various resources (such as food, clothing, equipment, emergency personnel, transportation) to the victims of a crisis. These resources are typically geographically distributed among many warehouses, suppliers, military units, local fire departments, bus terminals, etc. Each location may maintain a *local database* that stores and monitor information about available resources and their quantities. Our *distributed database* in this example is a loosely-connected collection of local databases, which are related, however, because of a global constraint on resources.

The global constraint in such a distributed database may originate, for example, from a number of pre-defined crisis management scenarios that require that certain amounts of resources be delivered to any potential disaster area within bounded time using available transportation. For example, a Hurricane Relief Mission to Florida may require that the following resources be delivered there in 24 hours: 1) sufficient canned food to feed 30,000 people for 4 days, 2) a supply of tents to support a tent city of 20,000 people, 3) medicines and vaccines to inoculate the tent city residents against cholera, 4) computers and communications equipment to support the coordination, command and control functions of the mission, 5) 10 medical units with medical personal and portable facilities to care for victims, and 6) DoD personnel to staff the mission.

For this scenario, the global integrity constraint would reflect that, for each resource type above, the overall amount of this resource available in all locations reachable in 24 hours (with the available transportation) is greater than or equal to the amount required in the scenario. Note that some resources are composed of other resources, which also needs to be reflected by the global constraint. For example, each of the required 10 medical units (resource 1) is composed of 2 MD's (resource 2), 5 paramedics (resources 3) and must have 2 tents (resource 4), 2000 vaccination packages (resource 5), 500 first aid packages (resource 6), etc. In turn, each vaccination package may be composed of certain quantities of other items and so on.

When a local site of the distributed database is being updated, for example when a certain amount of materiel is taken from a warehouse (not necessarily for crisis support), the update can only be allowed if it satisfies the global integrity constraint, which depends, in general, on the global database instance, not just on the updated local instance. Therefore, verification of the global constraint would require a distributed transaction involving possibly hundreds of loosely connected distributed sites, which might be an extremely expensive and time-consuming operation, especially when protocols such as two phase-commit are used to guarantee the standard properties of transaction atomicity, consistency, isolation and durability (e.g., [JK97]). Moreover, such distributed transaction would often not be possible in the presence of site and network failures, whereas the robustness feature, i.e., the ability to operate in the presence of (partial) failures, is crucial for applications such as Crisis Management. In

short, protocols managing local verification of the global constraint can significantly reduce distributed processing costs and increase the system robustness in the presence of failures.

2.1.3 Contributions

This chapter focuses on the problem of deriving the best possible decompositions, during both database design and update processing. It formulates a generic and powerful framework for finding optimal decompositions for a range of design and update-time scenarios, and provides a comprehensive solution for the case of general linear constraints, which are widely used in distributed applications such as resource allocation, reservations, financial transactions, and logistics. The comprehensive optimization-based solution includes (1) reducing the problem to mathematical programming, (2) developing algorithms for it, and (3) providing a distributed protocol to manage local updates and concurrent distributed constraint decompositions in the presence of communication and site failures, while guaranteeing the desirable properties of consistency, safety, optimality and last-resort update refusal.

More specifically, the contributions of this chapter are as follows. First, we introduce a generic optimization framework to achieve best decompositions by defining (1) the solution space of all *feasible* decompositions (explained below) $(C_1, \ldots, C_M)$ of the global constraint Ω over M distributed sites, and (2) the objective function that can describe a variety of optimization criteria, such as the probability that an update

satisfies its local constraint, the expected number of updates before the first update that violates a local constraint, or the expected overall cost of operations during an update.

The solution space of all *feasible* decompositions is the set of decompositions having the first and possibly other properties from the following list (depending on what is known at the time of a decomposition):

1. *Safety*, i.e., satisfaction of local constraints by a database instance must guarantee satisfaction of the global integrity constraint.

2. *Local Consistency* w.r.t. to a given database instance, i.e., each local instance must satisfy its local constraint (i.e., at the same local site). Clearly, local consistency and safety imply *global consistency.*

3. *Partial-constraint preservation* w.r.t. a given subset θ of sites and local constraints for sites outside θ, i.e., the decompositions cannot change the given local constraints outside θ.

4. *Resource partition* B_θ w.r.t. to a subset θ of sites. This property is based on the notion of *resources* and their *upper bounds* (explained below) associated with each local and the global constraint. Resource partition means that the global constraint resource upper bound is partitioned between the sites in and outside θ, and the cumulative *resources* of sites in and outside θ must be bounded by their corresponding upper bounds. The notion of resource partition is more flexible

than constraint preservation, and allows concurrent constraint (re-) decompositions.

One or more properties 1-4 are required for various decomposition scenarios, depending on what is known at the time of a decomposition. For example, to design a Crisis Management Database schema and local constraints when no actual database instance is known, the property of *safety* is required, while *local consistency* is not applicable. Often, only a partial design is required when local constraints for most sites have already been fixed, in which case the property of *partial constraint preservation* is needed in addition to *safety*. Assume now that the Crisis Management Database is operational, and the current local constraints entail the global constraint (i.e., *safety*) and the current database instance satisfies the local constraints (i.e., *local consistency*). Consider an update at site i, for example when a certain number of blankets is being taken from a warehouse, and the number of remaining blankets, stored at local database site i, has to be updated. If the update satisfies the current local constraint at site i, no processing except for the update itself is necessary, because *safety* guarantees that the global constraint is satisfied. However, if the update violates the local constraint (i.e., *local consistency) no longer holds*, a protocol can try to find a new feasible decomposition of the global constraint that will regain *local consistency* and still be *safe*. A more sophisticated protocol may try to re-decompose constraints only in a (hopefully small) subset θ of (well-connected) sites, which will be done under the assumption that the cumulative resources in and outside θ will stay within their corresponding resource

upper bounds, i.e., a new feasible decomposition will have the property of *resource partition*, in addition to *safety* and *local consistency*.

Second, for the case of general linear arithmetic constraints, we reduce the optimization-based framework to a standard, finitely-specified problem of mathematical programming. This is done by proving existence and actually developing a finite parametric (i.e., in terms of coefficients) characterizations of the properties 1-4 of feasible decompositions together with optimization criteria [1], as follows:

- *Compact Split Decompositions.* Given a global constraint Ω, a parametric characterization of safe decompositions mean formulating a constraint $D(\vec{w})$ whose variables $\vec{w}$ are the parameters (i.e., coefficients) of local constraints, such that $D(\vec{w})$ is true precisely for all safe decompositions. The problem, however, is that in general a constraint C_i at site i may be characterized by an unlimited number of atomic linear constraints; thus the size of a parametric description (using coefficients of those constraints) is unbounded. To overcome this problem, we introduce the notion of *compact split* safe decompositions, for which we prove that: (1) there does exist a parametric description of bounded size and (2) the optimum of any monotonic function [2] among all safe decompositions can always be found in the subspace of compact split safe decompositions.

[1] Not every family of constraints have such finite characterization, but we prove that the linear constraints do.

[2] We claim that any reasonable "decomposition quality" objective function must be monotonic, i.e., intuitively, the more databases instances a decomposition satisfies, the better.

- *Reducing Decompositions to Resource Distributions.* We introduce a *resource-based* characterization of split decompositions to reduce the problem of decomposing constraints to the problem of distributing resources, which significantly simplifies the distributed management of constraints. Specifically, every local constraint C_i for site i in a split decomposition D is uniquely associated with a *resource* vector[3] r_i, and the global constraint is associated with the *global resource* vector, for which we prove that: D is a compact split (safe decomposition) if and only if the cumulative *resource* in all sites is bounded by the *global resource*. Furthermore, given a database instance, every site i is also associated with a *lower resource bound*, for which we prove that: the local database instance at i satisfies its local constraint if and only if its *resource* is bounded from below by its *lower resource bound*. In addition, every site is associated with its *resource upper bound*. The *resource* and its *bounds* for every site constitute a *resource distribution*, which a protocol can maintain instead of explicit local constraints and database instance. The key advantage of a resource distribution is its small size of $O(nc)$ as compared with the size $O(nc*nv)$ of a constraint decomposition, where nc and nv are the number of constraints and variables, respectively, in the global constraint. In fact, nv may be as large as the size of a database, for example when the global constraint reflects that the summation of some quantity, one per relational tuple, is bounded by a constant.

[3]the dimension of this and other resource vectors equals to the number of atomic linear constraints (i.e., linear inequalities over reals) in the global constraint.

- *Concurrent Split Decompositions.* To manage concurrent constraint decompositions, a protocol needs to be able to (re-)decompose constraints autonomously in a (small) subset θ of sites, when the constraints and database instances outside θ are unknown, and, furthermore, may change. [4] The only imposed limitation is the property of *resource partition* B_θ w.r.t. θ, that is, the cumulative resources of sites in and outside θ must be bounded by their current *resource upper bounds* B_θ and its complement, respectively. We show that the decompositions can be done autonomously in θ by proving that, given a database instance for sites in θ, the following are equivalent: (1) there exists a (partial) *permissible* resource distribution for sites in θ w.r.t. B_θ, i.e., such that for each site in θ its *resource* is bounded between its *lower* and *upper bounds*, and that the cumulative *resource upper bound* in θ is exactly B_θ, and (2) there exists a (full) compact split (safe decomposition) of the global constraint that satisfies *resource partition* w.r.t. B_θ and local consistency. Furthermore, we show that optimal constraint decomposition adhering to *resource partition* can also be achieved autonomously in θ. Specifically, we prove that, given any (1) *resource partition* B_θ, (2) local constraints outside θ (and possibly (3) a database instance satisfying the local constraint outside θ), an optimal (partial) safe decomposition in θ adhering to the *resource partition* B_θ and, possibly, to *local consistency* w.r.t. the database instance yields a (full) safe decomposition that is optimal among all safe decom-

[4] Note that constraint preservation property is not adequate for this purpose because it assumes that the constraints outside θ are fixed (and known).

positions that hold the same properties plus the *partial constraint preservation* w.r.t. the local constraints outside θ. Moreover, we show that combining optimal (partial) safe decompositions for sites in and outside θ that satisfy *resource partition* B_θ and, possibly, *local consistency* yields an optimal (full) safe decomposition with the same properties.

- *Parametric Characterization of Objective Function and its localization.* While the parametric characterization of compact split decompositions is applicable to any monotonic objective function, we consider a specific optimization function in more detail: *maximizing the probability of not violating local constraints.* Specifically, we provide an analytical expression of this probability function in terms of parametric characterizations of compact split decompositions (i.e., resources), which we do by using the polyhedron volume function [CH79, Las83, Bea96], under the uniform distribution assumption of database instances, described precisely in Section 2.6. We also express this function in terms of partial resource distribution, so that the optimization could be done within a (small) subset of sites.

Third, we actually develop and partly implement an algorithmic framework to solve the resulting mathematical programming problems, for the case of maximizing the probability of not violating local constraints. For this case, the constructed optimization problems in terms of parametric descriptions have linear constraints and a non-

linear objective function, which is based on a parametric representation of the volume function. For safe decompositions, when each local constraint is in a single variable, the constructed objective function turns out to be concave; this property enables us to use a global search algorithm. We adopt the Frank-Wolfe algorithm [BS79, Kam84] to solve it. For other cases, the objective function is not concave and we use local search techniques in the algorithmic framework, that incorporate the Frank-Wolfe algorithm for search in local neighborhoods. To run experiments and prove the feasibility of the approach we have implemented an optimization engine for safe decompositions with local constraints in single variables. The experiments suggest that the approach is feasible and scalable, but more experimental study will be necessary to fine-tune and extend the algorithms for various specific cases.

Fourth, in order to exemplify the use of constraint decomposition techniques, we develop a distributed (tunable) protocol to manage resource distributions (i.e., local updates and concurrent distributed constraint decompositions) in the presence of site and network failures. We formulate desirable properties to hold for such a protocol, namely, local and global *Consistency*, decomposition *Safety* and *Optimality* and *Last-resort update refusal* (CSOL-properties), and come up with Distributed Protocol Assumptions under which, we prove, the CSOL-properties must hold. The suggested protocol satisfies the assumptions and thus possesses the CSOL-properties, but many other protocols are possible. In particular, our results are readily available to extend, with a significantly more powerful class of constraints and the guaranteed CSOL prop-

erties, a variety of database protocols, including [BGM92] for distributed databases and [SS90] for supporting local transactions in the presence of network partitions. Also, our results on decomposing constraints can easily extend [SK95, SK97] dealing with quasi-views, where global conditions (constraints) for re-materialization can be decomposed among subviews.

2.1.4 Related Work

The body of work on constraints in databases is too large to attempt to survey. The problem of integrity constraint verification has drawn much attention (e.g., [BGM92, GM91, GW93, Maz93, Qia89, QS87, SV86]). This includes local verification of global constraints in distributed databases (e.g., [BGM92, GM91, GW93, Maz93, Qia89, SV86, GSE+97, Huy97]). However, none of the works on local verification, to the best of our knowledge, has solved the problem of optimal selection of local constraints.

More closely associated with our work are the works [BGM92, SS90, MY98] which deal with numerical constraints, and the works [Las, LM92, HJLL] which consider parametric linear constraint queries and their connection to Fourie's elimination method; the work on parametric queries, however, assumes that the number of parameters (i.e., coefficients) is bounded, which is not the case for our *safe* decompositions.

Perhaps most closely related is the work of [BGM92], which was the first to consider verification of linear arithmetic constraints in the context of distributed databases. It considers a single atomic linear constraint at the global level. Intuitively, an atomic

constraint must be such that it could be decomposed between two sites storing individual variables using some constant *boundaries* [5]. For example, the global constraint $A + B \geq 100$ can be decomposed into two local constraints $A \geq a$ and $B \geq b$, where a and b are constants such that $a + b \geq 100$; or the global constraint $A \leq B$ can be decomposed into $A \leq a$ and $b \leq B$, where $a \leq b$. The focus in [BGM92] is on the "demarcation" distributed protocol which is concerned with efficient (in terms of communication and other costs) negotiation between two sites on synchronizing the change in constant boundaries, in case a local update violated its local constraint.

The work [SS90] uses an idea similar to the demarcation of [BGM92] [6], in the context of network partition failures, in order to overcome the problem by trying to perform transactions locally. Similar to [BGM92], a global constraint in the example considered in [SS90] is a single linear inequality of the form $x_1 + \ldots + x_n \leq c$, which is split among n sites (i.e., a single variable per site) by giving each site a quota of c. However, [SS90] focuses on the distributed transaction management and leaves the problem of how to achieve constraint decompositions, as well as the question on what constraint families its techniques are applicable open.

The recent work [MY98] extends the demarcation protocol of [BGM92] by considering a wider class constraints: linear, quadratic and polynomial constraints. A global constraint is a single inequality that is decomposed into local constraints involving one variable per site. Since for this case, the property of *safety* corresponds geometrically to

[5] There is no precise formulation of the allowed atomic constraints in [BGM92].
[6] In fact, [SS90] is earlier.

containment of a multidimensional rectangle in the shape described by the global constraint (inequality), [MY98] suggests the use of geometrical techniques for (dynamic) decompositions. However, geometrical techniques (e.g., from computational geometry) are restricted to low dimension (i.e., small overall number of variables) whereas typically distributed databases involve a large number of variables used in the global constraint (e.g., Crisis Management Scenario).

In contrast to [BGM92, SS90, MY98], our methods on linear arithmetic constraints have none of the above-mentioned restrictions, i.e., we allow atomic linear inequalities of any general form, a global constraint may have any number of atomic constraints, constraints may be partitioned among any number of sites, and each site may have not just one, but any number of variables. Furthermore, ours is the only work suggesting achieving optimal decompositions, and providing a comprehensive solution for it. Moreover, our decompositions can work for different scenarios, i.e., with different assumptions regarding what is known at the time of a decomposition.

The work [Maz93] dealt with first-order (not numerical) constraints, in the context of distributed databases, and suggested certain heuristics to select better decompositions. However, the questions of how, and under what conditions, these heuristics relate to optimization criteria such as maximizing the probability of not violating local constraints and the optimality of decomposition was not considered.

The work [QS87] has also considered a certain class of first-order (not numerical) constraints in the context of equivalent reformulation of a constraint (which is dif-

ferent from our *safety*) in the presence of additional semantic information (not for a distributed environment). They also suggested some heuristics, based on costs of constraint verification and reformulation, but no algorithm or guarantee of optimality in any sense was provided. Finally, [Qia89] applied the techniques of [QS87] for equivalent constraint reformulation in the context of distributed databases.

2.1.5 Organization

The chapter is organized as follows. Following the introduction, Section 2.2 provides a formal framework for selecting optimal decompositions, which is generic for all types of constraints. In Section 2.3 we then concentrate on linear arithmetic constraints. Section 2.4 concentrates on parametric characterizations for the case when each distributed site has just one variable, while Section 2.5 considers the case of unrestricted variable partitions. Section 2.6 discusses the local *uniformity* assumptions on the update space, a specific optimization function and its localization. In Section 2.7 we describe the distributed protocol manage global (integrity) constraints and its properties. Section 2.8 focuses on actual algorithms, implementation and experiments with a number of decomposition examples.

2.2 Decomposition Optimization Framework

In this section we define the central notion of *safe decompositions*, and formulate our problem as one of finding the best feasible safe decomposition of a global constraint. The problem formulation in this section is applicable to all types of constraints.

2.2.1 Safe Decompositions

Definition 1. *A constraint C is a Boolean function from the set of variables $\vec{x}$, to the Boolean set, i.e., C: $Domain(\vec{x}) \rightarrow \{True, False\}$*

We denote the dimension (number of elements) in $\vec{x}$ by $\mid \vec{x} \mid$ = n.

Definition 2. *A variable partition $\mathbb{P}$ of the set of variables $\vec{x}$ is defined as $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$, such that $\vec{y}_1 \cup \vec{y}_2 \cup \ldots \cup \vec{y}_M = \vec{x}$, and $\vec{y}_i \cap \vec{y}_j = \emptyset$ for all i, j ($1 \leq i, j \leq M, i \neq j$).*

Definition 3. *Let Ω be a constraint, and $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ be a partition of variables. We say that $\mathbb{C} = (C_1, \ldots, C_M)$ is a decomposition of Ω, if in every constraint C_i all free variables are from $\vec{y}_i$. Sometimes we will use $\mathbb{C}$ to indicate the conjunction $C_1 \wedge .. \wedge C_M$. We say that a decomposition $\mathbb{C} = (C_1, \ldots, C_M)$ is* safe *if $C_1 \wedge .. \wedge C_M \models \Omega$, where $\models$ denotes logical entailment.*

We also say that $\mathbb{G} = (G_1, \ldots, G_M)$ is a *cover* decomposition of Ω if $(G_1, \ldots, G_M)$ is a decomposition of Ω and $\Omega \models G_1 \vee \ldots \vee G_M$ [7]. The following proposition

[7] we will use $\mathbb{G}$ to indicate the disjunction $G_1 \vee \ldots \vee G_M$.

provides the relationship between safe and cover decompositions.

Proposition 1. *Let Ω be constraint, and $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ be a variable partition. Then, $(C_1, \ldots, C_M)$ is a safe (cover) decomposition of Ω if and only if $(\neg C_1, \ldots, \neg C_M)$ is a cover (safe) decomposition of $\neg\Omega$.*

Proof. Since $(C_1, \ldots, C_M)$ is a safe (cover) decomposition of Ω,

$$C_1 \wedge \ldots \wedge C_M \models \Omega \Leftrightarrow \neg(\neg C_1 \vee \ldots \vee \neg C_M) \models \Omega$$

$$\Leftrightarrow \neg\Omega \models \neg C_1 \vee \ldots \vee \neg C_M$$

This completes the proof. □

In the following we will only concentrate on safe decompositions, but the results can also be applied to cover decompositions using Proposition 1.

Definition 4. *Let $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$ be a database instance. We say that $\vec{x}^0$ satisfies a safe decomposition $\mathbb{C}$ if $\vec{y}_1^0$ satisfies C_1, $\vec{y}_2^0$ satisfies C_2, ..., and $\vec{y}_M^0$ satisfies C_M.*

Example 1. *Consider the following set of linear constraints: $X + Y \leq 6$, $-X + 5Y \leq 15$, $5X + 4Y \leq 15$, and both variables X and Y are non-negative. The partition $\mathbb{P}$ is $(\{X\}, \{Y\})$, and a graphic representation of Ω is given in Figure 2.1.*

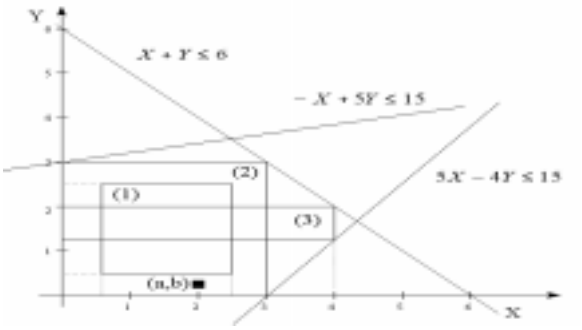

Figure 2.1: Safe Decompositions of Ω

Consider three safe decompositions $\mathbb{C}_1, \mathbb{C}_2, and\, \mathbb{C}_3$*, where:*

$$\begin{aligned}
\mathbb{C}_1 &= (C_{11}, C_{12}) \\
&= (\{0.5 \leq X \leq 2.5\}, \{0.5 \leq Y \leq 2.5\}) \\
\mathbb{C}_2 &= (C_{21}, C_{22}), and \\
&= (\{0.0 \leq X \leq 3.0\}, \{0.0 \leq Y \leq 3.0\}), and \\
\mathbb{C}_3 &= (C_{31}, C_{32}) \\
&= (\{0.0 \leq X \leq 4.0\}, \{1.25 \leq Y \leq 2.0\}).
\end{aligned}$$

$\mathbb{C}_1$ *is a safe decomposition of* Ω *because every point (X,Y) that satisfies* C_{11} *and* C_{12} *will also satisfy* Ω*. Geometrically, this means that the space (1) defined by* $\mathbb{C}_1$ *is contained in the space defined by* Ω*. Similarly,* $\mathbb{C}_2$ *and* $\mathbb{C}_3$ *are safe decompositions of* Ω*. Note that the database instance* (a, b) *satisfies* $\mathbb{C}_2$*, but not* $\mathbb{C}_1$ *and* $\mathbb{C}_3$*.*

Note that rectangle (1) (for $\mathbb{C}_1$) is strictly contained in rectangle (2) (for $\mathbb{C}_2$). Hence, the decomposition $\mathbb{C}_2$ is better than $\mathbb{C}_1$ in the sense that, in $\mathbb{C}_2$ we will have to perform

global updates less frequently than in $\mathbb{C}_1$, i.e., less overhead. This notion is defined formally as follows:

Definition 5. *Given an arbitrary constraint Ω and two decompositions $\mathbb{C}_1 = (C_{11}, \ldots, C_{1M})$ and $\mathbb{C}_2 = (C_{21}, \ldots, C_{2M})$, we say that $\mathbb{C}_2$ subsumes $\mathbb{C}_1$ (or $\mathbb{C}_1$ is subsumed by $\mathbb{C}_2$) if:*

$$\bigwedge_{i=1}^{M} C_{1i} \models \bigwedge_{i=1}^{M} C_{2i}$$

We will denote this by $\mathbb{C}_1 \models \mathbb{C}_2$. We say that $\mathbb{C}_2$ strictly subsumes $\mathbb{C}_1$ if $\mathbb{C}_1 \models \mathbb{C}_2$, but $\mathbb{C}_2 \not\models \mathbb{C}_1$. Furthermore, we say that a safe decomposition $\mathbb{C}$ is minimally-constrained, if there is no safe decomposition $\mathbb{C}'$ that strictly subsumes $\mathbb{C}$. Finally, we say that $\mathbb{C}_1$ is equivalent to $\mathbb{C}_2$, denoted by $\mathbb{C}_1 \equiv \mathbb{C}_2$, if $\mathbb{C}_1 \models \mathbb{C}_2$ and $\mathbb{C}_2 \models \mathbb{C}_1$.

Note that, in example 1, $\mathbb{C}_2$ and $\mathbb{C}_3$ are minimally-constrained safe decompositions, while $\mathbb{C}_1$ is not.

Proposition 2. *Let $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ be a partition of variables, and $\mathbb{C}$ be a conjunction of constraints $(C_1, \ldots, C_M)$, where C_i $(1 \leq i \leq M)$ is over $\vec{y}_i$. Then, $\mathbb{C}$ is satisfiable iff for every i, $1 \leq i \leq M$, C_i is satisfiable.*

Proof. IF-part, if every C_i is satisfiable, its conjunction is satisfiable, i.e., $\mathbb{C}$ is satisfiable. ONLY-IF-part, if $\mathbb{C}$ is satisfiable, and since (1) $(\vec{y}_1, \ldots, \vec{y}_M)$ is a partition of $\vec{x}$, and (2) each C_i, $1 \leq i \leq M$, has its free variables exclusively in $\vec{y}_i$, then, all C_i's are satisfiable. □

Proposition 3. *Let* $\mathbb{C}_1 = (C_{11}, \ldots, C_{1M})$ *and* $\mathbb{C}_2 = (C_{21}, \ldots, C_{2M})$ *be two lists of constraints over* $\vec{y}_1, \ldots, \vec{y}_M$ *respectively, for partition* $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$. *Then:*

1. *If* $\mathbb{C}_1$ *is satisfiable, then* $\mathbb{C}_1 \models \mathbb{C}_2$ *iff for all* i, $1 \leq i \leq M$, $C_{1i} \models C_{2i}$.

2. *If both* $\mathbb{C}_1$ *and* $\mathbb{C}_2$ *are satisfiable* $\mathbb{C}_1 \equiv \mathbb{C}_2$ *iff for all* i, $1 \leq i \leq M$, $C_{1i} \equiv C_{2i}$.

Proof. Part 2 immediately follows from Part 1. In Part 1, the "$\Leftarrow$" direction is obvious, while the "$\Rightarrow$" is due the fact that the variable partition $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ is disjoint as follows:

Assume that $\mathbb{C}_1 \models \mathbb{C}_2$, but, by way of contradiction, for some i, $1 \leq i \leq M$, $C_{1i} \not\models C_{2i}$. Then, there exists $\vec{a}_i$ over $\vec{y}_i$ that satisfies C_{1i}, but not C_{2i}. Since, $\mathbb{C}_1$ is consistent, each $C_{11}, \ldots, C_{1M}$ must be consistent, and therefore there must exist $\vec{b}_1, \ldots, \vec{b}_M$ over $\vec{y}_1, \ldots, \vec{y}_M$, that satisfy $C_{11}, \ldots, C_{1M}$, respectively. Then, $\vec{b}_1, \ldots, \vec{b}_{i-1}, \vec{a}_i, \vec{b}_{i+1}, \ldots, \vec{b}_M$ satisfies $\mathbb{C}_1$, but not $\mathbb{C}_2$, contradicting the fact that $\mathbb{C}_1 \models \mathbb{C}_2$. □

In practical cases, we are only interested in the case when Ω is satisfiable, because otherwise the database must be empty and no update would be allowed. Technically, however, every unsatisfiable (i.e., inconsistent) decomposition will be safe for unsatisfiable Ω. If Ω is satisfiable we have the following:

Proposition 4. *Let* Ω *be a satisfiable constraint. Then every minimally-constrained safe decomposition* $\mathbb{C}$ *of* Ω *is satisfiable.*

Proof. Since Ω is satisfiable, there exists $\vec{a} = (\vec{a}_1, \ldots, \vec{a}_M)$ over $\vec{x}$ that satisfies Ω.

Then, the decomposition $\mathbb{C}_1 = (\vec{y}_1 = \vec{a}_1, \ldots, \vec{y}_M = \vec{a}_M)$ is always a safe decomposition of Ω.

Consider now an arbitrary minimally-constrained safe decomposition $\mathbb{C}$. If, by way of contradiction, $\mathbb{C}$ is not satisfiable, then $\mathbb{C} \models \mathbb{C}_1$ and $\mathbb{C}_1 \not\models \mathbb{C}$, contradicting the minimality of $\mathbb{C}$. □

Clearly, safe or even minimally-constrained safe decompositions are not unique. In our example, both $\mathbb{C}_2$ and $\mathbb{C}_3$ are minimally-constrained, because there is no other safe decomposition that strictly subsumes $\mathbb{C}_2$ or $\mathbb{C}_3$.

Since safe decompositions are not unique, an important question is how to choose a safe decomposition that is optimal according to some meaningful criterion.

In our example, the rectangle with the maximum area may be a good choice. In fact, if update points (X,Y) are uniformly distributed over the given space (defined by Ω), then the larger area (volume in the general case) corresponds to greater probability that an update will satisfy local constraints, and thus no global processing will be necessary. We defer the discussion on optimality criteria to Section 2.6.

2.2.2 Optimization Problem Formulation

We suggest the following general framework for selecting optimal feasible decompositions:

$$\begin{aligned} &maximize\, f(s) \\ &s.t.\ s \in \mathbb{S} \end{aligned} \tag{2.1}$$

where $\mathbb{S}$ is the set of all feasible decompositions, and $f : \mathbb{S} \rightarrow \mathbb{R}$ (real numbers) is the objective function discussed in the next subsection.

Definition 6. *Let Ω be a global constraint, $\mathbb{C} = (C_1, \ldots, C_M)$ be a decomposition of Ω, $\theta = \{k+1, \ldots, M\}$ be a subset of sites $\{1, \ldots, M\}$, and $\vec{x} = (\vec{y}_1, \ldots, \vec{y}_M)$ be a database instance. Then, feasible decompositions are defined as the set of decompositions having one or more of the following properties:*

1. Safety. *Decomposition $\mathbb{C}$ has the safety property if $\mathbb{C}$ is a safe decomposition of Ω.*

2. Local Consistency. *Decomposition $\mathbb{C}$ has the local consistency property if each local instance $\vec{y}_i$ satisfies its local constraint C_i $(1 \leq i \leq M)$. Clearly, local consistency and safety imply global consistency.*

3. θ-Partial Constraint Preservation. *Decomposition $\mathbb{C}$ has the partial constraint preservation w.r.t. θ, if local constraints for sites outside of θ are fixed. This property can be used for scenarios where a (safe) constraint re-decomposition would involve only a (hopefully small) subset of sites θ, leaving the rest unchanged.*

4. θ-Resource Bound Partition [8]. *Decomposition* $\mathbb{C}$ *has the resource bound partition property w.r.t.* θ *and a bound* $\vec{B}_\theta$*, if the overall* resource *of sites in* θ *is bounded by* $\vec{B}_\theta$.

Resource bound partition property is for families of constraints in which resource specification is possible (such as linear constraints). Namely, the global constraint Ω is associated with the global resource bound $\vec{b}$, each local constraint C_i in the decomposition is associated with a resource $\vec{r}_i$, and each subset of sites θ is associated with the cumulative resource $\vec{r}_\theta$.

A partition of the global resource bound between θ and $\bar{\theta}$ (i.e., all sites except θ) is a pair $(\vec{B}_\theta, \vec{B}_{\bar{\theta}})$, such that $\vec{B}_\theta + \vec{B}_{\bar{\theta}} = \vec{b}$, is a partition identified by $\vec{B}_\theta$. We say that a decomposition $\mathbb{C}$ has a *resource bound partition* property w.r.t. a partition $\vec{B}_\theta$, if the cumulative resource $\vec{r}_\theta$ is bounded by $\vec{B}_\theta$, and $\vec{r}_{\bar{\theta}}$ is bounded by $\vec{B}_{\bar{\theta}}$.

Before we discuss how the decomposition problems can be solved effectively, we consider possible candidates for function f.

2.2.3 Optimization Criteria

There are many feasible (minimally-constrained) safe decompositions in $\mathbb{S}$, and we would like to formulate a criterion to select the best among them. This criterion should represent the problem characteristics, and the decomposition goals. Possible

[8] Note that the notion of resource bound partition is more flexible than constraint preservation, and allows one to perform concurrent constraint decompositions.

criteria include:

- Maximize the probability that an update will not violate the existing local constraints (decomposition).
- Minimize overall expected cost of computations during an update.
- Maximize the expected number of updates before the first update that violates local constraints.
- Maximize the expected length of time before an update violates local constraints.

Many other optimization criteria are possible. However, any reasonable criteria should be monotonic, as defined below.

Definition 7. *Let f be a function from the set of safe decompositions of Ω to $\mathbb{R}$. We say that f is monotonic if for every two decompositions $\mathbb{C}_1, \mathbb{C}_2$ of Ω, $\mathbb{C}_1 \models \mathbb{C}_2$ implies $f(\mathbb{C}_1) \leq f(\mathbb{C}_2)$.*

Intuitively, being monotonic for an optimization criterion means that enlarging the space defined by a decomposition can only make it better.

Note, that if f is monotonic, then $f(\mathbb{C}_1) = f(\mathbb{C}_2)$ for any two equivalent decompositions $\mathbb{C}_1$ and $\mathbb{C}_2$.

As we will see in Section 2.5, it is often necessary to consider a subspace of all safe decompositions (without loosing an optimal decomposition).

Definition 8. *Let $\mathbb{S}$ be a set of safe decompositions of Ω. A subset $\mathbb{S}'$ of $\mathbb{S}$ will be called a monotonic cover of $\mathbb{S}$ if for every decomposition $\mathbb{C}$ in $\mathbb{S}$ there exists a decomposition $\mathbb{C}'$ in $\mathbb{S}'$, such that $\mathbb{C}'$ subsumes $\mathbb{C}$ (i.e., $\mathbb{C} \models \mathbb{C}'$).*

The following proposition states that optimal decompositions are not missed when the search space is restricted to a monotonic cover and the optimization criteria are monotonic.

Proposition 5. *Let $\mathbb{S}$ be a (sub) set of all safe decompositions of Ω, $\mathbb{S}'$ be a monotonic cover of $\mathbb{S}$, and f be a monotonic function from $\mathbb{S}$ to $\mathbb{R}$. Then, the following two optimization problems yield the same maximum.*

Problem 1. *max $f(s)$, s.t. $s \in \mathbb{S}$.*

Problem 2. *max $f(s)$, s.t. $s \in \mathbb{S}'$.*

Proof. Suppose the maxima of f in Problem 1 and 2 are achieved by $s = \mathbb{C}$ in $\mathbb{S}$ and $s = \mathbb{C}'$ in $\mathbb{S}'$ respectively. Since $\mathbb{S}' \subseteq \mathbb{S}$, $f(\mathbb{C}') \leq f(\mathbb{C})$. Now, since $\mathbb{S}'$ is a monotonic cover of $\mathbb{S}$, there must exist $\mathbb{C}'' \in \mathbb{S}'$, such that $\mathbb{C} \models \mathbb{C}''$. Therefore, because f is monotonic, $f(\mathbb{C}) \leq f(\mathbb{C}'')$.

Finally, since the maximum of Problem 2 is achieved at $\mathbb{C}'$, $f(\mathbb{C}') \geq f(\mathbb{C}'') \geq f(\mathbb{C})$. Thus, $f(\mathbb{C}') = f(\mathbb{C})$ which completes the proof. □

The functions in the above criteria depend on the update distribution and other assumptions. Specifically we consider two assumptions.

- For design-time decompositions: We do not know the current database state, but we are given a probability distribution of database instances in the space defined by Ω.

- For update-time decompositions: We are given a current database instance, and a conditional distribution function of database instances on Ω.

Now, we present precise methods to characterize the set $\mathbb{S}$, function f, and algorithms to solve effectively the optimization problem (2.1), for the family of linear constraints.

2.3 Linear Arithmetic Constraints

Definition 9. *An atomic linear constraint is an inequality of the form* $a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$, *where* $a_1, a_2, ..., a_n$, *and* b *are real numbers, and* $x_1, x_2, .., x_n$ *are variables ranging over the reals.*

Definition 9 defines a constraint as a symbolic expression. However, an atomic linear constraint $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$ also defines a Boolean function $C : \mathbb{R}^n \rightarrow \{True, False\}$, where for each instantiation of values to variables $(x_1, x_2, \ldots, x_n)$, the expression $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$ is evaluated as true or false.

Definition 10. *A linear system* Ω *is a conjunction of atomic linear constraints.*

A linear system Ω containing n variables and (a conjunction of) m atomic linear constraints, can be written as follows:

$$\begin{array}{lllll} a_{11}x_1 & +a_{12}x_2 & +\ldots & +a_{1n}x_n & \leq b_1 \\ a_{21}x_1 & +a_{22}x_2 & +\ldots & +a_{2n}x_n & \leq b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1}x_1 & +a_{m2}x_2 & +\ldots & +a_{mn}x_n & \leq b_m \end{array} \tag{2.2}$$

This system Ω can also be written in matrix notation as the system $A\,\vec{x} \leq \vec{b}$, where A is the matrix

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix} \tag{2.3}$$

and $\vec{b}$ is the column vector $(b_1\, b_2\, \ldots\, b_m)$, and $\vec{x}$ is the vector $(x_1\, x_2\, \ldots\, x_n)$.

2.3.1 Parametric Optimization Problem

To solve the optimization problem for linear arithmetic constraints we want to rewrite the problem (2.1), i.e.,

$$max\ f(s)$$

$$s.t.\ s \in \mathbb{S}$$

where $\mathbb{S}$ is the set of feasible safe decompositions in the form

$$max\ f(\vec{w})$$

$$s.t.\ \Phi(\vec{w})$$

where $\vec{w}$ is the set of variables describing coefficients (i.e., parameters) of constraints on a decomposition $D(\vec{w})$, and $\Phi(\vec{w})$ is a logical condition in terms of $\vec{w}$ defining the search space

$$\mathbb{S}' = \{D(\vec{w}) \mid \Phi(\vec{w})\}$$

such that $\mathbb{S}'$ is a monotonic cover of $\mathbb{S}$.

By Proposition 5, the two problems are equivalent for any user-given monotonic optimization function, but the latter allows the use of known mathematical programming methods to solve it.

We do it for the case of an individual as well as a general variable partition as

described in Section 2.4 and 2.5, respectively, in which we study the problem of parametric characterization of decompositions.

2.4 Individual Variable Partitions

Individual partition case variables are partitioned in an individual way, i.e., we have a partition $\mathbb{P} = (\{x_1\}, \ldots, \{x_n\})$, where $\{x_1, x_2, \ldots, x_n\}$ is the set of all variables [9].

2.4.1 Parametric Characterization

In this case, safe decompositions can be parametrically described using intervals as follows:

Proposition 6. *Given a bounded set of constraints Ω, and an individual partition $\mathbb{P}$, every decomposition $\mathbb{C} = (C_1, \ldots, C_n)$ of Ω is equivalent to a decomposition $\mathbb{C}'$ of Ω of the form:*

$$(\{u_{11} \leq x_1 \leq u_{21}\}, \ldots, \{u_{1n} \leq x_n \leq u_{2n}\}) \tag{2.4}$$

Proof. Every atomic constraint C_i $(1 \leq i \leq n)$ of $\mathbb{C}$ over $\vec{x}$ can be written as $x_i \leq v_{ij}$ or $z_{ij} \leq x_i$. Thus C_i will be equivalent to $u_{1i} \leq x_i \leq u_{2i}$, where $u_{1i} = max\{z_{i1}, z_{i2}, \ldots, z_{in_i}\}$ and $u_{2i} = min\{v_{i1}, v_{i2}, \ldots, v_{in_i}\}$. □

[9] Note that this implies n =M.

In this section, we will denote the decomposition ($\{u_{11} \leq x_1 \leq u_{21}\}$,..., $\{u_{1n} \leq x_n \leq u_{2n}\}$) by $D(\vec{u})$, where $\vec{u} = (\vec{u}_2, \vec{u}_1)$, $\vec{u}_1 = (u_{11}, u_{12}, \ldots, u_{1n})$ and $\vec{u}_2 = (u_{21}, u_{22}, \ldots, u_{2n})$. We use the notation $\vec{u}_1 \leq \vec{u}_2$ to denote that $u_{1i} \leq u_{2i}$, for all i, $1 \leq i \leq n$.

To create a parametric characterization of the set of all safe decompositions we introduce the notion of characterization matrix as follows.

Definition 11. *Given an $n \times m$ matrix A, the characterization matrix A' of A is defined as $(A^+ A^-)$, where both are $n \times m$ matrices with elements a_{ij}^+ and a_{ij}^- respectively, defined as follows:*

$$a_{ij}^+ = \begin{cases} a_{ij} & \text{if } a_{ij} > 0, \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ij}^- = \begin{cases} a_{ij} & \text{if } a_{ij} < 0, \\ 0 & \text{otherwise} \end{cases}$$

In fact, we have developed the notion of characterization matrix so that the following holds:

Theorem 1. *Let S be the system $A\vec{x} \leq \vec{b}$. Let A' be the characterization matrix of A, and S' be the system $A'\vec{u} \leq \vec{b}$ and $\vec{u}_1 \leq \vec{u}_2$, where $\vec{u} = (\vec{u}_2, \vec{u}_1)$. Then, for every $\vec{u}_1^0, \vec{u}_2^0$ the following are equivalent:*

1. *$\vec{u}^0$ is a solution of S', i.e., $A'\vec{u}^0 \leq \vec{b}$ and $\vec{u}_1^0 \leq \vec{u}_2^0$*

2. *Every $\vec{v}$, $\vec{u}_1^0 \leq \vec{v} \leq \vec{u}_2^0$, is a solution of S, i.e., $A\vec{v} \leq \vec{b}$*

Proof. First, we prove (1) $\Rightarrow$ (2). Assume that $\vec{u}^0 = (\vec{u}_1^0, \vec{u}_2^0)$ is a solution of S'. Let $\vec{v}$ be a vector such that $u_{1i}^0 \leq v_i \leq u_{2i}^0$ for all i, $i \leq i \leq n$. By multiplying $v_i \leq u_{2i}^0$ by a non-negative number α_i^+, we get $\alpha_i^+ v_i \leq \alpha_i^+ u_{2i}^0$ for all i, $1 \leq i \leq n$. Now, choosing α_i^+'s as the elements of j^{th} column of A^+, and making a summation for all possible elements in that row, we get:

$$\sum_{i=1}^{n} a_{ji}^+ v_i \leq \sum_{i=1}^{n} a_{ji}^+ u_{2i}^0 \tag{2.5}$$

and extending for all possible rows in A^+, we get:

$$A^+\vec{v} \leq A^+\vec{u}_2^0 \tag{2.6}$$

Repeating the same operations for $u_{1i}^0 \leq v_i$, i.e., multiplying by a non-positive number α_i^-, we get: $\alpha_i^- u_{1i}^0 \geq \alpha_i^- v_i$ for all i, $1 \leq i \leq n$. Now, choosing α_i^-'s as the elements of j^{th} column of A^-, and making a summation for all possible elements in that row, we get:

$$\sum_{i=1}^{n} a_{ji}^- u_{1i}^0 \geq \sum_{i=1}^{n} a_{ji}^- v_i \tag{2.7}$$

and extending for all possible rows in A^-, we get:

$$A^- \vec{u}_1^0 \geq A^- \vec{v} \tag{2.8}$$

Now, adding (2.6) and (2.8) we get:

$$A^+ \vec{v} + A^- \vec{v} \leq A^+ \vec{u}_2^0 + A^- \vec{u}_1^0 \tag{2.9}$$

We know that $\vec{u}^0 = (\vec{u}_2^0, \vec{u}_1^0)$ is a solution of $A'\vec{u} \leq \vec{b}$, then:

$$\begin{aligned} A^+ \vec{v} + A^- \vec{v} \leq \vec{b}\, or \\ (A^+ + A^-)\vec{v} \leq \vec{b} \end{aligned} \tag{2.10}$$

Therefore, by definition of A^+ and A^-, $A\vec{v} \leq \vec{b}$, i.e., $\vec{v}$ is a solution of S, which completes the proof (1) $\Rightarrow$ (2).

We prove now that (2) $\Rightarrow$ (1),by proving $\neg$ (1) $\Rightarrow \neg$ (2). Assume that $\vec{u}^0$ is not a solution of S', i.e., vector $\vec{u}^0$ does not satisfy $A'\vec{u} \leq \vec{b}$. Then, there exists a column j such that:

$$\sum_{i=1}^{n} a_{ij}^+ u_{2i}^0 + \sum_{i=1}^{n} a_{ij}^- u_{1i}^0 > b_j \tag{2.11}$$

Consider a vector $\vec{v}$ defined as follows:

$$v_i = \begin{cases} u^0_{2i} & \text{if } a^+_{ij} > 0, \\ u^0_{1i} & \text{if } a^-_{ij} < 0, \\ u^0_{2i} & \text{if } a^+_{ij} = a^-_{ij} = 0 \end{cases}$$

Clearly, $\vec{u}^0_1 \leq \vec{v} \leq \vec{u}^0_2$. Then, we can rewrite (2.11) as

$$\sum_{i=1}^{n} a_{ij} v_i > b_j \tag{2.12}$$

Therefore, $\vec{v}$ is not a solution to $A\vec{x} \leq \vec{b}$. This completes the proof. □

2.4.2 Parametric Optimization Problem

We are now ready to characterize parametrically the optimization problem of safe decompositions of Ω.

Proposition 7 (Parametric Feasible Properties). *Let $\Omega = A\vec{x} \leq \vec{b}$ be a global satisfiable constraint, $\mathbb{P} = (x_1, \ldots, x_n)$ be an individual variable partition of $\vec{x}$, A' the characterization matrix of A, $\vec{x}^0 = (x^0_1, \ldots, x^0_n)$ be an instance of $\vec{x}$, θ be a subset $\{k+1, \ldots, M\}$ of sites $\{1, \ldots, M\}$ and $\bar{\theta}$ be its complement, and $D(u^0_{11}, u^0_{21}, \ldots, u^0_{1k}, u^0_{2k})$ be a (partial) $\bar{\theta}$-safe decomposition that satisfies $(x^0_1, \ldots, x^0_k)$. Then, for any decomposition $D(u_{11}, \ldots, u_{1n}, u_{21}, \ldots, u_{2n})$*

1. *$D(u_{11}, \ldots, u_{1n}, u_{21}, \ldots, u_{2n})$ is safe iff $A'\vec{u} \leq \vec{b}$ and $\vec{u}_1 \leq \vec{u}_2$. We denote this condition by $\Phi_{safe}(\vec{u})$.*

2. *$D(u_{11}, \ldots, u_{1n}, u_{21}, \ldots, u_{2n})$ satisfies local consistency w.r.t. $\vec{x}^0$ iff for all i, $1 \leq i \leq n$, $u_{1i} \leq x_i^0 \leq u_{2i}$. We denote this condition by $\Phi_{lc}(\vec{u})$.*

3. *$D(u_{11}, \ldots, u_{1n}, u_{21}, \ldots, u_{2n})$ satisfies partial constraint preservation w. r.t. $\bar{\theta}$-safe decomposition $D(u_{11}^0, u_{21}^0, \ldots, u_{1k}^0, u_{2k}^0)$ iff $u_{11} = u_{11}^0, u_{21} = u_{21}^0, \ldots, u_{1k} = u_{1k}^0, u_{2k} = u_{2k}^0$. We denote this condition by $\Phi_{pcp}(\vec{u})$.*

Proof. 1) follows directly from Theorem 1 and Proposition 6. IF-part: if $A'\vec{u} \leq \vec{b}$ and $\vec{u}_1 \leq \vec{u}_2$, by Theorem 1 (i.e., (1) $\Rightarrow$ (2)), $D(\vec{u})$ is safe. ONLY-IF-part: if $D(u_{11}, \ldots, u_{1n}, u_{21}, \ldots, u_{2n})$ is safe, i.e., $\vec{u}_1 \leq \vec{u}_2$. Since $D(\vec{u})$ is a safe decomposition, the condition (2) of Theorem 1 holds, and thus the condition (1), namely $A'\vec{u} \leq \vec{b}$ and $\vec{u}_1 \leq \vec{u}_2$, which completes this part of the proof.

Now, 2) follows from the definition of local consistency, and 3) This follows directly from partial constraint preservation definition. □

We denote by $\mathbb{S}_{safe}, \mathbb{S}_{lc}, \mathbb{S}_{pcp}$ the set of safe decompositions, the set of decompositions satisfying local consistency w.r.t. $\vec{x}^0$, and decompositions satisfying partial constraint preservation w.r.t. a θ-safe decomposition $D(u_{11}^0, u_{21}^0, \ldots, u_{1k}^0, u_{2k}^0)$ respectively. We will use Pr to denote a subset of the set of properties $\{safety, lc, pcp\}$. Finally, set $\mathbb{S}_{Pr}$ will denote the set of all decompositions that satisfy the properties Pr, i.e., $\mathbb{S}_{Pr} = \cap_{p \in Pr} \mathbb{S}_p$, and $\Phi_{Pr}(\vec{u})$ will be the conjunction of the corresponding conditions, i.e.,

$\Phi_{Pr}(\vec{u}) = \wedge_{p \in Pr} \Phi_p(\vec{u})$. We can present the optimization problem in terms of resource characterization.

Theorem 2 (Parametric Optimization Problem). *Let $\Omega = A\vec{x} \leq \vec{b}$ be a satisfiable global constraint, f be a monotonic function from the set of all safe decompositions to $\mathbb{R}$, $\mathbb{P} = (\vec{x}_1, \ldots, \vec{x}_n)$ a individual variable partition of $\vec{x}$, $\vec{x}^0 = (\vec{x}^0_1, \ldots, \vec{x}^0_n)$ be an instance of $\vec{x}$, $D(u^0_{11}, u^0_{21}, \ldots, u^0_{1k}, u^0_{2k})$, $1 \leq k \leq M$, a partial safe decomposition that satisfies $(x^0_1, \ldots, x^0_k)$, and Pr be the subset of properties $\{safety, lc, pcp\}$ that must contain safety. Then, solving the optimization problem*

$$max\ f(s)$$

$$s.t.\ s \in \mathbb{S}_{Pr}$$

is equivalent to solving the parametric problem

$$max\ f(D(\vec{u}))$$

$$s.t.\ \Phi_{Pr}(\vec{u})$$

Proof. First, by Proposition 6, every decomposition $\mathbb{C}$ in $\mathbb{S}_{Pr}$ has an equivalent decomposition $\mathbb{C}'$ of the form $D(\vec{u}^0)$, where $\vec{u}^0_1 \leq \vec{u}^0_2$. Then, by Proposition 7, $\mathbb{S}_{Pr}$ and $\Phi_{Pr}(\vec{u})$ represent equivalent search spaces. Therefore, both problems yield the same maximum. This completes the proof. □

2.5 General Variable Partitions

This section addresses the general partition case. Let $\mathbb{P} = (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_M)$ be a partition of $\vec{x}$, where $\vec{y}_i$ is the subset of variables at site i, ($\mid \vec{y}_i \mid = n_i$), and $\vec{x}$ is the vector of all variables in our problem.

The problem in the general case is that, for a safe decomposition, a constraint C_i at site i may be characterized by an unbounded set of atomic linear constraints; thus the size of a parametric description (using coefficients of those constraints) is unbounded. To overcome this problem, we reduce the search space to the set of what we call *compact split* decompositions, for which we prove that: (1) there does exist a parametric description of bounded size and (2) the optimum of the objective function among all safe decompositions *can always be found* in the subspace of split decompositions.

2.5.1 Split Decompositions

Definition 12. *Let $\Omega = A\vec{x} \leq \vec{b}$ be a constraint on $\vec{x}$, and $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ be a variable partition of $\vec{x}$. A* split *of Ω, denoted by $D(\vec{r}_1, \ldots, \vec{r}_M)$, is a tuple $(A_1\vec{y}_1 \leq \vec{r}_1, \ldots, A_M\vec{y}_M \leq \vec{r}_M)$ of constraints, where A_i, $1 \leq i \leq M$, are those columns of A associated with $\vec{y}_i$. We say that a split $D(\vec{r}_1, \ldots, \vec{r}_M)$ is safe (respectively minimally-constrained) if it is a safe (respectively minimally-constrained) decomposition of Ω. For a subset θ of sites, say $\{k+1, \ldots, M\}$, a (partial) θ-split of Ω, denoted by $D(\vec{r}_{k+1}, \ldots, \vec{r}_M)$, is a tuple $(A_{k+1}\vec{y}_{k+1} \leq \vec{r}_{k+1}, \ldots, A_M\vec{y}_M \leq \vec{r}_M)$ of constraints.*

Recall that, by Proposition 4, $D(\vec{r}_1, \ldots, \vec{r}_M)$ is satisfiable if and only if for all i, $1 \leq i \leq M$, $A_i\vec{y}_i \leq \vec{r}_i$ is satisfiable.

For our classification we introduce the notion of *tight form* for a system $A\vec{x} \leq \vec{b}$, which states, intuitively, that the values of $\vec{b}$ are tight. This is formalized as follows.

Definition 13. *We say that a constraint $A\vec{x} \leq \vec{b}$ is* tight*, if there does not exist $\vec{b}'$, such that $\vec{b}' \leq \vec{b}$, $\vec{b}' \neq \vec{b}$, and $A\vec{x} \leq \vec{b}$ is equivalent to $A\vec{x} \leq \vec{b}'$. We say that a split $D(\vec{r}_1, \ldots, \vec{r}_M)$ is tight if every satisfiable constraint $A_i\vec{y}_i \leq \vec{r}_i$ in it ($1 \leq i \leq M$) is tight.*

Claim 1. *For any satisfiable system $A\vec{x} \leq \vec{b}$ (respectively safe split) there exists an equivalent system (respectively split) that is tight. Furthermore, every tight constraint $A\vec{x} \leq \vec{b}$ is satisfiable.*

Definition 14. *We say that a split $D(\vec{r}_1, \ldots, \vec{r}_M)$ of Ω is* compact *if*

$$\sum_{i=1}^{M} \vec{r}_i \leq \vec{b}$$

Lemma 1 (Split Properties). *Let $\Omega = A\vec{x} \leq \vec{b}$. Then:*

1. *Every compact split is safe.*
2. *If $D(\vec{r}_1, \ldots, \vec{r}_M)$ is a tight split, it is compact iff it is safe.*
3. *For every safe decomposition $\mathbb{C}$ of Ω, there exists a minimally-constrained safe split $D(\vec{r}_1, \ldots, \vec{r}_M)$ of Ω, that subsumes $\mathbb{C}$, i.e., $\mathbb{C} \models D(\vec{r}_1, \ldots, \vec{r}_M)$.*

4. Every minimally-constrained safe decomposition of Ω is equivalent to (1) a minimally - constrained safe split of Ω and to (2) a compact split of Ω.

Proof. (1) Let $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$ be an arbitrary point that satisfies $D(\vec{r}_1, \ldots, \vec{r}_M)$, i.e., $A_i\vec{y}_i^0 \leq r_i$, for all i, $1 \leq i \leq M$. Then,

$$\sum_{i=1}^{M} A_i\vec{y}_i^0 \leq \sum_{i=1}^{M} \vec{r}_i \leq \vec{b}$$

Finally, since $\sum_{i=1}^{M} A_i\vec{y}_i^0 = A\vec{x}^0$, we get that $A\vec{x}^0 \leq \vec{b}$, i.e., $\vec{x}^0$ satisfy Ω.

(2) The IF-direction is subsumed by (1). For the ONLY-IF-direction, suppose $\sum_{i=1}^{M} \vec{r}_i \leq \vec{b}$ is not true. Then, there exists a row j, such that $\sum_{i=1}^{M} r_{ij} > b_j$. Now, we select $\vec{x}^0 = \{\vec{y}_1^0, \vec{y}_2^0, \ldots, \vec{y}_M^0\}$, as follows. For each $i, 1 \leq i \leq M$, we take $\vec{y}_i^0$ to be the point that achieves $Max\ \vec{A}_{ij}\vec{y}_i$, subject to $A\vec{y}_i \leq \vec{r}_i$, with $\vec{A}_{ij}$ is the j^{th} row of matrix A. Because the system $A\vec{y}_i \leq \vec{r}_i$ is in reduced form, $\vec{A}_{ij}\vec{y}_i^0 = r_{ij}$. Thus, $\vec{x}^0$ satisfies $D(\vec{r}_1, \ldots, \vec{r}_M)$ but not $\Omega = A\vec{x} \leq \vec{b}$, because

$$\vec{A}_{ij}\vec{x}^0 = \sum_{i=1}^{M} \vec{A}_{ij}\vec{y}_i^0 = \sum_{i=1}^{M} r_{ij} > b_j$$

This complete the ONLY-IF direction of (2).

(3) We first prove that:

Claim 2. *For every safe decomposition* $\mathbb{C}$ *of* Ω*, there exists a safe split* $D(\vec{r}_1, \ldots, \vec{r}_M)$ *that subsumes* $\mathbb{C}$*.*

Let $\mathbb{C} = (C_1, \ldots, C_M)$ be a safe decomposition of Ω. We want to construct $D(\vec{r}_1, \ldots, \vec{r}_M)$. Consider first the case when $\mathbb{C}$ is satisfiable. For every $i, 1 \leq i \leq m$, where m is the number of rows in matrix A, we construct $\vec{r}_i = (r_{i1}, \ldots, r_{im})$ as follows: for every $j, 1 \leq j \leq m$, we take r_{ij} to be the minimal value such that

$$C_i \models \vec{A}_{ij}\vec{y}_i \leq r_{ij}$$

Then, $C_i \models A_i\vec{y}_i \leq \vec{r}_i$, and therefore $\mathbb{C} \models D(\vec{r}_1, \ldots, \vec{r}_M)$. Therefore, by (1) of this Theorem, to prove that $D(\vec{r}_1, \ldots, \vec{r}_M)$ is a safe decomposition, it is sufficient to prove that $D(\vec{r}_1, \ldots, \vec{r}_M)$ is a safe split, i.e., $\sum_{i=1}^{M} \vec{r}_i \leq \vec{b}$. We prove that for every row $j, 1 \leq j \leq m$, i.e., $\sum_{i=1}^{M} r_{ij} \leq b_j$.

Selecting $\vec{x}^{(j)} = (\vec{y}_1^{(j)}, \ldots, \vec{y}_M^{(j)})$ is done as follows: $\vec{y}_i^{(j)}$ is the value of $\vec{y}_i$ that maximizes the function $\vec{A}_{ij}\vec{y}_i$, subject to $A_i\vec{y}_i \leq \vec{r}_i$. Because $D(\vec{r}_1, \ldots, \vec{r}_M)$ is constructed so that it is in a reduced form, $\vec{A}_{ij}\vec{y}_i^{(j)} = r_{ij}$. Then, for every $j, 1 \leq j \leq m$

$$\sum_{i=1}^{M} d_{ij} = \sum_{i=1}^{M} \vec{A}_{ij}\vec{y}_i^{(j)} = \vec{A}_j\vec{x}^{(j)} \leq b_j$$

This completes the proof of Claim 2 for the case when $\mathbb{C}$ is satisfiable.

If $\mathbb{C}$ is not satisfiable, consider $\mathbb{C}' = (\vec{x} = \vec{x}^0)$, where $\vec{x}^0$ is an arbitrary point that

satisfies Ω. Clearly,$\mathbb{C}'$ is satisfiable. By what has been proved, we can construct a safe decomposition $D(\vec{r}_1, \ldots, \vec{r}_M)$ of Ω such that:

$$\mathbb{C}' \models D(\vec{r}_1, \ldots, \vec{r}_M)$$

and, since $\mathbb{C} \models \mathbb{C}'$, it follows that $\mathbb{C} \models D(\vec{r}_1, \ldots, \vec{r}_M)$. This complete the proof of Claim 2.

Claim 3. *Every minimally-constrained safe decomposition $\mathbb{C}'$ is equivalent to a minimally - constrained safe split $D(\vec{r}_1, \ldots, \vec{r}_M)$.*

Indeed, by Claim 2, there exists a safe split $D(\vec{r}_1, \ldots, \vec{r}_M)$ such that $\mathbb{C}' \models D(\vec{r}_1, \ldots, \vec{r}_M)$. Then, because $\mathbb{C}'$ is minimally-constrained, $D(\vec{r}_1, \ldots, \vec{r}_M)$ must be equivalent to $\mathbb{C}'$. Therefore, $D(\vec{r}_1, \ldots, \vec{r}_M)$ is a minimally-constrained safe split.

Now, to prove (3) given a safe decomposition $\mathbb{C}$, we construct a minimally-constrained safe decomposition $\mathbb{C}'$, such that $\mathbb{C} \models \mathbb{C}'$. Then, by Claim 3, there exists a minimally-constrained safe split $D(\vec{r}_1, \ldots, \vec{r}_M)$ that is equivalent to $\mathbb{C}'$. Therefore, $D(\vec{r}_1, \ldots, \vec{r}_M)$ is a minimally-constrained safe split that subsumes $\mathbb{C}$. This completes the proof of (3).

(4) It is essentially Claim 3 that has been proved in (3). □

The next theorem shows that the maximum of a monotonic function in the space of safe decompositions can always be found in the subspace of compact splits.

Theorem 3. *Let $\Omega = A\vec{x} \leq \vec{b}$ be a satisfiable global constraint, f be a monotonic function from the set of all safe decompositions of Ω to $\mathbb{R}$, $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ be a variable partition of $\vec{x}$, and $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$ be an instance of $\vec{x}$. Let $\mathbb{S}$ and $\mathbb{SS}$ be the sets of all safe decompositions and all compact splits of Ω, respectively, and let $\mathbb{S}_{\vec{x}^0}$ and $\mathbb{SS}_{\vec{x}^0}$ be the sets $\mathbb{S}$ and $\mathbb{SS}$ restricted to decompositions that satisfy $\vec{x}^0$. Then,*

1. $max\ f(s)\ s.t.\ s \in \mathbb{S} = max\ f(s)\ s.t.\ s \in \mathbb{SS}$

2. $max\ f(s)\ s.t.\ s \in \mathbb{S}_{\vec{x}^0} = max\ f(s)\ s.t.\ s \in \mathbb{SS}_{\vec{x}^0}$

Proof. (1) The proof follows from the fact that $\mathbb{SS} \subseteq \mathbb{S}$. Thus, by Lemma 1, $\mathbb{SS}$ is a *monotonic cover* of $\mathbb{S}$. Then, using Proposition 5, both problems yield the same maximum.

(2) The proof follows directly from 1), because 2) is a particular case of 1). This completes the proof. □

Following Theorem 3, from now on we only consider compact splits. Vectors $(\vec{r}_1, \ldots, \vec{r}_M)$ can be considered as resources assigned to sites, because they represent how much of vector $\vec{b}$ is distributed to each site. The following subsection presents a parametric resource characterization of splits and a parametric formulation of the optimization problem in terms of resources.

2.5.2 Resource Characterization

This subsection characterizes (compact) splits in terms of resources. This resource-based characterization supports a concurrent constraint (re-) decompositions.

In particular, we formulate the properties of *compactness*, *local consistency (lc)*, *partial constraint preservation (pcp)*, and *resource bound partition (rp)* for splits in terms of resources. Then, the optimization problem is formulated in terms of such a characterization. First, we introduce the concept of *resources* of (compact) splits.

Definition 15 (Resource Parameters). *Let* $\Omega = A\vec{x} \leq \vec{b}$ *be a satisfiable global constraint,* $D(\vec{r}_1, \ldots, \vec{r}_M)$ *be a tight compact split of* Ω, $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$ *be an instance of* $\vec{x}$, *and* θ *be a subset* $\{k+1, \ldots, M\}$ *of sites* $\{1, \ldots, M\}$. *Then, we say that:*

1. $\vec{b}$ *is the* global upper bound *of resources in* Ω.

2. $\vec{r}_i$ *is the* resource *assigned to site* i, $1 \leq i \leq M$.

3. $\vec{r} = \sum_{i=1}^{M} \vec{r}_i$ *is the* global resource.

4. $\vec{\delta} = \vec{b} - \vec{r}$ *is the* global passive slack *of* Ω *w.r.t.* $D(\vec{r}_1, \ldots, \vec{r}_M)$.

5. $(\vec{\delta}_1, \ldots, \vec{\delta}_M)$ *such that* $\vec{\delta}_i \geq 0$, $1 \leq i \leq M$ *and* $\sum_{i=1}^{M} \vec{\delta}_i = \vec{\delta}$, *is a partition of* $\vec{\delta}$. *Each* $\vec{\delta}_i$, $1 \leq i \leq M$, *is called the passive slack at site* i.

6. $\vec{ur}_i = \vec{r}_i + \vec{\delta}_i$ *is the* upper resource bound *at site* i, $(1 \leq i \leq M)$.

7. *Given an instance* $\vec{y}_i^0$ *at site* i, $\vec{lr}_i = A_i\vec{y}_i^0$, $1 \leq i \leq M$, *is the* lower resource bound *at site* i *w.r.t.* $\vec{y}_i^0$.

8. *Given an instance* $\vec{y}_i^0$ *at site* i, $\vec{\Delta}_i = \vec{r}_i - \vec{lr}_i$, $1 \leq i \leq M$, *is the* active slack *at site* i *w.r.t.* $\vec{y}_i^0$.

Finally, we define all the parameters for θ *by* $\vec{r}_\theta = \sum_{i\in\theta}\vec{r}_i$, $\vec{ur}_\theta = \sum_{i\in\theta}\vec{ur}_i$, $\vec{lr}_\theta = \sum_{i\in\theta}\vec{lr}_i$, $\vec{\delta}_\theta = \sum_{i\in\theta}\vec{\delta}_i$, *and* $\vec{\Delta}_\theta = \sum_{i\in\theta}\vec{\Delta}_i$.

The above resource parameters are shown in Figure 2.2. In this figure, each resource $\vec{r}_i$ is bounded between its lower and upper bound ($\vec{lr}_i$ and $\vec{ur}_i$), the difference between upper bound ($\vec{ur}_i$) and the resource is the passive slack $\vec{\delta}_i$, and the difference between the resource ($\vec{r}_i$) and its lower bound ($\vec{lr}_i$) is the active slack ($\vec{\Delta}_i$).

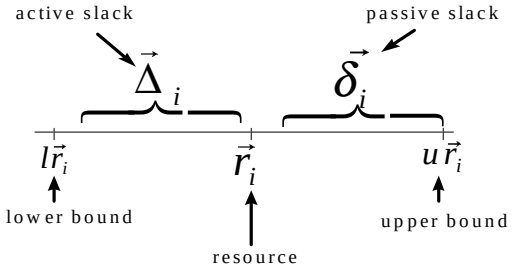

Figure 2.2: Resource Representation at Site i

The following proposition characterizes the split properties of *compactness*, *local consistency (lc)*, *partial constraint preservation (pcp)*, and *resource bound partition (rp)* in terms of the resource parameters.

Proposition 8 (Parametric Feasible Properties). *Let* $\Omega = A\vec{x} \leq \vec{b}$ *be a global satisfiable constraint,* $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ *be a variable partition of* $\vec{x}$, $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$

be an instance of $\vec{x}$, θ *be a subset of sites, say* $\theta = \{k+1, \ldots, M\}$, $\bar{\theta}$ *be the set* $\{1, \ldots, k\}$, *and* $D(\vec{r}_1^0, \ldots, \vec{r}_k^0)$ *be a (partial)* $\bar{\theta}$*-split that satisfies* $(\vec{y}_1^0, \ldots, \vec{y}_k^0)$. *Then, for any split* $D(\vec{r}_1, \ldots, \vec{r}_M)$

1. $D(\vec{r}_1, \ldots, \vec{r}_M)$ *is compact iff the global resource* $\vec{r}$ *is bounded by the global upper bound* $\vec{b}$, *i.e.,* $\vec{r} \leq \vec{b}$. *We denote this condition by* $\Phi_{compact}(\vec{r}_1, \ldots, \vec{r}_M)$.

2. $D(\vec{r}_1, \ldots, \vec{r}_M)$ *satisfies local consistency w.r.t.* $\vec{x}^0$ *iff the resource* $\vec{r}_i$ *assigned to site* i *is bounded from below by its lower bound* $\vec{lr}_i$, *i.e., for every site* i, $1 \leq i \leq M$, $\vec{lr}_i \leq \vec{r}_i$. *We denote this condition by* $\Phi_{lc}(\vec{r}_1, \ldots, \vec{r}_M)$.

3. $D(\vec{r}_1, \ldots, \vec{r}_M)$ *satisfies partial constraint preservation w.r.t.* $\bar{\theta}$*-split* $D(\vec{r}_1^0, \ldots, \vec{r}_k^0)$ *iff the resources at each site outside* θ *are fixed, i.e.,* $r_1 = r_1^0, \ldots, r_k = r_k^0$. *We denote this condition by* $\Phi_{pcp}(\vec{r}_1, \ldots, \vec{r}_M)$.

We will also denote by $\Phi_{rp,\vec{B}_\theta}(\vec{r}_1, \ldots, \vec{r}_M)$ *the condition stating that* $D(\vec{r}_1, \ldots, \vec{r}_M)$ *satisfies resource partition w.r.t. a resource bound partition* $\vec{B}_\theta$.

Proof. 1) follows directly from Lemma 1 part 1, 2) follows from the definition of local consistency, i.e., for a given instance $\vec{y}_i^0$, it satisfies its local constraint iff $A_i \vec{y}_i^0 \leq \vec{r}_i$, and 3) follows directly from partial constraint preservation definition. □

In the following, we denote by $\mathbb{SS}_{compact}$, $\mathbb{SS}_{lc}$, $\mathbb{SS}_{pcp}$ and $\mathbb{SS}_{rp}$ the set of splits satisfying compactness, local consistency w.r.t. $\vec{x}^0$, partial constraint preservation w.r.t. a θ-split $D(\vec{r}_1, \ldots, \vec{r}_k)$, and resource bound partition w.r.t. θ and bound B_θ respectively. We will use Pr to denote a subset of the set of properties $\{compact, lc, pcp, rp\}$.

Finally, set $\mathbb{S}_{Pr}$ will denote the set of all splits that satisfy the properties Pr, i.e., $\mathbb{SS}_{Pr} = \cap_{p \in Pr} \mathbb{SS}_p$, and $\Phi_{Pr}(\vec{r}_1, \ldots \vec{r}_M)$ will be the conjunction of the corresponding conditions, i.e., $\Phi_{Pr}(\vec{r}_1, \ldots \vec{r}_M) = \wedge_{p \in Pr} \Phi_p(\vec{r}_1, \ldots \vec{r}_M)$. We can present the optimization problem in terms of resource characterization.

Theorem 4 (Resource Optimization). *Let $\Omega = A\vec{x} \leq \vec{b}$ be a satisfiable global constraint, f be a monotonic function from the set of all safe decompositions to $\mathbb{R}$, $\mathbb{P} = (\vec{y}_1, \ldots, \vec{y}_M)$ a variable partition of $\vec{x}$, $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$ be an instance of $\vec{x}$, and Pr be the subset of properties $\{compactness, lc, pcp, rp\}$ that must contain compactness or resource bound partition. Then, solving the optimization problem*

$$max\ f(s)$$

$$s.t.\ s \in \mathbb{SS}_{Pr}$$

is equivalent to solving the parametric problem [10]

$$max\ f(D(\vec{r}_1, \ldots, \vec{r}_M))$$

$$s.t.\ \Phi_{Pr}(\vec{r}_1, \ldots, \vec{r}_M)$$

Proof. The proof follows from Lemma 1, Proposition 8, and the fact that resource bound partition is a stronger property than compactness as follows:

(1) The *compactness* part. By Proposition 8, the set $\mathbb{SS}_{compactness}$ is characterized

[10] we will sometimes write $f(D(\vec{r}_1, \ldots, \vec{r}_M))$ as $f(\vec{r}_1, \ldots, \vec{r}_M)$.

by condition $\Phi_{compactness}(\vec{r}_1, \ldots, \vec{r}_M)$. Then, both problems yield the same maximum.

The other Pr cases follow directly from Proposition 8. This completes the proof. □

2.5.3 Concurrent Split Decompositions

The resource characterization of the previous subsection assumes that information from all sites is used. However, in order to support distributed and autonomous protocols we would like to make constraint decompositions and re-decompositions involving only a (small) subset of sites, say $\theta = \{(k+1), \ldots, M\}$ of sites $\{1, \ldots, M\}$. To do that a formulation of the decomposition problem can only involve or affect information that is stored in sites θ.

Definition 16. *Let $\Omega = A\vec{x} \leq \vec{b}$ be a global constraint, $D(\vec{r}_1, \ldots, \vec{r}_M)$ be a compact split of Ω, $(\vec{\delta}_1, \ldots, \vec{\delta}_M)$ be a partition of the global passive slack $\vec{\delta}$, $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$ be a database instance, θ is a subset, say $\{k+1, \ldots, M\}$, of sites. A* resource distribution *is a tuple (of triples) $((\vec{lr}_1, \vec{r}_1, \vec{ur}_1), \ldots, (\vec{lr}_M, \vec{r}_M, \vec{ur}_M))$ [11]; a θ-resource distribution is $((\vec{lr}_{k+1}, \vec{r}_{k+1}, \vec{ur}_{k+1}), \ldots, (\vec{lr}_M, \vec{r}_M, \vec{ur}_M))$. We say that the resource distribution is* permissible *if*

$$\sum_{i=1}^{M} \vec{ur}_i = \vec{b} \text{ and } \vec{lr}_i \leq \vec{r}_i \leq \vec{ur}_i, \text{for every } 1 \leq i \leq M$$

Given a resource bound partition $\vec{B}_\theta$, we say that the θ-resource distribution $((\vec{lr}_{k+1},$

[11] Note $(\vec{lr}_i, \vec{r}_i, \vec{ur}_i)$'s are defined in Definition 15.

$\vec{r}_{k+1}, \vec{ur}_{k+1}), \ldots, (\vec{lr}_M, \vec{r}_M, \vec{ur}_M))$ *is permissible w.r.t.* $\vec{B}_\theta$ *if*

$$\sum_{i \in \theta} \vec{ur}_i = \vec{B}_\theta \text{ and } \vec{lr}_i \leq \vec{r}_i \leq \vec{ur}_i, \text{for every } i \in \theta$$

Note that if θ is the set of all sites, the resource distribution permissibility is equivalent to θ resource distribution permissibility.

The following proposition motivates the notion of permissible distribution and provides a local criterion for a subset θ of sites to decide whether a feasible resource distribution exists, after database instances have been updated (i.e., lower bounds).

Proposition 9 (θ-Resource Distribution Feasibility). *Let θ be a subset, say $\{k+1, \ldots, M\}$, of sites $\{1, \ldots, M\}$, and $\bar{\theta}$ be its complement, $\vec{B}_\theta$ be a resource bound partition. Then,*

1. *Given a database instance $(\vec{y}^0_{k+1}, \ldots, \vec{y}^0_M)$ at sites θ (and thus lower bounds $(\vec{lr}_{k+1}, \ldots, \vec{lr}_M)$), the following are equivalent:*

   (a) *There exists a compact split of Ω satisfying resource partition $\vec{B}_\theta$ and local consistency w.r.t. $(\vec{y}^0_{k+1}, \ldots, \vec{y}^0_M)$.*

   (b) *There exists a θ-permissible resource distribution w.r.t. $\vec{B}_\theta$ (with the above lower bounds).*

   (c) $\vec{lr}_\theta = \sum_{i \in \theta} \vec{lr}_i \leq \vec{B}_\theta$

2. *The combination of θ-permissible resource distribution w.r.t. $\vec{B}_\theta$ and $\bar{\theta}$-permissible*

resource distribution w.r.t. $\vec{B}_{\bar{\theta}}$ constitutes a permissible resource distribution.

Proof. (1) To prove this part, we first prove that (c) implies (a), (a) implies (b), and (b) implies (c).

(c) implies (a). Using the definition of $\vec{lr}_\theta$,

$$\vec{lr}_\theta \leq \vec{B}_\theta \Leftrightarrow \sum_{i\in\theta} \vec{lr}_i \leq \vec{B}_\theta$$
$$\Leftrightarrow \sum_{i\in\theta} A_i\vec{y}_i^0 \leq \vec{B}_\theta$$

Then, selecting $\vec{r}_i = A_i\vec{y}_i^0$, for all i, $k+1 \leq i \leq M-1$, and $\vec{r}_M = \vec{B}_\theta - \sum_{i=k+1}^{M-1} \vec{r}_i$, we build a θ-compact split of Ω satisfying resource partition $\vec{B}_\theta$ and local consistency w.r.t. $(\vec{y}_{k+1}^0, \ldots, \vec{y}_M^0)$. Now, selecting $\vec{r}_i' = (\vec{b} - \vec{B}_\theta)/k$, for $1 \leq i \leq k$, $D(\vec{r}_1', \ldots, \vec{r}_k', \vec{r}_{k+1}, \ldots, \vec{r}_M)$ is a compact split of Ω. This completes this part of the proof.

(a) implies (b). Let $D(\vec{r}_1', \ldots, \vec{r}_M')$ be a compact split of Ω satisfying resource partition $\vec{B}_\theta$ and local consistency w.r.t. $(\vec{y}_{k+1}^0, \ldots, \vec{y}_M^0)$. Then, $\vec{lr}_i \leq \vec{r}_i'$ for all i, $k+1 \leq i \leq M$, and $\sum_{i\in\theta} \vec{r}_i' \leq \vec{B}_\theta$. Then, selecting $\vec{ur}_i = \vec{r}_i'$, for all i, $k+1 \leq i \leq M-1$, and $\vec{ur}_M = \vec{B}_\theta - \sum_{i=k+1}^{M-1} \vec{r}_i'$. Therefore, $\vec{lr}_i \leq \vec{r}_i' \leq \vec{ur}_i$, $k+1 \leq i \leq M$, i.e., $((\vec{lr}_{k+1}, \vec{r}_{k+1}', \vec{ur}_{k+1}), \ldots, (\vec{lr}_M, \vec{r}_M', \vec{ur}_M))$ is a permissible θ-resource distribution. This completes this part of the proof.

(b) implies (c). If there exists a θ-permissible resource distribution w.r.t. $\vec{B}_\theta$ (with

lower bounds $\vec{lr}_i = A\vec{y}^0_i$, for all i, $k+1 \leq i \leq M$). Then,

$$\vec{lr}_i \leq \vec{r}_i \leq \vec{ur}_i,\ k+1 \leq i \leq M \Rightarrow \sum_{i\in\theta} \vec{lr}_i \leq \sum_{i\in\theta} \vec{r}_i \leq \sum_{i\in\theta} \vec{ur}_i$$

$$\Leftrightarrow \sum_{i\in\theta} \vec{lr}_\theta \leq \vec{B}_\theta$$

This completes this part of the proof.

(2) Let $((\vec{lr}_1, \vec{r}_1, \vec{ur}_1), \ldots, (\vec{lr}_k, \vec{r}_k, \vec{ur}_k))$ and $((\vec{lr}_{k+1}, \vec{r}_{k+1}, \vec{ur}_{k+1}), \ldots, (\vec{lr}_M, \vec{r}_M, \vec{ur}_M))$ be a $\bar{\theta}$- and θ- permissible resource distribution w.r.t. $\vec{B}_{\bar{\theta}}$ and $\vec{B}_\theta$ respectively. Since, $\vec{B}_{\bar{\theta}} + \vec{B}_\theta = \vec{b}$, ($\bar{\theta}$-,$\theta$-) permissible resource distributions constitute a permissible resource distribution. □

The optimization functions f are defined in terms of the information of all sites (i.e., $D(\vec{r}_1, \ldots, \vec{r}_M)$), whereas we need to work with the information only on a subset θ of sites. To do that, we define the notion of θ-localizer as follows.

Definition 17. *Let Ω be a constraint over $\vec{x}$, and $\mathbb{S}$ be the set of all splits of Ω, $f : \mathbb{S} \to \mathbb{R}$ be a function. Let $\theta = \{k+1, \ldots, M\}$ be a subset of sites $\{1, \ldots, M\}$, $\bar{\theta}$ its complement, $\mathbb{S}_\theta$ be the set of all θ-splits. Then, function $f_\theta : \mathbb{S}_\theta \to \mathbb{R}$ is called θ-localizer of f if for any $\vec{r}^0_1, \ldots, \vec{r}^0_k$, and for every two splits $D(\vec{r}^0_1, \ldots, \vec{r}^0_k, \vec{r}_{k+1}, \ldots, \vec{r}_M)$ and $D(\vec{r}^0_1, \ldots, \vec{r}^0_k, \vec{r}'_{k+1}, \ldots, \vec{r}'_M)$ in $\mathbb{S}$,*

$$f(\vec{r}^0_1, \ldots, \vec{r}^0_k, \vec{r}_{k+1}, \ldots, \vec{r}_M) \geq f(\vec{r}^0_1, \ldots, \vec{r}^0_k, \vec{r}'_{k+1}, \ldots, \vec{r}'_M) \Leftrightarrow$$

$$f_\theta(\vec{r}_{k+1}, \ldots, \vec{r}_M) \geq f_\theta(\vec{r}'_{k+1}, \ldots, \vec{r}'_M)$$

(i.e., f_θ preserves monotonicity for any resource instantiation outside of θ).

We are now ready to formulate a theorem to be used for concurrent (re-) decompositions of the global constraints.

Theorem 5. *Let $\Omega = A\vec{x} \leq \vec{b}$ be a global constraints, θ be a subset, say $\{k+1, \ldots, M\}$, of sites $\{1, \ldots, M\}$ and $\bar{\theta}$ be its complement, $\mathbb{S}$ be the set of all compact splits of Ω, $f : \mathbb{S} \to \mathbb{R}$ be a monotonic function and f_θ be its θ-localizer, $\vec{B}_\theta$ be a resource bound partition, and $D(\vec{r}_1^0, \ldots, \vec{r}_k^0)$ be a (partial) $\bar{\theta}$-split for sites outside θ. Let Pr be a subset of properties that contains* rp *and $\Phi_{\theta-Pr}$ be the condition (1) $\vec{r}_\theta \leq \vec{B}_\theta$ for the case that Pr contains just* rp*, and the conditions (2) $\vec{r}_\theta \leq \vec{B}_\theta, \vec{lr}_i \leq \vec{r}_i$ for every $i \in \theta$, for the case of Pr that contains both* rp *and* lc*. Then,*

1. *For Pr being the set of properties $\{rp, pcp\}$ or $\{rp, pcp, lc\}$, let $(\vec{r}'_{k+1}, \ldots, \vec{r}'_M)$ be a solution to the problem*

$$max f_\theta(\vec{r}_{k+1}, \ldots, \vec{r}_M)$$

$$s.t. \Phi_{\theta-Pr}$$

Then, $(\vec{r}_1^0, \ldots, \vec{r}_k^0, \vec{r}'_{k+1}, \ldots, \vec{r}'_M)$ is a solution to the problem

$$max f(\vec{r}_1, \ldots, \vec{r}_M)$$

$$s.t. \Phi_{Pr}$$

2. *For Pr being the set of properties $\{rp\}$ or $\{rp, lc\}$, let $(\vec{r}'_{k+1}, \ldots, \vec{r}'_M)$ be a solution*

to the problem

$$max f_{\theta}(\vec{r}_{k+1}, \ldots, \vec{r}_M)$$

$$s.t. \Phi_{\theta - Pr}$$

and $(\vec{r}'_1, \ldots, \vec{r}'_k)$ *be a solution to the problem*

$$max f_{\bar{\theta}}(\vec{r}_{k+1}, \ldots, \vec{r}_M)$$

$$s.t. \Phi_{\bar{\theta} - Pr}$$

Then, $(\vec{r}'_1, \ldots, \vec{r}'_k, \vec{r}'_{k+1}, \ldots, \vec{r}'_M)$ *is a solution to the problem*

$$max f(\vec{r}_1, \ldots, \vec{r}_M)$$

$$s.t. \Phi_{Pr}$$

Proof. The proof follows from the fact that f has a θ-localizer, and from Propositions 8 and 9. □

The next subsection presents the analytical expression for the objective function, under uniformity assumptions on updates.

2.6 Optimization Function

While the optimization problems from previous section are applicable to an arbitrary monotonic objective function, we now consider a specific optimization criterion in this section in more detail: *maximizing the probability of not violating local constraints.*

We provide an analytical expression of this probability function in terms of a parametric characterization of compact split decompositions, and polyhedron volume function (V) [CH79, Las83, Bea96].

2.6.1 Uniformity Assumptions

Assumptions described in this section are related to how the database instances are distributed over the space defined by Ω. We consider this distribution as a uniform one, i.e., each database instance on Ω has the same probability of occurrence. More formally, our assumptions are as follows:

- Ω is fully dimensional, and therefore $Volume(\Omega) \neq 0$.
- Not using local consistency (lc) property (f_{no-lc}): updates $\vec{x}$ of the database are uniformly distributed on the space defined by constraint Ω. Thus, if $Volume(\Omega) \neq 0$, then

$$prob[\vec{x}\ satisfies\ \mathbb{C}/\vec{x}\ satisfies\ \Omega] = \frac{Volume(\mathbb{C})}{Volume(\Omega)}$$

- Using local consistency (lc) property (f_{lc}): we define the following two predicates α: $\vec{x}$ *satisfy* Ω *and site* i *is being updated*, and β: $\vec{x}$ *satisfies* Ω *and one of the sites* $k+1, \ldots, M$ *is updated.*

  1. The probability p_i that a site i is being updated is given, for every $1 \leq i \leq M$. Therefore, for full decomposition:

$$prob[\vec{x} \textit{ satisfies } \mathbb{C}/\vec{x} \textit{ satisfies } \Omega] = \sum_{i=1}^{M} p_i \times prob[\vec{x} \textit{ satisfies } \mathbb{C}/\alpha]$$

     and for θ-decomposition:

$$prob[\vec{x} \textit{ satisfies } \mathbb{C}/\beta] = \sum_{i=k+1}^{M} \frac{p_i}{\sum_{j=k+1}^{M} p_j} \times prob[\vec{x} \textit{ satisfies } \mathbb{C}/\beta]$$

  2. The distribution of updates at site i (on variables $\vec{y}_i$) is uniformly distributed on Ω, when values for all variables, except $\vec{y}_i$, are fixed. We denote by $\vec{z}_i$ all variables on $\vec{x}$ except those on $\vec{y}_i$. Then for full decompositions:

$$\begin{aligned} prob[\vec{x} \textit{ satisfies } \mathbb{C}/\alpha] &= \frac{Volume(\mathbb{C}[\vec{z}_i \backslash \vec{z}_i^0])}{Volume(\Omega[\vec{z}_i \backslash \vec{z}_i^0])} \\ &= \frac{Volume(C_i)}{Volume(\Omega[\vec{z}_i \backslash \vec{z}_i^0])} \end{aligned}$$

     where $\vec{z}_i^0$ are the values for $\vec{z}$ before the update, $\mathbb{C}[\vec{z}_i \backslash \vec{z}_i^0]$ and $\Omega[\vec{z}_i \backslash \vec{z}_i^0]$ denote the formulas after $\vec{z}_i$ is replaced with $\vec{z}_i^0$ values.

For θ-compact split decompositions, we also assume that updates on the space defined by $D(\vec{r}_1, \ldots, \vec{r}_k)$ are uniformly distributed, and, therefore,

$$prob[\vec{x} \text{ satisfies } \mathbb{C}/\beta] = \int_{D(\vec{r}_1,\ldots,\vec{r}_k)} \frac{Volume(C_i)}{Volume(\Omega[\vec{z}_i \backslash \vec{z}_i^0])} d\vec{y}_1^0 \ldots d\vec{y}_k^0$$

2.6.2 Parametric Representation

The following is a parametric description of the optimization criteria for the probability of not violating local constraints.

Proposition 10. *Let $\Omega = A\vec{x} \leq \vec{b}$ be a constraint, $\vec{x}^0 = (\vec{y}_1^0, \ldots, \vec{y}_M^0)$ be a database instance, p_i be the probability that an update arrives at site i, θ be a subset of sites (say $\{k+1, \ldots, M\}$) and $\bar{\theta}$ be its complement, $D(\vec{r}_1, \ldots, \vec{r}_k)$ be a (partial) $\bar{\theta}$-split of Ω, and f_{no-lc} and f_{lc} are the probability of not violating local constraints as defined in Subsection 2.6.1. Then, under the uniformity assumptions of Subsection 2.6.1, the following holds:*

1. *Not using local consistency (lc) property: the function f_{no-lc} is monotonic and has a θ-localizer as follows:*

$$f^{\theta}_{no-lc}(\vec{r}_{k+1}, \ldots, \vec{r}_M) = \prod_{i=k+1}^{M} V(n_i, A_i, \vec{r}_i)$$

2. *Using local consistency (lc) property: the function f_{lc} is monotonic and has a*

θ-localizer as follows:

$$f_{lc}^{\theta}(\vec{r}_{k+1}, \ldots, \vec{r}_M) = \sum_{i=k+1}^{M} \frac{p_i}{P} V(n_i, A_i, \vec{r}_i) \times I$$

where $\vec{b}_i = (\vec{b} - \sum_{(j=1, j\neq i)}^{M} \vec{A}_j \vec{y}_j^0)$, $P = \sum_{j=k+1}^{M} p_j$, *and*

$$I = \int_{D(\vec{r}_1, \ldots, \vec{r}_k)} \frac{1}{V(n_i, A_i, \vec{b}_i)} d\vec{y}_k^0 \ldots d\vec{y}_1^0$$

Proof. First, f is monotonic since it is a probability function. Now, we prove that f has θ-localizers.

Proof of 1 (not using local consistency). We know that the probability to satisfy $\mathbb{C}$ given that Ω is satisfied, is given by:

$$\frac{Volume(\mathbb{C})}{Volume(\Omega)}$$

but, $Volume(\mathbb{C}) = \prod_{i=1}^{M} Volume(C_i)$, since different C_i's are defined in disjoint set of variables. We denote $Volume(C_i)$ as $V(n_i, A_i, \vec{r}_i)$. Then,

$$f_{no-lc}(\vec{r}_1, \ldots, \vec{r}_M) = \frac{\prod_{i=1}^{M} V(n_i, A_i, \vec{r}_i)}{V(n, A, \vec{b})}$$

Let $\mathbb{V} = \prod_{i=1}^{k} V(n_i, A_i, \vec{r}_i)$. Then, it is easy to see that for any two splits of Ω,

$D(\vec{r}_1, \ldots, \vec{r}_k, \vec{r}'_{k+1}, \ldots, \vec{r}'_M)$ and $D(\vec{r}_1, \ldots, \vec{r}_k, \vec{r}''_{k+1}, \ldots, \vec{r}''_M)$

$$\frac{\mathbb{V} \times \prod_{i=k+1}^{M} V(n_i, A_i, \vec{r}'_i)}{V(n, A, \vec{b})} \geq \frac{\mathbb{V} \times \prod_{i=k+1}^{M} V(n_i, A_i, \vec{r}''_i)}{V(n, A, \vec{b})} \Leftrightarrow$$
$$\frac{\prod_{i=k+1}^{M} V(n_i, A_i, \vec{r}'_i)}{V(n, A, \vec{b})} \geq \frac{\prod_{i=k+1}^{M} V(n_i, A_i, \vec{r}''_i)}{V(n, A, \vec{b})}$$

because $\mathbb{V}$ is a non-negative constant. Therefore, f_{no-lc} is monotonic and its θ-localizer is equivalent to

$$f^{\theta}_{no-lc} = \prod_{i=k+1}^{M} V(n_i, A_i, \vec{r}_i)$$

This completes this part of the proof.

Proof of 2 (using local consistency). We know that the probability to satisfy $\mathbb{C}$ given that Ω is satisfied by $\vec{x}^0$, is given by:

$$\sum_{i=1}^{M} p_i \times \frac{Volume(C_i)}{Volume(\Omega[\vec{z}_i \backslash \vec{z}^0_i])}$$

but, $\Omega[\vec{z}_i \backslash \vec{z}^0_i]$ is equivalent to $A_i \vec{y}_i \leq (\vec{b} - \sum_{(j=1, j \neq i)}^{M} A_j \vec{y}^0_j)$, and denoting the right hand side by $\vec{b}_i$. Then, $Volume(\Omega[\vec{z}_i \backslash \vec{z}^0_i])$ is the volume $A_i \vec{y}_i \leq \vec{b}_i$, i.e., $V(n_i, A_i, \vec{b}_i)$. Let $\mathbb{V} = \sum_{i=1}^{k} p_i V(n_i, A_i, \vec{r}_i)/V(n_i, A_i, \vec{b}_i)$. Then, for any two splits of Ω, $D(\vec{r}_1, \ldots, \vec{r}_k,$

$\vec{r}'_{k+1},\ldots,\vec{r}'_M)$ and $D(\vec{r}_1,\ldots,\vec{r}_k,\vec{r}''_{k+1},\ldots,\vec{r}''_M)$,

$$\begin{aligned}\mathbb{V}+\sum_{i=k+1}^{M} p_i\times\frac{V(n_i,A_i,\vec{r}'_i)}{V(n_i,A_i,\vec{b}_i)}\geq\mathbb{V}+\sum_{i=k+1}^{M} p_i\times\frac{V(n_i,A_i,\vec{r}''_i)}{V(n_i,A_i,\vec{b}_i)}\Leftrightarrow\\ \sum_{i=k+1}^{M} p_i\times\frac{V(n_i,A_i,\vec{r}'_i)}{V(n_i,A_i,\vec{b}_i)}\geq\sum_{i=k+1}^{M} p_i\times\frac{V(n_i,A_i,\vec{r}''_i)}{V(n_i,A_i,\vec{b}_i)}\end{aligned}\tag{2.13}$$

because $\mathbb{V}$ is a non-negative constant for $D(\vec{r}_1,\ldots,\vec{r}_k,\vec{r}'_{k+1},\ldots,\vec{r}'_M)$ and $D(\vec{r}_1,\ldots,\vec{r}_k,\vec{r}''_{k+1},\ldots,\vec{r}''_M)$. Now, we show which is the θ-localizer of f_{lc}. Since, $\vec{b}_i=\vec{b}-\sum_{(j=1,j\neq i)}^{M} A_j\vec{y}^0_j$, for all i, $1\leq i\leq M$, function $V(n_i,A_i,\vec{b}_i)$ depends of values $(\vec{y}^0_1,\ldots,\vec{y}^0_k)$ and $(\vec{y}^0_{k+1},\ldots,\vec{y}^0_{i-1},\vec{y}^0_{i+1},\ldots,\vec{y}^0_M)$, for all i, $k+1\leq i\leq M$. However, $(\vec{y}^0_1,\ldots,\vec{y}^0_k)$ are values outside of θ, that satisfy $\mathbb{C}_p$. Then, for every $(\vec{y}^0_1,\ldots,\vec{y}^0_k)$ with those properties,

$$\sum_{i=k+1}^{M} p_i\times V(n_i,A_i,\vec{r}'_i)\int_{D(\vec{r}_1,\ldots,\vec{r}_k)}\frac{1}{V(n_i,A_i,\vec{b}_i)}d\vec{y}^0_1\ldots d\vec{y}^0_k$$

Finally, dividing by the constant $\sum_{j=k+1}^{M} p_j$,

$$\sum_{i=k+1}^{M}\frac{p_i}{\sum_{j=k+1}^{M} p_j}\times V(n_i,A_i,\vec{r}'_i)\int_{D(\vec{r}_1,\ldots,\vec{r}_k)}\frac{1}{V(n_i,A_i,\vec{b}_i)}d\vec{y}^0_1\ldots d\vec{y}^0_k$$

This is the θ-localizer of f_{lc}. This completes the proof. □

Proposition 10 characterizes the θ-localizer for the probability of not violating local constraints, based on uniformity update assumptions (Section 2.6.1), and convex polyhedron volume ($V(n_i,A_i,\vec{r}_i)$). This volume calculation has been addressed

in [CH79, Las83, Bea96]. Those papers prove that under certain conditions the volume exists, and provide algorithms to compute it.

For the individual partition case, the volume calculation is easy, since local constraints are on individual variables. Those constraints define a (multi-dimensional) rectangle. More specifically,

$$V(C_{k+1}, \ldots, C_n) = \prod_{k+1}^{n} l_i \tag{2.14}$$

where l_i is the length of the i^{th} side of the rectangle. This simple formula gives Proposition 11 below, which is the simplification of Proposition 10 for the individual partition case.

Proposition 11. *Let $\Omega = A\vec{x} \leq \vec{b}$ be a constraint, $\vec{x}^0 = (\vec{x}_1^0, \ldots, \vec{x}_n^0)$ be a database instance, p_i be the probability that an update arrives at site i, θ be a subset of sites (say $\{k+1, \ldots, n\}$), $\mathbb{C}_p(u_{11}, u_{21}, \ldots, u_{1k}, u_{2k})$ be a partial split of Ω, and f_{no-lc} and f_{lc} are the probability of not violating local constraints as defined in Subsection 2.6.1. Then, under the uniformity assumptions of Subsection 2.6.1 hold,*

1. *Not using local consistency (lc) property: the function f_{no-lc} is monotonic and its θ-localizer is as follows:*

$$f^{\theta}_{no-lc}(u_{1k+1}, u_{2k+1}, \ldots, u_{1n}, u_{2n}) = \prod_{i=k+1}^{n} (u_{2i} - u_{1i})$$

2. *Using local consistency (lc) property: the function f_{lc} is monotonic and its θ-localizer is as follows:*

$$f^{\theta}_{lc}(u_{1k+1}, u_{2k+1}, \ldots, u_{1n}, u_{2n}) = \sum_{i=k+1}^{n} \frac{p_i \times (u_{2i} - u_{1i})}{\sum_{j=k+1}^{n} p_j} \times I$$

where

$$I = \int_{x_1=u_{11}}^{u_{21}} \ldots \int_{x_1=u_{1k}}^{u_{2k}} \frac{1}{(v_{2i} - v_{1i})}\, dx_1 \ldots dx_k$$

$$v_{1i} = Max_{(b'_l/a_{li}<0)} \left\{ \frac{b'_l}{a_{li}} \right\},$$

$$v_{2i} = Min_{(b'_l/a_{li}>0)} \left\{ \frac{b'_l}{a_{li}} \right\},$$

$$\vec{b'} = (\vec{b} - \sum_{(j=1, j\neq i)}^{n} A_j x_j^0), \; and$$

A_j *is the* j^{th} *column of* A.

Therefore, optimization problems defined in Theorems 4 and 5 can use f^{θ}_{no-lc}, f^{θ}_{lc} or equivalent objective functions. Note that the optimization problem has linear constraints, and a non-linear objective function. An algorithm to solve this problem is presented in Section 2.8. Now, we present our distributed protocol.

2.7 Distributed Protocol

In this section we describe a protocol to manage linear arithmetic constraints over a distributed system. In fact, the protocol manages resource distributions as a mechanism to manage updates and constraint decompositions (as is described in Subsection 2.5.3). The protocol is general, in the sense that it can be applied to different network and system architectures.

First, we explain a distributed transaction primitive called RESOURCE - TRANSFER that is used in the protocol. RESOURCE-TRANSFER(i,j,$\vec{rc}$) works on a pair of sites, and transfers the *resource-contribution* $\vec{rc}$ from the *giving* site i to the *receiving* site j. The effect of the transfers is that the upper resource bound $\vec{ur}_i$ at site i will be decreased by $\vec{rc}$, after which $\vec{ur}_j$ at site j will be increased by $\vec{rc}$. It is assumed that standard distributed transaction techniques are used to assure that (1) under no circumstances (possibly involving failures) resource-contribution is added to $\vec{ur}_j$ of the receiving site before it has been reduced from $\vec{ur}_i$ of the giving site, and (2) the standard ACID properties (i.e., atomicity, consistency, isolation, and durability) of distributed transactions. Important to note is that distributed transaction protocols to ensure these properties (e.g., two phase commit) are less expensive for transactions involving two sites only, as is done in our RESOURCE-TRANSFER.

We can now provide the basic assumptions for our protocol which are as follows:

Distributed Protocol Assumptions

1. The global database is abstracted by real values for the vector of variables $\vec{x} = (\vec{y}_1, \ldots, \vec{y}_M)$, where $(\vec{y}_1, \ldots, \vec{y}_M)$ is a partition of $\vec{x}$ [12].

2. A set of M distributed sites, where at each site i, $1 \leq i \leq M$, variables $\vec{y}_i$ are maintained. Since $(\vec{y}_1, \ldots, \vec{y}_M)$ is a partition of $\vec{x}$, i.e., there is no variable replication [13], i.e., $\vec{y}_i \cap \vec{y}_j = \emptyset$, for every $i \neq j$. Furthermore, each site i, $1 \leq i \leq M$, has a local transaction manager that guarantees the standard ACID properties (i.e., atomicity, consistency, isolation, and durability), as well as back-up and recovery from failure.

3. The global constraint is of the form $\Omega = A\vec{x} \leq \vec{b}$, i.e., Ω is a system of linear constraints. Local constraints are given by compact splits of Ω. Every site i maintains in addition to its local instance $\vec{y}_i^0$ and the global constraint Ω, the triple $(\vec{lr}_i, \vec{r}_i, \vec{ur}_i)$, where $((\vec{lr}_1, \vec{r}_1, \vec{ur}_1), \ldots, (\vec{lr}_M, \vec{r}_M, \vec{ur}_M))$ is a permissible resource distribution (thus the local constraint $A_i\vec{y}_i \leq \vec{r}_i$ is also implicitly given). Safe (re-) decompositions will be done by updating the permissible resource distribution.

4. When an update is required at site k, and if the new update (i.e., for $\vec{y}_i$) satisfies the current local constraint at site k, the update is performed. If it does not satisfy its local constraint, then, site k designates one site as a *coordinator* and

[12] Any database model, e.g., relational or object-oriented, can be used, we assume that values for $\vec{x}$ are either explicitly stored in the actual database or expressed as views (e.g., aggregations).
[13] If there is replication, i.e., a variable x appears at two sites, it can be reduced to a non-replica case by replacing x with x_1 and x_2 and adding the constraint $x_1 = x_2$ to Ω.

sends to it a request for the minimal resource-contribution necessary to make the update (i.e., by Proposition 9, the minimal $\vec{rc}_k$ is $\vec{ur}_k - \vec{lr}'_k$, where $\vec{lr}'_k$ is the new lower bound of k reflecting the new update).

5. The task of a coordinator is to facilitate a local update at site k (when the local update does not satisfy the local constraint C_k), coordinating the (re-) decomposition process. This is done by finding a set of sites θ, containing site k, and trying to create a new permissible θ-resource distribution, as follows. The coordinator asks for resource contribution from sites until either:

   (a) A subset of connected [14] (to the coordinator) and operational sites θ is found, for which $\vec{lr}'_\theta \leq \vec{ur}_\theta$, where $\vec{lr}'_\theta$ is the cumulative lower bound for θ (i.e., $\vec{lr}'_\theta = \sum_{i\in\theta} \vec{lr}'_i$) if the new update(s) in θ were reflected, (i.e., by Proposition 9, there exists a compact split of Ω satisfying resource partition $\vec{ur}_\theta$ and local consistency w.r.t. database instances in θ), and a new permissible θ-resource distribution is created, or

   (b) For the maximal set θ^* of sites that are connected (to the coordinator) and operational, and $\vec{lr}'_{\theta^*} \not\leq \vec{ur}_{\theta^*}$ (i.e., by Proposition 9 there does not exist a compact split of Ω satisfying resource partition $\vec{ur}_\theta$ and local consistency w.r.t. database instances in θ). Then, the update is refused.

In case (a), resources will be re-distributed in the optimal way in the sense of

[14] By "connected" we mean that every site on θ can send messages to any other site in θ.

Theorem 5.

6. Exchanging resource contributions (thus modifying the current upper resource bounds in the resource distribution) is only done via the distributed transaction primitive RESOURCE-TRANSFER. Each *giving* site can provide a resource contribution $\vec{rc}$ such that $\vec{0} \leq \vec{rc} \leq \vec{ur}_i - \vec{lr}_i$, i.e., local consistency will still be preserved. When the upper resource bound $\vec{ur}_i$ is reduced or increased at site i, the local protocol adjusts its resource $\vec{r}_i$ accordingly (see Section 2.5.3).

7. Failure model: both sites and communication links may fail, but persistent storage does not fail. We assume that the site failures stop site execution without performing any incorrect actions. Communication link failures may separate the sites into more than one connected component of communicating sites (θ's).

First, we discuss the properties we would like to guarantee, and then, a single coordinator protocol is presented as a specific architecture case implementation.

2.7.1 Properties

When a local constraint is violated, our protocol performs distribute processing. In order to guarantee a correct distributed processing, we propose the following properties.

CSOL Properties

1. Global and Local Consistency: every database instance $\vec{x} = (\vec{y}_1, \ldots, \vec{y}_M)$ must satisfy the global constraint Ω; every local instance $\vec{y}_i$ at every site i must satisfy the local constraint C_i.

2. Partial (θ-)Decomposition Soundness: If the protocol performs a (re-) decomposition of constraints in θ, then there must exist a safe (compact split) decomposition of Ω that satisfies resource partition $\vec{ur}_\theta$ (i.e., the global resource bound in θ) and local consistency w.r.t. the current database instance.

3. Last-Resort Update Refusal: Let $\vec{y}_i^0$ be a new update for site i, and let θ^* be a maximal set of operational and connected sites that contains the site i (i.e., no resources outside θ^* are available). Last-Resort Update Refusal means that the protocol refuses the update $\vec{y}_i^0$ at site i only if there does not exist a safe (compact split) decomposition of Ω that satisfies resource partition $\vec{ur}_{\theta^*}$ (i.e., the global resource bound in θ^*) and local consistency w.r.t. the current database instance.

4. θ-Optimality: if a protocol performs (re-)decompositions of sites in θ, it is must be optimal in the sense of Theorem 5.

Global and local consistency are standard properties that we would like to preserve. Partial θ-Decomposition Soundness says, intuitively, that the protocol does the best under the circumstances, i.e., using only the knowledge of resources at sites in θ. It also

implies that resources are not lost because of failures, because Partial θ-Decomposition Soundness must hold at all times, including times after (local) recoveries from failures. Finally, Last-Resort Update Refusal says that the protocol refuses updates (and re-decompositions), only when there is no choice under the circumstances, i.e., no site outside of θ^* can be reached (that is, a θ^* is maximal set) and, based just on the information at sites in θ^*, we cannot guarantee satisfaction of global and local consistency.

Now we are able to present the main result of this section, which is a theorem guaranteeing CSOL-properties under the distributed protocol assumptions.

Theorem 6. *Any protocol that uses exclusively $RESOURCE-TRANSFER$ primitive for exchanging resources, is guaranteed to satisfy (1) safety and local consistency (and thus global consistency), (2) partial θ-decomposition soundness, (3) last-resort update refusal, and (4) θ-decomposition optimality.*

Proof. Local consistency follows from the fact that RESOURCE-TRANSFER can only reduce resources such that local consistency is satisfied (Distributed Protocol Assumption 6). Global consistency follows from the fact that RESOURCE-TRANSFER never create resources (i.e., $\vec{rc} \leq \vec{ur}_i - \vec{lr}_i$) and it first reduces resource-contribution from the giving site and then add it to the receiving site (RESOURCE-TRANSFER assumption). Therefore, the protocol only produces permissible resource distribution. Finally, partial θ-decomposition soundness, last-resort update refusal, and θ decomposition optimality follow directly from assumption 5) of our Distributed Protocol Assumptions. □

We exemplify an instance of the protocol, which has a single coordinator, in the next subsection.

2.7.2 Protocol with one Coordinator

Here we exemplify an instance of the distributed protocol for one-coordinator architecture. The suggested protocol is based on RESOURCE-TRANSFER primitive, and the Distributed Protocol Assumptions. In addition to that, we assume the following.

1. Coordinator corresponds to site p. Since, there is only one coordinator, it knows the complete resource distribution among sites, and time to time decides to (1) collect information from sites (i.e., lower bounds), and (2) re-decompose without any non-coordinator site requirement.

2. Each site i has an underlying layer mechanism to inform whether or not a site j is connected and operative. We assume that such a mechanism triggers a variable $ALERT_{ij}$ which indicates that site j is disconnected (w.r.t. site i) or inoperative.

Now we are ready to describe the implementation of our protocol at non coordinator (regular) and coordinator sites.

Activities by Non-Coordinator Sites

Each site checks if local updates satisfies the current local constraints. When an update $\vec{y}_k$ arrives to site k, this site performs the following.

1. (a) If $\vec{y}_k$ satisfies the local constraint, then perform the update.

   (b) If $\vec{y}_k$ does not satisfy the local constraint. Then, site k sends a request to the coordinator with the necessary resource-contribution $\vec{rc}_k$ to make the update. This resource-contribution is calculated as follows:

$$\vec{rc}_k = \begin{cases} \vec{A}_{ki}\vec{y}_k - ur_{ki} & \text{if } \vec{A}_{ki}\vec{y}_k > ur_{ki}, \\ 0 & \text{otherwise.} \end{cases}$$

   Then, site k waits until the coordinator provides the resource-contribution needed or $ALERT_{kp}$ is triggered.

2. When site k receives resource-contribution from the coordinator, then the resources are updated. If there are enough resources, then the update is performed. Otherwise, site k rejects the update.

3. If site k receives the $ALERT_{kp}$ saying that the coordinator is not connected or operative, site k reject the update.

When a site k receives a request from the coordinator, asking for resource - contribution, site k performs the following:

1. Site k decides resource-contribution $\vec{rc}_k$ to provide to the coordinator as follows: $\vec{0} \leq \vec{rc}_k \leq max\{\vec{0}, \vec{ur}_k - \vec{lr}_k\}$. Then, site k initiates RESOURCE-TRANSFER(k,p,$\vec{rc}_k$) to the coordinator.

Activities by Coordinator p

Coordinator p maintains set ξ for all sites requesting resource-contribution from it, and a vector $\vec{rc}_p$ with the current resource-contribution holds by the coordinator. Then, the coordinator performs the following:

1. When a request $\vec{rc}_k$ is received from a site k, and the coordinator has enough resources (i.e., $\vec{rc}_p \geq \vec{rc}_k$) and ξ is empty. Then, the coordinator reduces $\vec{rc}_k$ from $\vec{rc}_p$ and provides resource-contribution $\vec{rc}_k$ to site k using primitive RESOURCE-TRANSFER(p,k,$\vec{rc}_k$). Otherwise, the following is performed.

2. The coordinator include site k in ξ and decides an initial set θ of sites to ask for resource-contribution. For each site $i \in \theta$ sends a request asking for it. Coordinator waits until the resource-contributions arrive or $ALERT_{pi}$ is triggered.

3. All possible answers from θ have been received (i.e., all connected and operational sites in θ have ended RESOURCE-TRANSFER primitive), and there are no enough resources, i.e., $\vec{lr}'_\theta \not\leq \vec{ur}_\theta$. Then,

   (a) θ can be increased, the coordinator increases it and sends a request asking for resource-contribution for the new sites in θ.

   (b) θ can not be increased. Then, for each site $i \in \xi$, the coordinator initiates primitive RESOURCE-TRANSFER(p,i,$\vec{0}$), and deletes site i from ξ.

4. If enough resources have been collected, i.e., i.e., $\vec{lr}'_\theta \leq \vec{ur}_\theta$, coordinator decides

according to Theorem 5, how much resource-contribution $\vec{rc}_i^*$ is distributed to sites in ξ and θ. Then, for each site initiates RESOURCE-TRANSFER(p,i,$\vec{rc}_i^*$) and deletes sites from ξ.

This implementation satisfies the Distributed Protocol Assumptions (see Section 2.7) and uses the primitive RESOURCE-TRANSFER to exchange resources, thus by Theorem 6 it satisfies all our CSOL-properties. The following section presents the implementation and some experiment of our optimization framework.

2.8 Algorithms, Implementation and Experiments

This section presents a general algorithm to solve the optimization problems presented in Sections 2.4 and 2.5. Experimental results are presented for a single partition case. We use a set of experimental linear systems to show the algorithm behavior for varying numbers of constraints and variables. We use the problem size (product of the number of variables and the number of constraints) and the algorithm's running time as main measures. The algorithm was implemented using visual C++ 4.0, and was run it on a 120 Mhz PC compatible.

The optimization problem has linear constraints, and a non-linear objective function, based on volume representation. In general, volume representation is based on vertex enumeration (implicit or explicit), or recursive representations [Las83, Bea96], where some of its properties are: positive homogeneous function of its right-hand-side

vector, local convexity, and local concavity. However, these properties are not enough to guarantee optimal solutions using a global search algorithm.

We use a local search algorithm [LH96, Glo89, Jea91] to solve our optimization problem. The structure is as follows: a number of local searches are performed, where for each one, the algorithm checks if the local optimum is better than the current objective function value. This procedure is repeated until there is no acceptable neighborhood possible, i.e., a neighborhood where the objective function is locally concave.

Minimally-Constrained Safe Decomposition Algorithm

Let Pr be a subset of properties $\{compactness, lc, pcp, rp\}$, where properties *compactness* or *rp* must be included, and $\theta = \{k+1, \ldots, M\}$ be a subset of sites $\{1, \ldots, M\}$.

- Step 0. Assign an initial solution to $\vec{r}_{k+1}, \ldots, \vec{r}_M$, the objective function $f^*_{Pr} = f_t(D(\vec{r}_{k+1}, \ldots, \vec{r}_M))$, and select an initial acceptable neighborhood s.

- Step 1. Perform a local search in s, obtain the solution $\vec{h}^*_{k+1}, \ldots, \vec{h}^*_M$ of

$$maximize f_{Pr}(D(\vec{h}_{k+1}, \ldots, \vec{h}_M))$$

$$s.t.\ \Phi_{Pr}(\vec{h}_{k+1}, \ldots, \vec{h}_M) \wedge$$

$$(\vec{h}_{k+1}, \ldots, \vec{h}_M) \in s$$

- Step 2. If $f_{Pr}(D(\vec{h}^*_{k+1}, \ldots, \vec{h}^*_M)) > f^*_{Pr}$, then $f^*_{Pr} = f_{Pr}(D(\vec{h}^*_{k+1}, \ldots, \vec{h}^*_M))$, and $\vec{r}_i = \vec{h}^*_i, (k+1) \leq i \leq M$.

- Step 3. Select a new acceptable neighborhood s, and go to step 2. If there is no acceptable neighborhood, go to step 4.

- Step 4. Report the solution as: objective function f^*_{Pr}, and the solution $\vec{r}_{k+1}, \ldots, \vec{r}_M$.

The local search (step 1) is a non-linear optimization problem, with concave objective function. We use the Frank-Wolfe algorithm [BS79, Kam84], and volume algorithms [Las83, Bea96] to calculate at each iteration the gradient of f_{Pr}. This algorithm is as follows:

- Step 1.1 Let $q \leftarrow 0$ be an iteration index, and ϵ a stop condition. Obtain an initial solution and assign it to $\vec{w}_0$,

- Step 1.2 Obtain the solution of the following linear optimization problem:

$$Max\ \nabla f_{Pr}(D(\vec{w}_q))\ (\vec{h}_{k+1}, \ldots, \vec{h}_M)$$

$$s.t.\ \Phi_{Pr}(\vec{h}_{k+1}, \ldots, \vec{h}_M)$$

  where $\nabla f_{Pr}(D(\vec{w}_q))$ is the gradient of f_{Pr} evaluated at the point $\vec{w}_q$. Analyze the solution, and if: (a) it is not feasible, then the original system is infeasible, then stop, and (b) there exists the solution, then assign it to $\vec{v}$.

- Step 1.3 Get the optimal step λ^* as the solution of the problem:

$$Max\ f_{Pr}(D(\lambda\vec{v} + (1-\lambda)\vec{w}_q))$$

$$s.t.\ 0 \leq \lambda \leq 1$$

- Step 1.4 Assign to $w_{q+1} \leftarrow \lambda^*\vec{v} + (1-\lambda^*)\vec{w}_q$, if $\parallel w_{q+1} - w_q \parallel \leq \epsilon$ then go to step 1.5, or $q \leftarrow q+1$, and go to step 1.2.

- Step 1.5 Report $\vec{w}_{q+1}$ as the solution.

Note that steps 1.2 and 1.3 require additional algorithms. We use the simplex algorithm [BS79, CZ96] to solve the linear problem (step 1.2), and we use the bisecting search method [BS79] to solve the one-variable optimization problem (step 1.3).

The algorithm's complexity depends on the complexity of finding an acceptable neighborhood, and for each neighborhood visited, the complexity of n linear programs (Frank-Wolfe algorithm), and for each linear program, the complexity of M volume calculations. However, volume calculation complexity is mitigated for the smaller matrix range associated with each partition element.

Note that for the schema-based and individual partition case, we have only one acceptable neighborhood, and therefore we need to solve just step 2. The results for this special case are depicted in Table 2.1.

Table 2.1 summarizes the results for 21 experiments, where P is the problem number assigned, N is the number of variables, M the number of constraints, and Time is the

Table 2.1: Empirical Results

P	N	M	Time	P	N	M	Time
1	4	3	0.66	12	50	50	9.01
2	7	7	0.30	13	80	80	40.86
3	12	7	0.66	14	100	100	92.16
4	100	10	1.05	15	6	5	0.44
5	8	6	12.00	16	12	7	1.10
6	18	6	1.21	17	21	10	1.76
7	8	5	1.87	18	33	14	3.79
8	6	6	0.39	19	55	16	10.21
9	10	10	0.88	20	80	18	20.38
10	20	20	0.94	21	100	20	35.05
11	30	30	2.58	-	-	-	-

running time measured in seconds. The general structure of the experiments is as follows: experiments from 1 to 7 are random linear systems (with at least one solution), experiments from 8 to 14 correspond to scheduling problems, and experiments from 15 to 21 have a transportation problem structure.

Figure 2.3 shows that the running time does not behave exponentially in the size of the problem (notice that the x axis is the natural logarithm of the problem size). However, there is a high variance in this relation. The following figures present the time for each class of problem.

However, the relation between problem size and run time for the transportation type problem is clearer as shown in Figure 2.4. One observes a smoother relation between these variables.

Figure 2.5 shows the relation between these variables for the scheduling type problem. In the same way that the transportation case, this relation is smother than the

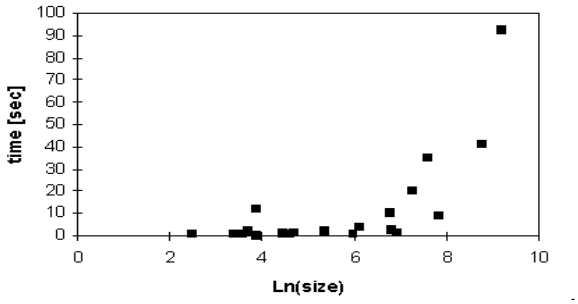

Figure 2.3: Experimental Run Time

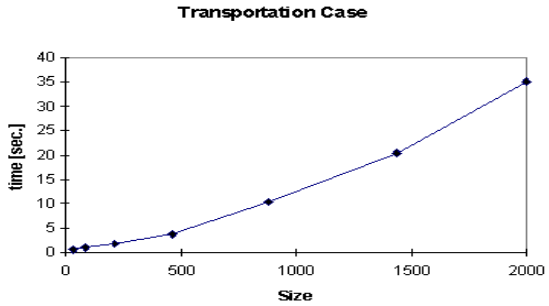

Figure 2.4: Run Time for Transportation Case

general case.

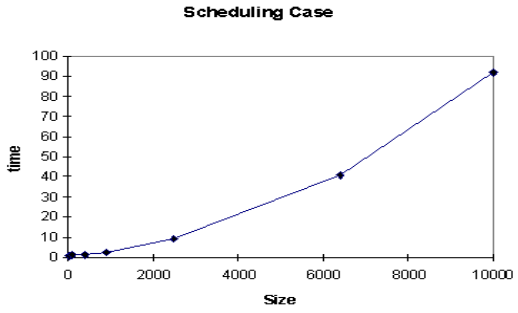

Figure 2.5: Run Time for Scheduling Case

The main conclusion of this experiment, is that the decomposition algorithm is feasible and scalable, especially when one considers that the restructuring of local constraints occurs infrequently. However, more experiments are necessary.

Chapter 3

OPTIMIZING MATERIALIZED VIEWS

3.1 Introduction

The evaluation of materialized views (queries) may require considerable computational effort because some materialized views (1) can share some intermediate results (views) with other materialized views, or (2) are complex enough to justify some intermediate pre-computed views. To reduce the effort to maintain views updated, some intermediate views can be materialized. However, how many and which intermediate views will be materialized will depend on many factors, such as view maintenance costs, response time, available storage, etc.

The materialized view (query) optimization problem has been studied in different contexts, such as: query optimization, view maintenance, and data warehouse design and configuration. However, none of these works have provided an explicit problem formulation in terms of materialized view interrelations, neglecting the possibility to take advantage from it.

In this chapter we consider the optimizing materialized views problem, i.e., selecting views to materialize in order to optimize a criterion, subject to a set of materialization constraints (maintenance time, available storage, etc.). We propose an *optimization framework* to decide the optimum way to materialize views, i.e., which additional views need to be materialized. We use an *expression*-DAG to express equivalent view evaluation plans. Then, we formulate an optimization problem to make a decision.

This chapter is organized as follows: Section 3.2 presents related work. Section 3.3 presents the contributions. Section 3.4 presents the problem characterization, where the view optimization is formulated based on shortest (cheapest) path in an expression-DAG. In Section 3.5 we present a linear-time algorithm when all possible view evaluation alternatives are available, and a local search strategy for the general case. Finally, Section 3.6 presents some experiments and the implementation of the general shortest path algorithm.

3.2 Related Work

The view (query) selection problem has been studied in different contexts, such as: query optimizations [NKOD95], view maintenance [CW91, GL95, GMS93, GM95, RSS96], and data warehouse design [Gup97, HRU96, GHRU97, YKL97, BPT97] and configuration [TS97]. However, in terms of the solution, current research provides *near-optimal* heuristics (without guarantee of the solutions' quality), very expensive optimal

exhaustive search algorithms, or they just address to special cases.

In particular, works [BPT97, RSS96, TS97, YKL97] provide frameworks, heuristics, and an exhaustive search algorithm in order to optimize the sum of response and maintenance time without any constraints. Ross [RSS96] has proposed view selection based on a minimization of maintenance cost. However, [RSS96] presents an exhaustive search algorithm, which is exponential (double exponential) in the number of possible views to be materialized. Work [YKL97] formulates the problem as one of *integer programming* in terms of the view evaluation plan. However, the number of evaluation plans is exponential in terms of the number of possible views to be materialized.

The results reported in [Gup97, HRU96, GHRU97, GM98] provide a formulation with storage constraint and time evaluation constraint. Three of them provide *near-optimal* heuristics (greedy algorithm). In particular, [Gup97, GHRU97] extend [HRU96], and present a formulation as an optimization problem. Works [Gup97, GHRU97] provide polynomial-time heuristics (in terms of the number of possible views to be materialized) for two special cases (AND and OR graphs), and *near-optimal* exponential-time greedy algorithm for AND-OR graphs. However, the solution quality is not guaranteed. Finally, [GM98] extends previous works, which minimizes the response and maintenance time of selected views, subject to a maximum maintenance time. However, the heuristics and algorithms still present the same previous behavior.

3.3 Contributions

This chapter addresses the optimal view materialization problem, where for a given set of materialized views (queries), one must decide which additional (intermediate) views should be materialized in order to reduce the overall maintenance effort, under some materialization constraints. A standard mechanism to represent intermediate views corresponds to AND-OR graphs [Gup97, HRU96, GHRU97, GM98]. However, there is no work in which the structural properties of such graphs are exploited. In general, this problem is NP-hard [RSS96, Gup97], because it corresponds to selecting a subset of elements from the set of all intermediate views, where the number of subsets is exponential in the number of additional views.

This research exploits the structure of the representation mechanism for intermediates views. More specifically, the contributions are as follows. First, it extends the expression-DAG (Direct Acyclic Graphs) [RSS96] as mechanism to represent compactly intermediate views (queries) using equivalence and operation nodes. It shows that equivalence nodes correspond to nodes in an AND-OR graph, and operation nodes correspond to AND arcs. It characterizes an expression-DAG in terms of its size and expression-paths [1] (i.e., complete view evaluation plans). However, while the size of a standard AND-OR graph is defined in terms of its nodes and arcs, the size of an expression-DAG is defined in terms of the cardinality of its operational nodes.

Second, an optimization framework is formulated in terms of the expression-DAG

[1] Note that, under certain conditions, AND-paths are equivalent to expression-paths.

structure. Thus, under certain objective function conditions, the problem of optimal selection of materialized views can be formulated as the constrained shortest path in an expression-DAG, i.e., a complete expression-path or AND-path. For this case, a linear-time algorithm (in terms of the expression-DAG size) is presented. Note that this special case can be found in important applications such as the case when the evaluation time is the critical variable, and therefore, while more intermediate views are materialized, the complete evaluation should be more efficient. A set of experiments was run, and the results suggest that our approach is feasible.

Third, for the general optimization problem and if the expression-DAG has all possible view evaluation plans, then the linear-time algorithm can be applied to obtain a solution. Note that this algorithm does not evaluate all possible equivalent evaluation plans, because all those which are subsumed by others are eliminated earlier. Finally, if the expression-DAG with all possible view evaluation plans is not available, a local search algorithm is presented. However, further research is necessary in this area.

3.4 View Materialization Characterization

In this section we characterize the optimization problem to support an efficient and cost effective view materialization and maintenance. First, we present the basic definitions based on the relational model. Then, we formulate the optimization problem, characterizing the search space and the objective function. Finally, we present an equivalent

formulation and we discuss their effective solutions.

3.4.1 Definitions

This subsection describes basic definitions and concepts based on the relational database model [Ull88]. In particular, view and view evaluation plan are described.

Definition 18. *A database DB is a collection of n-relations* $(r_1, \ldots, r_n)$ *over relational schemes* $(R_1, \ldots, R_n)$. *The set of relational schemes is called a database schema. Each relational schema is formed by a finite set of attributes names,* $A_i = \{A_{i1}, \ldots, A_{im_i}\}$, *and each attribute has a set of its possible values called domain.*

We consider the set $\mathbb{A}$ as the set of all attributes in DB, i.e., $\mathbb{A} = A_1 \cup \ldots \cup A_n$. The n-relations $(r_1, \ldots, r_n)$ is called an extensional database or simply database.

Definition 19. *A view* $\mathcal{V}$ *over a database DB is an expression of the form:*

DEFINE VIEW $\mathcal{V}$ **AS**

SELECT $\mathbb{B}$

FROM $T_1, T_2, \ldots, T_M$

WHERE $\mathbb{C}$

where $\mathbb{B}$ *is a subset of the set of attributes* $\mathbb{A}$, T_i *could be either a relational name (*R_i*'s) or other view name (*V_i*'s) on DB, called* base relations, *and* $\mathbb{C}$ *is a constraint called* selection *condition. We will denote view* $\mathcal{V}$ *as* $\mathcal{V}(T_1, \ldots, T_M)$.

Note that a view $\mathcal{V}$ is a derived relation, i.e., it is not included in the database schema. Therefore, in order to keep view $\mathcal{V}$ consistent with the data sources, it has to be maintained or re-evaluated from any relevant change produced at $T_1, \ldots, T_M$.

Definition 20. *Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be two views. We say that $\mathcal{V}_2$ is subsumed by $\mathcal{V}_1$, denoted by $\mathcal{V}_2 \models \mathcal{V}_1$, if $\mathcal{V}_2 \subseteq \mathcal{V}_1$, where $\subseteq$ refers to view containment. $\mathcal{V}_1$ is equivalent to $\mathcal{V}_2$, $\mathcal{V}_1 \equiv \mathcal{V}_2$, if $\mathcal{V}_1 \models \mathcal{V}_2$ and $\mathcal{V}_2 \models \mathcal{V}_1$.*

Intuitively, view $\mathcal{V}_1$ subsumes view $\mathcal{V}_2$ if for any legal database instance $\mathcal{V}_2$ can be derived from $\mathcal{V}_1$, i.e., $\mathcal{V}_2$ is contained by $\mathcal{V}_1$. In general, a view $\mathcal{V}$ can be decomposed into a set of equivalent views $\{\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_n\}$, such that $\mathcal{V} \equiv \mathcal{V}_0(\mathcal{V}_1, \ldots, \mathcal{V}_n)$. Then, we can use either $\{\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_n\}$ or $\mathcal{V}$ to answer $\mathcal{V}$. Note that $\mathcal{V}$ can be decomposed recursively, i.e., each one of its $\mathcal{V}_i$, $1 \leq i \leq n$, can be decomposed in $(\mathcal{V}_{i0}, \mathcal{V}_{i1}, \ldots, \mathcal{V}_{in_i})$, and so on.

We will say that a view decomposition is a view *evaluation plan* if a view $\mathcal{V}$ is decomposed recursively until all its base relations are reached. Note that two or more different view evaluation plans may share one or more intermediate views.

3.4.2 Optimal View Materialization Problem

In this subsection we formulate and characterize the problem of selecting a set of materialized views as an optimization one. Informally, given a set of materialized views (queries) $\mathcal{V}$, defined over a set of base-relations or views $\mathcal{R}$, we have to decide what additional views $\mathcal{V}^*$ should be materialized in order to optimize a criterion (maintenance

costs, response time, etc.), satisfying a set of materialization constraints (maintenance time, available storage, budget, etc.). Selecting one subset of additional views (evaluation plan) or another requires that the following issues be considered.

1. Incremental Evaluation Tradeoff: Selecting an evaluation plan with few views may spoil the performance, because materializing views will require more effort. On the other hand, evaluation plans with many views will require, when some data sources change (due to update, insert or delete) occurrences, more materialized view recalculations.

2. Design Constraints: There are some constraints that restrict the solutions, and some evaluation plans may not satisfy them. For instance, available storage, processing and view maintenance time, a limited budget, and any other resource constraints.

3. Selection Criteria: When several alternatives satisfy our design constraints, we need to define some criteria to select one among all possible. For example, time, cost, and storage measures of view materialization and maintenance could be the selected criterion.

The following definition is a general characterization of the optimization criterion and search space for the optimization problem formulation.

Definition 21. *Let $\mathcal{V}$ be a view. We say that the set $\mathbb{V}$ is the set of all equivalent*

evaluation plans (subset of materialized views) for $\mathcal{V}$, the function $f : \mathbb{V} \rightarrow \mathbb{R}$ is a real function that characterizes our selection criterion, and $\Psi(v), v \in \mathbb{V}$ is a constraint.

Therefore, the view selection problem corresponds to select a set of views ($\mathcal{V}^*$), i.e., an evaluation plan for $\mathcal{V}$, among all feasible alternatives (i.e., those $v \in \mathbb{V}$ that satisfy $\Psi(v)$), such that a criterion f is optimized. We will assume that the optimization is a minimization, and then the following optimization problem formulates the view selection problem.

$$\begin{aligned} &\min_{v} \{f(v)\} \\ &s.t.\ \Psi(v),\ v \in \mathbb{V} \end{aligned} \tag{3.1}$$

Note that [RSS96] uses a similar formulation to (3.1) but without constraints. Before we discuss how problem (3.1) can be solved effectively, we will concentrate on the more precisely characterization of set $\mathbb{V}$, function f, and constraint $\Psi(v)$. The following subsections address these characterizations.

3.4.3 Expression DAG

In this subsection we introduce *expression-DAGs* as a mechanism to represent equivalent view (query) evaluation plans, i.e., a characterization of the set $\mathbb{V}$. Expression-DAGs were originally introduced in [RSS96], we adopt the original definition of [RSS96], and extend the expression-DAG concept with some useful properties.

Definition 22. *An* expression DAG E_{DAG} *is a directed acyclic graph (acyclic digraph), represented by the pair* $E_{DAG} = (E, O)$*, where* $E = \{e_1, \ldots, e_n\}$ *is the set of* equivalence nodes*, and* $O = \{O_1, \ldots, O_m\}$ *the set of* operation nodes*, with the following properties:*

1. *An equivalence node has edges to one or more operation nodes.*

2. *An operation node contains an operator, has edges to one or more equivalence nodes, and its parent is an equivalence node.*

We denote by $\mid O_j \mid$ the *cardinality* of the *operation* node O_j, i.e., the total number of incoming and out-coming edges on O_j. In addition to that, we denote by $C(O_j)$ and $C(e_i)$ the children set (equivalence nodes and operation node respectively) of the operation node O_j and equivalence node e_i, respectively. Finally, we denote by $P(O_j)$ the parent (equivalence node) of O_j, and $L(E_{DAG})$ the set of all leaf nodes, i.e., those nodes without children.

Definition 23. *Let* $E_{DAG} = (E, O)$ *be an expression DAG, with equivalence node* $E = \{e_1, \ldots, e_n\}$ *and operation nodes* $O = \{O_1, \ldots, O_m\}$*. We denote the size of* E_{DAG} *by* $size(E_{DAG})$ *and the average cardinality by* $\bar{E}_{DAG}$*, defined as follow:*

$$size(E_{DAG}) = \sum_{O_i \in O} \mid O_i \mid \quad and \quad \bar{E}_{DAG} = \frac{1}{m} \sum_{O_i \in O} \mid O_i \mid$$

Expression-DAGs are used to compactly represent the space of equivalent view (query) *evaluation plans* in [RSS96], where equivalence nodes represent views, operation

nodes represent relational equivalence between parent and children, and the leaves of an expression -DAG correspond to the base relations. Furthermore, expression-DAGs are equivalent to AND-OR graphs (used in [Gup97, RSS96]) if the equivalence nodes are equivalent to the nodes in the AND-OR graph, and for each AND arc there exists an operation node and vice versa.

Definition 24. *A* path P_{st} *of length* q*, in an expression-DAG* $E_{DAG} = (E, O)$*, is a sequence of equivalence and operations nodes,* $P_{st} = (e_1 = s, O_{i_1}, e_2, O_{i_2}, \ldots, O_{i_q}, e_{q+1} = t)$*, where:*

$$s \in P(O_{i_1}), t \in C(O_{i_q}), \text{ and } e_j \in C(O_{i_{j-1}}) \cap P(O_{i_j}),\ j = 2, \ldots, q$$

nodes s *and* t *are the origin and destination respectively, and we say that* t *is connected to* s*.*

Note that a path is simply a sequence of equivalence and operation nodes. However, we would like to extend that concept to one that provides the notion of view evaluation plans. This extension is as follows.

Definition 25. *Let* E_{DAG} *be an expression-DAG, we say that an* expression-Path*, denoted by* $\pi = (E_\pi, O_\pi)$*, is a rooted expression-DAG, where for each* $e_i \in E_\pi$ *there exists only one* $O_j \in O_\pi$ *selected.*

It is easy to see that an expression-Path, from the root of E_{DAG} to the base relations,

represents a view evaluation plan, because each view is materialized using only one operation node.

An expression-DAG can be subdivided in sub expression-DAGs, which are a portion of an expression-DAG. We formalize this concept as follows.

Definition 26 (Sub-expression-DAG). *Let $E_{DAG} = (E, O)$ be an expression-DAG, we say that $E_{SDAG} = (E', O')$ is a* sub-expression-DAG *of E_{DAG} rooted in node e_i if E_{SDAG} is an expression-DAG with the following sets:*

$$E' = \{e_j \mid e_j \in E \wedge \exists P_{e_i e_j} \neq \emptyset\} \quad \textit{and} \quad O' = \{O_j \mid O_j \in O \wedge \exists P_{e_i O_j} \neq \emptyset\}$$

The following example shows an expression-DAG to represent view $\mathcal{V}$. Note that we use a different notation than [RSS96], where the equivalence nodes are represented as a circle (views), and the operation nodes as an inverted black triangle.

Example 2. *Consider the relation schema (AB), (CDE), and (FG), where A, B, C, D, F, G are the attribute names, and a view $\mathcal{V}$ defined over this schema as follows:*

$$\mathcal{V} = \pi_{A,D,G}(\sigma_{(A=C)\wedge(B<C\vee B<D)\wedge(G<E)\wedge(F=f)\wedge(B=b)}(AB \times CDE \times FG))$$

Figure 3.1 *presents an expression-DAG $E_{DAG} = (E, O)$ for view $\mathcal{V}$, where the set of equivalence nodes is $E = \{V, V_1, V_2, V_3, V_4, AB, CDE, FG\}$, and the set of operation nodes $O = \{O_1, O_2, O_3, O_4, O_5\}$, each one associated with a specific operation.*

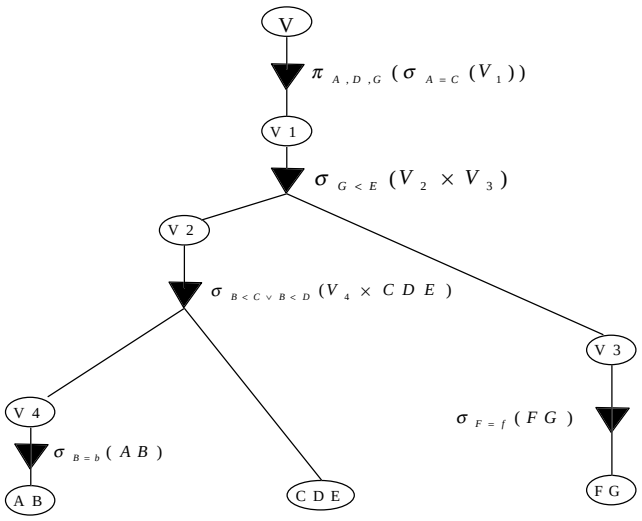

Figure 3.1: An Expression-DAG of $\mathcal{V}$

An expression-DAG represents equivalent view evaluation plans for a given view. However, there are some of those evaluation plans that are implicitly represented. Thus, from Figure 3.1, view V_1 can directly evaluated from base relations AB, CDE, and FG, and in the same sense, view V_2 from AB and CDE. We formalize those additional and implicit ways to evaluate view V as follows.

Definition 27. *Let $E_{DAG} = (E, O)$ be an expression-DAG for a view $\mathcal{V}$. Then, we say that $E^+_{DAG} = (E, O^+)$ is the* transitive closure *of E_{DAG} if E^+_{DAG} is an acyclic digraph and set O^+ is defined as follows:*

$$O^+ = \{O^+_j \mid \forall e_i \neq e_j \in E,\ \exists\, P_{e_i e_j} \neq \emptyset \wedge \\ O^+_j \in C(e_i) \wedge C(O^+_j) = \cup_{O_k \in P_{e_i e_j}} C(O_k)\} \tag{3.2}$$

The transitive closure E^{+}_{DAG} of an E_{DAG} has all possible equivalent evaluation plans for a given view $\mathcal{V}$, i.e., it characterizes the search space ($\mathbb{V}$) for the view selection problem. However, the main drawback in this concept is that $size(E^{+}_{DAG})$ grows exponentially in the number of equivalence nodes (views) in E_{DAG}.

3.4.4 Objective Function

In this subsection we characterize the optimization criterion by means of a function f. To characterize f we adopt the same optimization function proposed in [Gup97]. We consider that a set of views (queries) $\mathcal{V}$ need to be answered and that a set of views $\mathcal{V}^*$ have been selected to be additionally materialized. Then, function f has the following items.

Requirement Cost: this cost is associated to answer views in $\mathcal{V}$, i.e., each time that a view $\mathcal{V}_i \in \mathcal{V}$ is required, the system will compute it from the additional materialized views $\mathcal{V}^*$ (if there are some) or from the base relations. Therefore, the cost incurred by such operations represent the *requirement cost*. We use $C_A(\mathcal{V}_i, \mathcal{V}^*)$ to denote this cost.

Maintenance Cost: this cost is associated with the marginal maintenance of views in $\mathcal{V}^*$. Since views in $\mathcal{V}^*$ are all materialized views, when a base relation changes (update, delete, or insert), views in $\mathcal{V}^*$ have to be maintained. The cost of such maintenance is captured in the *maintenance cost*. We use $C_M(\mathcal{V}^*_i, \mathcal{V}^*)$ to denote

the maintenance cost of materialized view $\mathcal{V}_i^*$.

Finally, we consider that for each view $\mathcal{V}_i \in \mathcal{V}$ there exists a frequency β_i representing the number of times that view $\mathcal{V}_i$ is required per unit time. In the same way, for each materialized view $\mathcal{V}_j^* \in \mathcal{V}^*$ there exists a frequency λ_j representing the number of times that view $\mathcal{V}_j^*$ is maintained per unit time. Therefore, the objective function f is expressed as follows.

$$f = \sum_{i \in \mathcal{V}} \beta_i C_A(\mathcal{V}_i, \mathcal{V}^*) + \sum_{j \in \mathcal{V}^*} \lambda_j C_M(\mathcal{V}_j^*, \mathcal{V}^*) \tag{3.3}$$

Function f models the trade off between the number of materialized views and the total cost. Thus, when more additional views are materialized, the cost to answer views in $\mathcal{V}$ is reduced, but the cost to maintain materialized views increases. The tradeoff associated between these costs forms the crux of the materialization view problem. The next subsection presents two additional, but equivalent, formulations for our problem.

3.4.5 Optimization Problem

In this section we re-formulate problem (3.1) in two equivalent formulations. One of them is based on selecting a subset of views $\mathcal{V}^*$ among all candidate views to be materialized, the other, is based on recursive programming, taking advantage from the E_{DAG} representation. Before we present such formulations, we define the *resource*

constraint concept as follows.

Definition 28 (Resource Constraint). *Let $\mathcal{V}^*$ be the set of selected materialized views, and T be the maximum resource available. Then, a resource constraint is a constraint saying that the resource used by views in $\mathcal{V}^*$ must be lower or equal to T.*

In general, this type of constraint has been associated with storage (hard disk capacity), processing time, or budget [Gup97, HRU96, GM98]. Independently of which meaning the constraint has, we formulate the optimization problem in terms of that type of constraint.

Definition 29. *Let $\mathcal{V}$ be a set of views, and $\mathcal{V}^*$ be the set of materialized views. We say that the function $S : \mathcal{V}^* \to \mathbb{R}$ represents the resource used by views in $\mathcal{V}^*$.*

Proposition 12. *Let $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_m\}$ be a set of views (queries) to be answered, $E_{DAG} = (E, O)$ be their expression-DAG, $\mathcal{V}^*$ be a subset of (views) E, and $S(\mathcal{V}^*)$ be the resource utilization by $\mathcal{V}^*$. Then, the following optimization problem*

$$\begin{aligned} \min_{\mathcal{V}^*} \quad & \sum_{i \in \mathcal{V}} \beta_i C_A(\mathcal{V}_i, \mathcal{V}^*) + \sum_{j \in \mathcal{V}^*} \lambda_j C_M(\mathcal{V}_j^*, \mathcal{V}^*) \\ s.t. \quad & S(\mathcal{V}^*) \leq T \\ & \mathcal{V}^* \subseteq E \end{aligned} \tag{3.4}$$

is equivalent to (3.1)*, if $\Psi(v), v \in \mathbb{V}$ is equivalent to $S(\mathcal{V}^*) \leq T$ and $\mathcal{V}^* \subseteq E$.*

Proof. The proof follows directly from the fact that an evaluation plan of $\mathcal{V}$ (i.e., an element $v \in \mathbb{V}$) is a subset $\mathcal{V}^*$ of E. Then, if $\Psi(v), v \in \mathbb{V}$ is equivalent to $S(\mathcal{V}^*) \leq T$ and

$\mathcal{V}^* \subseteq E$, both search spaces are equivalent, and the objective function are equivalent too. This completes the proof. □

Optimization problem (3.4) is the same problem formulated in [Gup97, GHRU97, GM98], and it is clearly NP-hard problem, because (3.4) selects a subset of elements (views) $\mathcal{V}^*$ from the set E on E_{DAG}, and the number of subsets is exponential in the number of elements in E. However, [Gup97, GHRU97, GM98] provide near-optimal greedy heuristics for some special E_{DAG} cases. These heuristics perform the following three steps: (1) select a subset $\mathcal{V}^* \subseteq E$, (2) check if it is feasible (i.e., if it satisfies the constraints), and (3) evaluate the objective function and compare with previous solutions.

In general, these heuristics have been applied to some particular cases (AND and OR graphs), and they do not guarantee quality in their solutions. We extend formulation (3.4) taking advantage of the E_{DAG} structure, i.e., the interrelation between equivalence and operation nodes, as follows.

Proposition 13. *Let $E_{DAG} = (E, O)$ be an expression-DAG, and E^+_{DAG} be its transitive closure. Then, for every solution $\mathcal{V}^* \in E$ of* (3.4)*, there exists an expression-Path $\pi = (E_\pi, O_\pi)$ in E^+_{DAG}, such that $\mathcal{V}^* \equiv E_\pi$.*

Proof. Let $\mathcal{V}^*$ be a solution of (3.4), i.e., it is an evaluation plan. Then, as E^+_{DAG} has all possible of those plans, there exists an expression-Path with those selected views, i.e., $\pi = (\mathcal{V}^*, O_\pi)$. □

Therefore, selecting an optimal subset of views $\mathcal{V}^*$ using (3.4) is equivalent to find the cheapest expression-Paths in E^+_{DAG} that satisfies the resource constraint.

Definition 30. *Let $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_m\}$ be a set of views (queries) to be answered, and $E_{DAG} = (E, O)$ be the expression-DAG of $\mathcal{V}$, and $E^+_{DAG} = (E, O^+)$ its transitive closure. Then, we say that $\mathcal{V}_0$ is a* dummy *view of $\mathcal{V}$, if its expression-DAG $E'_{DAG} = (E', O')$ and transitive closure $E^{+'}_{DAG} = (E', O^{+\prime})$ are defined by $E' = E \cup \{\mathcal{V}_0\}$, $O' = O \cup \{O_0\}$, $O^{+\prime} = O^+ \cup \{O_0\}$, $C(\mathcal{V}_0) = \{O_0\}$, and $C(O_0) = \{\mathcal{V}_1, \ldots, \mathcal{V}_m\}$.*

A dummy view is a concept to create an expression-DAG with a unique root ($\mathcal{V}_0$). All the following results are based on this concept.

Definition 31. *Let $E_{DAG} = (E, O)$ be an expression-DAG rooted at e_0, $\{e_1, \ldots, e_{n_j}\}$ the set of all children of O_j, where $O_j \in C(e_0)$, and $E^k_{DAG} = (E_k, O_k)$ be a sub-expression-DAG rooted at e_k. Then, we say that $f : E \to \mathbb{R}$ is an* additive *function if there exists $f_i : E_i \to \mathbb{R}$, $1 \leq i \leq n_j$, a nondecreasing function $\mathcal{F}_j : \mathbb{R}^{n_j} \to \mathbb{R}$, for all $j \in C(e_0)$, and $\nu : O_j \to \mathbb{R}$, such that $f(e_0) = \nu(O_j) + \mathcal{F}_j(f_1(e_1), \ldots, f_n(e_{n_j}))$.*

Additive functions allow us to write functions recursively in terms of an expression-DAG structure. In particular, we are interested in rewriting optimization problem of (3.4).

Proposition 14. *Let $\mathcal{V}_0$ be a view with closure expression-DAG $E^+_{DAG} = (E, O^+)$ rooted at e_0, $L(E^+_{DAG})$ be the set of base relations of E^+_{DAG}, $\{e_1, \ldots, e_{n_j}\}$ the set of all children of O_j, where $O_j \in C(e_0)$, $E^{k+}_{DAG} = (E^+_k, O^+_k)$ be a sub-expression-DAG*

of E^+_{DAG} *rooted at* e_k*, and* $\pi = (E_\pi, O_\pi)$ *be a expression-Path on* E^+_{DAG}*, such that* $O_j \in O_\pi$*. Then,*

1. *Let* T *be the available resource,* $S(E_\pi)$ *be a resource utilization function, and* $\hat{S}(e_0) = T - S(E_\pi)$ *be the available resource at* e_0*. Then, if* $S(E_\pi)$ *is an additive resource utilization function, with* $r(O_j)$ *the resource utilization of operation node* O_j*, constraint* $S(E_\pi) \leq T$ *is equivalent to*

$$\hat{S}(e_0) = \min_k \left\{ \hat{S}(e_k) \right\} - r(O_j) \geq 0$$
$$\hat{S}(e_k) = T, e_k \in L(E^+_{DAG})$$

2. *Let* $\eta(e_0, \pi) = \beta_0 C_A(\mathcal{V}_0, E_\pi) + \sum_{e_i \in E_\pi} \lambda_i C_M(e_i, E_\pi)$ *the cost of expression-Path* π*. Then, if function* $C_A(\mathcal{V}_0, E_\pi)$ *and* $C_M(e_i, E_\pi)$ *are additive with* $C_A(\mathcal{V}_0, e_0) = \mu(O_j)$ *and* $C_M(e_0, E_\pi) = \psi(O_j)$*,*

$$\eta(e_0, \pi) = \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k, \pi_k)$$

Proof. (1) If operational node $O_j \in \pi$, and as $S(E_\pi)$ is an additive function, the resource utilization at e_0 is the maximum resource utilization at $\{e_1, \ldots, e_{n_j}\}$ plus the

resource utilization at node O_j (i.e., $r(O_j)$). Therefore,

$$\begin{aligned} S(E_\pi) \leq T &\Leftrightarrow r(O_j) + \max_k \{S(E_{\pi_k})\} \leq T \\ &\Leftrightarrow 0 \leq -r(O_j) + \min_k \{T - S(E_{\pi_k})\} \\ &\Leftrightarrow \min_k \left\{\hat{S}(e_k)\right\} - r(O_j) \geq 0 \end{aligned}$$

This completes this part of the proof.

(2) From the $\eta(e_0, \pi)$ definition, and since C_A and C_M are additive functions,

$$\begin{aligned} \eta(e_0, \pi) &= \beta_0 C_A(\mathcal{V}_0, E_\pi) + \sum_{e_i \in E_\pi} \lambda_i C_M(e_i, E_\pi) \\ &= \beta_0 C_A(\mathcal{V}_0, e_0) + \lambda_0 C_M(e_0, E_\pi) + \sum_{k \in C(O_j)} \{\eta(e_k, E_{\pi_k}) + \beta_0 C_A(\mathcal{V}_0, E_{\pi_k})\} \\ &= \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k, \pi_k) \end{aligned}$$

This completes the proof. □

Theorem 7. *Let $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_m\}$ be a set of views (queries) to be answered, $\mathcal{V}_0$ be its dummy view with expression-DAG transitive closure $E^+_{DAG} = (E, O^+)$, and $\eta(e_i)$ be the minimum cost at equivalence node e_i, i.e., $\eta(e_i) = \min_{\pi_k}\{\eta(e_i, \pi_k)\}$. Then, the*

following optimization problem is equivalent to (3.4)

$$\begin{aligned} \eta(e_i) = \min_j \left\{ \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k) \mid O_j \in C(e_i) \right\} \\ s..t. \quad e_i \in E^+ - L(E^+_{DAG}) \\ \hat{S}(e_i) \geq 0 \\ \eta(e_u) = 0,\ C_A(\mathcal{V}_0, e_u) = 0,\ S(e_u) = T,\ u \in L(E^+_{DAG}) \end{aligned} \tag{3.5}$$

If functions C_M and C_A are additive.

Proof. First, problem (3.1) can be expressed as follows, $\min_\pi \{\eta(\mathcal{V}_0, \pi) \mid S(E_\pi) \leq T\}$, i.e., finding the cheapest evaluation plan for $\mathcal{V}_0$, subject to the resource constraint. From Proposition 13 and using the fact that C_A and C_M are additive functions, then we can re-write (3.1) as follows:

$$\min_j \left\{ \mu(O_j) + \psi(O_j) + \sum_{e_k \in C(O_j)} \min_{\pi_k} \{\eta(\mathcal{V}_0, \pi_k) \mid \hat{S}(e_k) \geq 0\} \mid \hat{S}(\mathcal{V}_0) \geq 0 \right\}$$

Now, renaming $\min_j \left\{ \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k) \mid O_j \in C(e_i) \right\}$ by $\eta(e_i)$, we get (3.5). This complete the proof. □

Problem (3.5) finds the cheapest evaluation plan for $\mathcal{V}_0$, when there exist resource constraints. In the next section we present two algorithms to solve (3.5).

3.5 Solution and Algorithms

In this section we propose two algorithms to solve (3.5). We first consider that the solution is a complete expression-Path in an expression-DAG, and present a linear time algorithm (in terms of the expression-DAG size). Then, we consider the case where the solution is a subset of views, and it is not necessarily a complete expression-Path. We present a local search algorithm to solve (3.5), where we use a special case of the unconstrained shortest path algorithm as global search.

3.5.1 Shortest Path Algorithm in an Expression-DAG

The algorithm is a general one in the sense that it finds the shortest path in an expression-DAG. However, to guarantee that the algorithm solves (3.5), we need some additional conditions. The algorithm uses as input a general expression-DAG E_{DAG}, the available resource T, and a vector $\vec{y}$ with value $y_i = 1$ if view e_i can be materialized and $y_i = 0$ otherwise. The output is a set $\{P_i \mid i \in E\}$ indicating, for each equivalence node (view) i, which operation node has been selected . Therefore, selected views are found starting from $\mathcal{V}_0$ and recursively selecting views from P_i's. The algorithm assumes that E_{DAG} is ordered in inverse topological order, i.e., its nodes are enumerated, such that the following condition is satisfied:

$$(P(O_k) = \{i\}) \wedge (j \in C(O_k)) \Rightarrow j < i. \tag{3.6}$$

Since E_{DAG} is ordered in inverse topological order, the root $\mathcal{V}_0$ has the highest order. Finally, we consider that the set $BS(e_i)$ is the set of all operation nodes having node e_i as parent, i.e., $BS(e_i) = \{O_j \mid i \in P(O_j)\}$. The algorithm is as follows.

ViewSelection($E_{DAG} = (E, O)$, T,$\vec{y}$)

for each $i \in E$ **do**

$P_i = 0$

if ($i \in L(E_{DAG})$) **then**

$\eta(i) = 0, \hat{S}(i) = T$

else $\eta(i) = \infty$

for each $j \in O$ **do** $k_j = 0$

for i = 1 **to** $\mid E \mid$ -1 **do**

for each $O_j = (\{z\}, C(O_j)) \in BS(i)$ **do**

$k_j = k_j + 1$

if $k_j =\mid C(O_j) \mid$ **then**

$f = G(\{\eta(e_k), \vec{y} \mid e_k \in C(O_j)\})$

$\hat{S}^* = \min_k\{\hat{S}(e_k) \mid e_k \in C(O_j)\} - r(O_j)$

if $(\eta(z) > f \wedge \hat{S}^* \geq 0)$ **then**

$\eta(z) = f$

$P_z = O_j$

$\hat{S}(z) = \hat{S}^*$

Function $G(\{\eta(e_k), \vec{y} \mid e_k \in C(O_j)\})$ evaluates the cost from all children of operation node O_j. Note that vector $\vec{y}$ modifies this evaluation, allowing some equivalence nodes not to be considered. In general, when $\vec{y} = \vec{1}$, all nodes are considered to be materialized. The next theorem provides the correctness of our algorithm.

Theorem 8. *Let $\mathcal{V}_0$ be a view, and E^+_{DAG} be the transitive closure of $\mathcal{V}_0$ expression-DAG. Then, algorithm $ViewSelection(E^+_{DAG}, T, \vec{1})$ produces an optimal solution $\mathcal{V}^*$ to the optimization problem* (3.5).

Proof. Since E^+_{DAG} has all possible $\mathcal{V}_0$ evaluation plans, and $ViewSelection(E^+_{DAG}, T, \vec{1})$ finds the *shortest* (cheapest) path that satisfies the resource constraint, then, all views belonging to this path are the solution of (3.5), because they satisfy the constraint and produce the cheapest cost. This completes the proof. □

To analyze the complexity of the $ViewSelection()$ algorithm, we first note that each node and each edge is selected at most once, and for each operation node O_j we evaluate f using all nodes belonging to its children, i.e., $C(O_j)$, we consider this evaluation has a complexity $O(f)$. Therefore, the overall complexity is $O(O(f) \times size(E_{DAG}))$.

Although $ViewSelection(E^+_{DAG}, T, \vec{1})$ produces the optimal solution of (3.5), the complexity of E^+_{DAG} is still exponential in the number of views. However, if the solution of (3.5) is a complete expression-Path in E_{DAG}, then, $ViewSelection()$ algorithm provides the optimal solution. The next subsection presents a local search algorithm to solve (3.5).

3.5.2 Local Search Algorithm for Expression-DAG

Algorithm $ViewSelection(E^{+}_{DAG}, T, \vec{1})$ requires an exponential time to solve problem (3.5), because $size(E^{+}_{DAG})$ is exponential in terms of the number of views. We propose a local search algorithm [LH96, Glo89, Jea91] to overcome this drawback. The local search algorithm is structured as follows: *a number of local searches are performed, where for each one, the algorithm checks if the local optimum is better than the current objective function value.* This procedure is repeated until there is no acceptable neighborhood possible. Before describing the algorithm, we define the concept of materialization vector.

Definition 32. *Let $\mathcal{V}_0$ be a view, and $E_{DAG} = (E, O)$ be the expression-DAG of $\mathcal{V}_0$, where $E = (e_1, \ldots, e_n)$ and $O = (O_1, \ldots, O_m)$. We say that a vector $\vec{y} = (y_1, \ldots, y_n)$ is a* materialization vector *if*

$$y_i = \begin{cases} 1 & \textit{if view } e_i \in E \textit{ can be materialized,} \\ 0 & \textit{otherwise} \end{cases}$$

Materialization vectors represent a mechanism to define neighborhoods, i.e., initial solutions for each local search. The local search framework is described as follows.

Step 0. Assign 0 to k, $\vec{y}_k = (1, \ldots, 1)$, and $f^* = \infty$.

Step 1. Perform a local search, solving the following problem

$$\min_{\vec{x}} f(\vec{x}/\vec{y}_k)$$

$$s.t.\ \Psi(\vec{x}/\vec{y}_k)$$

Step 2. if $f(\vec{x}) < f^*$, then $f^* = f(\vec{x})$, and $\vec{y}^* = \vec{x}$.

Step 3. Increase k by 1, select a new $\vec{y}_k$, and go to step 2. If there is no $\vec{y}_k$ additional, go to step 4.

Step 4. Report objective function f^* and solution $\vec{y}^*$.

We have to explain how procedures local search (step 1) and selection of a new $\vec{y}_k$ are performed (step 4). First, we explain local search.

Local-Search We iteratively solve $ViewSelection(E_{DAG}, \infty, \vec{y})$ and put the solution in vector $\vec{x}$. Then, we evaluate if the constraint is satisfied. If it is not, then we select a *victim* and eliminate it. We solve $ViewSelection(E_{DAG}, \infty, \vec{y})$ again. The procedure is as follows.

Local-Search($E_{DAG} = (E, O), T, \vec{y}$)

$\vec{x} = ViewSelection(E_{DAG}, \infty, \vec{y})$

repeat until $(S(\mathcal{V}_0) \leq T)$

$EliminateMaterialized(\vec{y}, \vec{x})$

$\vec{x} = ViewSelection(E_{DAG}, \infty, \vec{y})$

Procedure $EliminateMaterialized(\vec{y}, \vec{x})$ selects a *victim*, i.e., a view k with the highest resource occupancy index. When the victim has been selected, we assign $y_k = 0$.

Neighborhood Selection Vector $\vec{y}$ is used to defined a neighborhood, because selecting some $y_i = 0$ we do not allow some solutions. We propose an aggressive strategy to move from neighborhood to neighborhood. This strategy eliminates a certain number N of views with the lowest resource occupancy index.

Although we do not offer a complexity analysis for this algorithm, the global complexity will depend of the number of internal iteration in $ViewSelection(E_{DAG}, \infty, \vec{y})$ ($\mid E \mid$ in the worse case), and the number of neighborhoods visited ($2^{|E|}$ in the worse case). However, there are many of those alternatives that are subsumed by others, and therefore, they do not need to be visited. A more exhaustive work is needed in this area to create an efficient mechanism to select neighborhoods.

3.6 Implementation and Experiments

This section presents the main results of our $ViewSelection(E_{DAG}, T, \vec{y})$ algorithm. Table 3.1 summarizes the results for 25 experiments, where $Size$ is the expression-DAG size, $\bar{E}$ is the average cardinality, $\sigma_{\bar{E}}$ is its standard deviation, and Time is the

running time measured in milliseconds. The algorithm was implemented using visual C++ 4.0, and it was run on a 120 Mhz PC compatible.

In general, expression-DAGs with a lower number of equivalence and operation nodes than those shown in Table 3.1, reported a time less than 10 milliseconds.

Table 3.1: Empirical Results

Size	$\bar{E}$	$\sigma_{\bar{E}}$	Time	Size	$\bar{E}$	$\sigma_{\bar{E}}$	Time
15818	3.7	3.0	170	6168	4.3	5.2	110
10089	5.2	5.7	110	9504	4.3	4.2	110
7716	2.6	1.9	50	16950	2.5	1.2	110
23759	2.9	2.1	280	4475	5.1	4.4	60
20128	2.4	0.8	160	39777	8.4	13.6	770
39921	4.1	3.5	550	19856	3.2	1.8	220
21881	2.8	1.3	280	27548	4.3	3.4	440
24890	2.3	0.8	270	25496	2.7	1.1	280
18226	3.1	1.6	170	30355	3.2	2.1	440
21188	2.6	1.4	270	33875	2.6	1.2	390
37047	2.5	1.2	490	39163	5.5	4.8	820
35241	2.4	1.1	440	27549	8.5	7.7	660

The experiments where generated randomly, where the minimum and maximum number of equivalence nodes were approximately 2,000 and 15,000 respectively, and operation nodes were 1,400 and 15,000 respectively.

Note that in this case (when all possible views are candidates to be materialized), function $G(\{\eta(e_k), \vec{y} \mid e_k \in C(O_j)\})$ corresponds to a simple summation of all $\eta(e_k)$, for all $e_k \in C(O_j)$, which reduces the complexity of our algorithm.

Figure 3.2 shows that the running time against the expression-DAG size. This time presents a positive linear behavior when the expression-DAG size increases. The

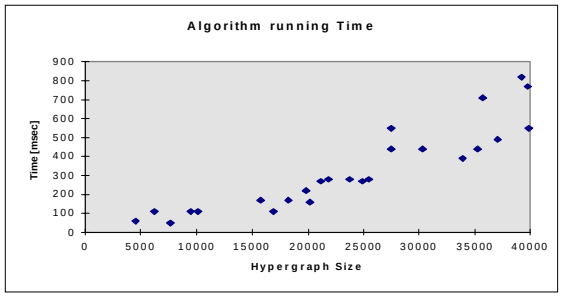

Figure 3.2: Experimental Run Time

variance produced in this relation is due to different expression-DAG structures, i.e., expression-DAG depth and average density (number of children at each node).

The main conclusion of these experiments, is that for a given view expression-DAG, the time to find the shortest (cheapest) path between the root and all base relations is a linear function of the expression-DAG size. Therefore, if the transitive closure E^+_{DAG} is available, the algorithm is a good alternative, and when E^+_{DAG} is not available, we can apply our local search algorithm in a reasonable searching time.

Chapter 4

OPTIMAL DECOMPOSITION IN QUASI-VIEWS

4.1 Introduction

Many applications must support the monitoring of distributed data for the occurrence of critical events or complex conditions among data items. Furthermore, much of that information is not necessarily required to be up-to-date, allowing some (controlled) degree of incoherence between users and information sources.

This chapter introduces and extends *quasi-views* as mechanism to materialize information in distributed environments with controlled incoherence. Quasi-views are views with explicit materialization conditions called *refresh conditions*. They were originally introduced in [Sel94, SK97] based on the quasi-copies concept [ABGM90]. We propose an optimization problem to materialize quasi-views optimally, based on results regarding constraints (Chapter 2) and view materialization (Chapter 3). We show that, in general, a quasi-view decomposition can not be considered as two independent problems (i.e., constraint decomposition and view materialization), and we provide

an iterative framework to solve this problem. Finally, we consider a special case of refresh conditions (disjunctive set of atomic linear arithmetic constraints), where the algorithms proposed in Chapter 2 to find a *safe* decomposition can be used.

This chapter is organized as follows. Section 4.2 presents the related work, Section 4.3 presents the contributions. Section 4.4 presents the problem characterization, where quasi-views are introduced, and the general materialization problem is formulated as an optimization one. We show, through an example, that the problem is not separable. Section 4.5 presents the solution strategy based on an iterative procedure, and we provide a solution algorithm for a special refresh condition case.

4.2 Related Work

Quasi-views were originally introduced in [Sel94, SK97] as an extension and generalization of quasi-caching [ABGM90]. This two concepts have been used as mechanisms to reduce the overhead cost incurred when derived data or multiple copies are maintained. quasi-views are regular views with formal materialization conditions, called refresh conditions. Quasi-caches contain quasi-copies, which are client-cached copies of database objects which are allowed to deviate in controlled ways from the primary copies. In particular, [Sel94, SK97] extend quasi-caching by: resolving active and passive data sources heterogeneity, new types of coherency (refresh) conditions, supporting transformation of cached data by view definition. However, both works have been limited

to refresh conditions over only one data source.

Since quasi-views and quasi-caching cache information from data sources (primary copies), they differ from the traditional cache concept [ASA+95, CI97, Vah97], in the sense that both quasi-views and quasi-caching maintain an *approximate* coherency between views (quasi-copies) and data sources (primary copies) rather than complete coherency as cache techniques do.

Finally, quasi-views (quasi-caching) are in somewhat related to materialized views [BLT86, Cer91, CW91, GL95, GMS93, LMSS95, GM95], in the sense that when a base relation is modified, views may have to be refreshed. However, materialized view techniques are more related to cache techniques, because views have to be refreshed when base relations do. Therefore, quasi-views are an extension of those techniques, allowing a controlled materialization (refreshing) policy through the refresh conditions.

4.3 Contributions

The problem addressed in this chapter is how a quasi-view may be evaluated efficiently in a distributed environment. In general, this problem is complex because it requires the efficient coordination of both the view materialization strategy (i.e., the optimal view materialization problem) and the refresh condition strategy. The contributions are as follows.

First, we show that the quasi-view decomposition problem is not a separable, that is,

it has to be considered as both decomposing materialized views and refresh conditions together. This formulation is completely new, since previous work addressed individual problems (constraint and view decompositions separately).

Second, a general solution strategy is proposed, which introduces the notion of a conditional problem, i.e., an optimization problem where some of its variables are fixed. Then, the quasi-view decomposition problem is reduced to the optimal view materialization and the constraint decomposition problems; a general solution framework, based on iterative conditional problem is proposed.

Third, for the special case of refresh conditions represented as disjunctive arithmetic linear constraints, we prove that the conditional refresh condition decomposition problem is equivalent to finding a (compact) safe decomposition, and therefore, all the results from Chapter 2 can be applied.

4.4 Problem Characterization

In this section we introduce and extend the *quasi-views* concept [Sel94] and we provide a formal quasi-view specification based on an extension of the relational model. Finally, we provide an optimization framework to decide the best quasi-view design for a given quasi-view definition, i.e., a mechanism to materialize and maintain a quasi-view.

4.4.1 Quasi-views

Intuitively a quasi-view is a view (see Chapter 4) with explicit re-evaluation conditions, called *refresh conditions*. A refresh condition is a concept that provides a controlled mechanism to maintain a view, i.e., it specifies when or under what conditions a view should be refreshed. In this subsection we extend [Sel94] characterizing a quasi-view over a distributed environment. We use the relational model presented in previous chapters.

Definition 33. *A* quasi-database *QDB is a pair (DB, RB), where DB is a regular database called* passive *database, and RB is a data set used on refresh conditions of views, called* refresh *database.*

Note that a quasi-database could have an empty refresh database RB, i.e., there is no variables defined for refresh conditions, and in this case the quasi-database is a regular one. Furthermore, in a quasi-database DB and RB are not necessarily disjoint, i.e., they can share some data.

Definition 34. *A quasi-view QV over a quasi-database QDB = (DB, RB) is a pair* (V, Ω)*, where V is a view over DB, and* Ω *is a refresh condition over RB. A quasi-view*

is expressed of the form:

DEFINE QUASI – VIEW QV **AS**

SELECT $\mathbb{B}$

FROM $T_1, T_2, \ldots, T_M$

WHERE $\mathbb{C}$

REFRESH WHEN Ω

where T_i's can be either a relational name, a view name, or a quasi-view name, Ω is the refresh condition, and $\mathbb{B}$ and $\mathbb{C}$ have the same interpretation as in Definition 19.

The semantics of a quasi-view QV over QDB is defined as a particular case of *event-condition-action* (ECA) rule [Cer91, CW91], and it is as follows: an event is any change in RB, the condition is Ω, and the action is re-materializing the view V. In other words, every time when RB is changed, check Ω, and if it is satisfied, then re-materialize the view.

Definition 35. *Let $QV = (V, \Omega)$ be a quasi-view, we say that the* refreshing frequency *of QV is the number of changes in RB that satisfy Ω, i.e., those changes that refresh QV. We will denote this frequency by r_{QV}.*

Note that the refreshing frequency r_{QV} will depend of how restrictive is Ω. Thus, if Ω is very restrictive, i.e., few values in RB satisfy Ω, then r_{QV} will be small, and vice versa. Therefore, defining Ω is the mechanism to define the refresh policy.

Now, we concentrate on the multi-database concept, and how the quasi-view concept can be re-defined over a multi-database.

Definition 36. *A multi-database system MDBS is* $(G_S, LS_1, \ldots, LS_n)$*, where* G_S *is a quasi-database called* global source*, and each* LS_i*, called* local sources*, is either a regular or a quasi-database.*

The global schema G_S is the union of all local schemes. Hence, the global DB is a collection of all relations in all sources, and the global RB is the collection of all RB_i's. We assume here for simplicity that the relation names used at different sources are unique. If this is not the case, an appropriate renaming can be done.

Definition 37. *A quasi-view over a multi database system MDBS is simply a quasi-view over the global schema MDBS.*

Note that this is a natural extension of the quasi-view concept. However, checking if Ω is satisfied or not for local RB changes becomes a complex problem, as we have shown in Chapter 2. In the next section we discuss how refresh conditions and quasi-view materialization problem are interrelated.

4.4.2 Motivation Example

Consider a quasi-view $QV = (V, \Omega)$ defined over two relations R_1 and R_2, where $R_1 = (ABC)$, $R_2 = (DEFG)$, and A, B, C, D, F, and G are attribute names. View V is defined by $\Pi_{B,C,G}(\sigma_{D \leq d \wedge B=D}(R_1, R_2))$, and the refresh conditions are two disjunctive

atomic linear arithmetic constraints define as follows: $F \geq f \vee A + F \geq g$. We assume that A and F are non-negative variables ranging over the Real numbers. Furthermore, we assume that both A and F values are distributed according to a probability distribution $h_A()$ and $h_F()$, respectively.

We consider that view V can be evaluated using the following two alternatives: a) direct evaluation, i.e., using original view V, or b) using $V_1 = \Pi_{B,C,G}(\sigma_{B=D}(R_1, V_2))$ and $V_2 = \Pi_{D,G}(\sigma_{D \leq d}(R_2))$. Figure 4.1 shows the expression-DAG to represent these alternatives.

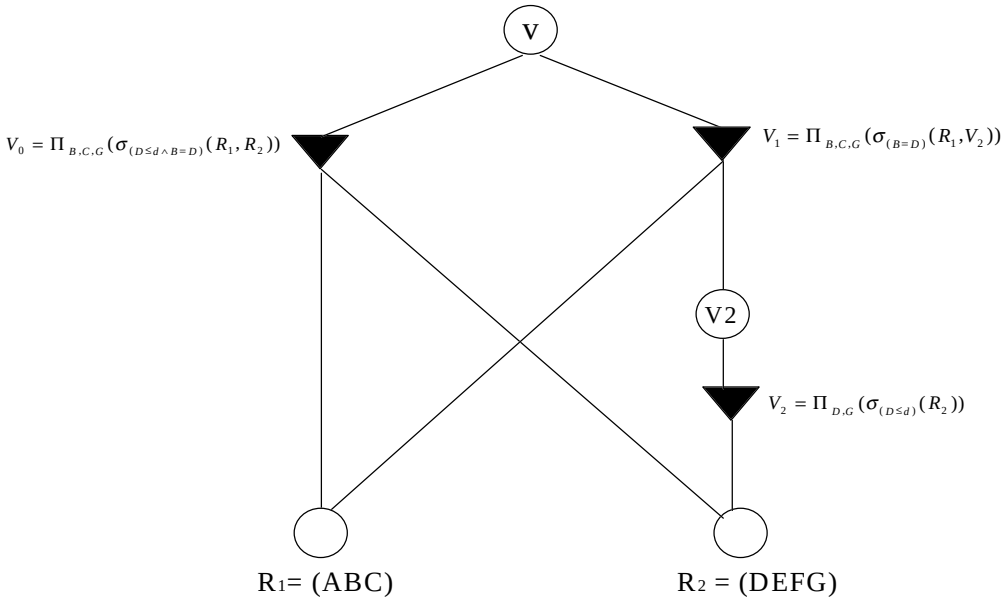

Figure 4.1: View V Expression-DAG

We will consider the following cost model: using direct evaluation the cost is $(r_A + r_F)C_1$, and using (V_1, V_2) the cost is $(r_A + r_F)C_2 + r_F C_3$, where C_1, C_2, and C_3 are

constants ($C_2 \leq C_1$), and r_A and r_F are the refresh frequencies from site 1 and 2 respectively.

We consider that the refresh conditions can be decomposed in two alternative decompositions: $s_1 = ((a_{11} \geq A \vee A \geq a_{12}), (f_{11} \geq F \vee F \geq f_{12}))$ with refresh frequencies (r_{A1}, r_{F1}) and $s_2 = ((a_{21} \geq A \vee A \geq a_{22}), (f_{21} \geq F \vee F \geq f_{22}))$ with refresh frequencies (r_{A2}, r_{F2}). The following figure shows these decompositions.

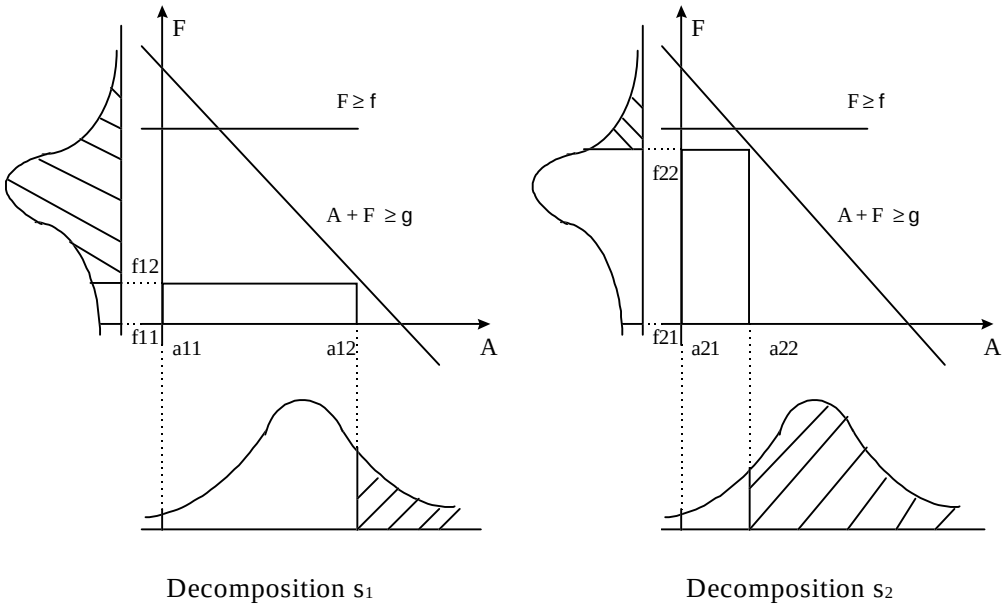

Figure 4.2: Refresh Conditions Alternatives

From Figure 4.2 we can conclude that locally $r_{A1} \leq r_{F1}$ and $r_{A2} \geq r_{F2}$, and $r_{A1} \leq r_{A2}$ and $r_{F1} \geq r_{F2}$, because the probability to violate local constraints in variable A of s_1 is greater than s_2, and vice versa for variable F. Therefore, the following scenarios are possible:

1. If s_1 is selected as decomposition of Ω. Then, site 2, variable F, will refresh more often than if s_2 is selected ($r_{F1} \geq r_{F2}$). Thus, we will select V rather than (V_1, V_2) if

$$r_{F1} \geq r_{A1} \times \frac{C_1 - C_2}{C_2 + C_3 - C_1}$$

   Otherwise, we select (V_1, V_2).

2. If s_2 is selected as decomposition of Ω. Then, site 1, variable F, will refresh less often than if s_1 is selected ($r_{F1} \geq r_{F2}$). Thus, we will select V rather than (V_1, V_2) if

$$r_{F2} \geq r_{A2} \times \frac{C_1 - C_2}{C_2 + C_3 - C_1}$$

   Otherwise, we select (V_1, V_2).

In particular, since $r_{A1} \leq r_{F1}$, we will probably select V if s_1 has been selected, and as $r_{A2} \geq r_{F2}$, we probably select (V_1, V_2) if s_2 has been selected. This clear dependency among views and refresh conditions problems indicate that the quasi-view design problem should consider both problems simultaneously (views and refresh conditions). In the next section we formulate formally the quasi-view problem.

4.4.3 Quasi-view Design Problem

In this section we define an optimization problem to decompose and evaluate a *quasi-view* defined over a multi-database.

Definition 38. *Let $QV = (V, \Omega)$ be a quasi-view defined over a multi-database, $\mathbb{V}$ be the set of all feasible equivalent view evaluation plans for V, and $\mathbb{S}$ be the set of all decompositions for Ω. We say that $f_{QV} : (\mathbb{V}, \mathbb{S}) \rightarrow \mathbb{R}$ is a real function that provides the selection criterion.*

The quasi-view design problem corresponds to selecting a set of views, i.e., an evaluation plan of V, and a Ω decomposition, such that the criterion (f_{QV}) is optimized. We will assume that the optimization is a minimization, and then the following optimization problem formulates the quasi-view design problem.

$$\begin{aligned} &\min_{v,s} \{f_{QV}(v, s)\} \\ &s.t.\ v \in \mathbb{V} \\ &s \in \mathbb{S} \end{aligned} \tag{4.1}$$

Note that optimization problem (4.1) requires decomposing views and refresh conditions simultaneously. Even though the search space $(\mathbb{V}, \mathbb{S})$ is completely orthogonal (i.e., independent), the selection criterion f_{QV} (objective function) depends of both variables, i.e., v and s, simultaneously.

This inseparability characteristic adds an additional complexity to the quasi-view design problem. Since, the view selection problem is a NP-hard one, the quasi-view is also a NP-hard problem.

4.5 Solution Strategy

Solving (4.1) entirely as an optimization problem may require a huge amount of computational effort, or simply may be impossible. In this section, we take advantage of structure of (4.1), i.e., its space search is composed of two independent search spaces ($\mathbb{V}$ and $\mathbb{S}$) and the objective function shares variables.

4.5.1 Framework

We propose an iterative framework based on Bender's decomposition [Geo72]. This framework is basically one of "learning from one's mistakes". It assigns a decomposition of Ω and solves the view problem, then this solution is used to find a new Ω decomposition. This process is repeated until both solutions converge. Before we describe this framework, we need to define the following concept.

Definition 39 (Conditional Problem). *Let P be an optimization problem in terms of variables x and y, and P_x be an optimization problem in terms of variable x, described*

as follows

$$P : \min_{x,y} f(x,y) \qquad \text{and} \qquad P_x : \min_{x} f(x,\bar{y})$$
$$s.t.\ x \in \mathbb{S}_x \qquad\qquad s.t.\ x \in \mathbb{S}_x$$
$$y \in \mathbb{S}_y$$

we say that P_x is the conditional optimization *problem of P with respect to x, if $\bar{y} \in \mathbb{S}_y$.*

Note that conditional problems have similarities with optimization problems formulated in previous chapters. In particular, the view conditional problem for a given refresh condition P_v, corresponds to the view materialization problem (Chapter 4). However, the refresh condition conditional optimization problem for a given view decomposition P_s, is not the same optimization problem formulated in Chapter 2, because our refresh conditions have different semantics. This particular problem is addressed in Section 4.5.2, and we prove that for a special case of refresh conditions, the optimization problem can be reduced to the constraint decomposition problem (2.1).

Conditional problems can reduce the complexity of (4.1). However, solving each conditional problem independently may produce suboptimal solutions, and therefore, the solution quality may be compromised. To overcome to this problem we use an iterative solving strategy based on Bender's decomposition [Geo72].

Quasi View Framework

Step 0 Let k = 0, assign an initial solution of P_v to v^k.

Step 1 Solve P_s with v^k, and assign its solution to s^k.

Step 2 Solve P_v with s^k, and assign its solution to v^{k+1}.

Step 3 If $v^k = v^{k+1}$ stop. Otherwise, $k = k + 1$ and go to 1.

The complexity of this framework is based on the complexity of each conditional optimization problem, i.e., refresh decomposition and view selection problems, multiplied by the number of iterations necessary to reach the solution. In the next section we address to special case for the refresh condition decomposition problem, and we prove that it is a special case of the constraint decomposition problem presented in Chapter 2.

4.5.2 Refresh Condition Decomposition

In this subsection we discuss the refresh condition decomposition problem, i.e., our conditional optimization problem P_s. In the general case, we provide a formulation in terms of *cover* decompositions (see Chapter 2), and for the special case of *disjunctive* linear arithmetic constraints we are able to provide an effective algorithm.

Since a refresh condition is a constraint that can be evaluated as true or false in light of quasi-database changes, we would like to derive a test, such that we can locally decide if a quasi-database change will not refresh our quasi-view. The following

theorem provides this test in terms of the cover decomposition concept.

Theorem 9. *Let $QV = (V, \Omega)$ be a quasi-view defined over a quasi-database $QDB = (DB, RB)$, with RB a partitioned data set, and $(G_1, \ldots, G_M)$ be a cover decomposition of Ω. Then, QV will not be refreshed if all $G_1, \ldots, G_M$ are not satisfied.*

Proof. The proof follows directly from the cover definition (see Chapter 2), because if $(G_1, \ldots, G_M)$ is a cover decomposition of Ω then $\Omega \models G_1 \vee \ldots \vee G_M$. However, this is equivalent to $\neg G_1 \wedge \ldots \wedge \neg G_M \models \neg\Omega$, which is our local test. This completes the proof. □

Therefore, our conditional optimization problem for refresh condition decomposition P_s corresponds to finding a cover decomposition of Ω, such that a criterion is optimized. However, in the general case, like safe decompositions, cover decompositions are hard to calculate effectively, but for a special case (a set of disjunctive atomic linear arithmetic constraints), and under certain (local) uniformity assumptions, cover decompositions can be effectively calculated. The following proposition formalizes this concept.

Proposition 15. *Let $QV = (V, \Omega)$ be a quasi-view defined over a quasi-database $QDB = (DB, RB)$, with RB a partitioned data set, and Ω be a set of disjunctive linear arithmetic constraints. Then, a cover decomposition $(G_1, \ldots, G_M)$ of Ω is equivalent to a safe decomposition $(\neg G_1, \ldots, \neg G_M)$ of $\neg\Omega$.*

Proof. The proof follows directly from Proposition 1 in Chapter 2. □

Therefore, in this case, all the algorithmic machinery developed in Chapter 2 can be used to find a cover decomposition of Ω. Note that we first find a safe decomposition $(C_1, \ldots, C_M)$ of $\neg\Omega$ (this is a set of conjunctive linear arithmetic constraints), and then we get the cover decomposition as $(\neg C_1, \ldots, \neg C_M)$.

Chapter 5

CONCLUSIONS

This dissertation has addressed to three database management problems: distributed integrity constraint management, optimal materialized views, and optimal quasi-view decomposition. Those are interrelated, but their solution can be addressed individually. In the following, we present the contributions and conclusions, and some important future research areas.

Chapter 2 addresses the problem of deriving the best possible decompositions, both during design and at update time, more specifically, the contributions are as follows. First, we introduce a generic optimization framework for achieving best decompositions: the search space is the set of all *feasible safe* decompositions $(C_1, \ldots, C_M)$ of the global constraint Ω over M distributed sites, i.e., $C_1 \wedge \ldots \wedge C_M$ imply Ω. The objective function can describe a variety of optimization criteria, such as the probability of an update satisfying local constraints, the expected number of updates before an update violates local constraints, or the average of the expected overall cost of manipulations during an update. Feasible decompositions are characterized by decompositions having the first

and possibly other properties from the following list): (1) safety, (2) local consistency, (3) partial constraint preservation, and (4) resource partition. One or more properties 1 through4 are required for various decomposition scenarios, depending on what is known at the time of a decomposition.

Second, for the case of general linear arithmetic constraints, we reduce the optimization -based framework to a standard, finitely-specified problem of mathematical programming. To do that, we introduced the notion of *compact split* (safe) decompositions, and prove that for any monotonic objective function the optimal safe decomposition can always be found in the subspace of compact split decompositions. Then, we prove existence and actually developing a finite parametric (i.e., in terms of coefficients) characterization of the properties 1 through4 of feasible decompositions together with optimization criteria.

Third, we develop an algorithmic framework to solve the resulting optimization problems. The constructed optimization problems, formulated in terms of parametric descriptions, have linear constraints and a non-linear objective function, based on a parametric representation of the volume function for the constraint space. For the design-time case, where each local constraint is in a single variable, the constructed objective function is concave; this property enables us to use a global search algorithm. We adopt the Frank-Wolfe algorithm to solve it. For other cases, the objective function is not concave and we use local search techniques in the algorithmic framework, that incorporate the Frank-Wolfe algorithm for search in local neighborhoods. To run

experiments and to show the feasibility of the approach, we have implemented an optimization engine for the schema-based full decompositions with local constraints in single variables. The experiments suggest that the approach is feasible and scalable, but more experimental study will be necessary to fine-tune the algorithms for specific cases.

Chapter 2 also proposes a general framework to manage global linear arithmetic constraints in distributed databases, extending [BGM92]. This framework can be applied to different network and system architectures (centralized, hierarchical, and fully distributed). The framework considers two types of sites: coordinators and non-coordinators. A coordinator is a site that coordinates the constraint decomposition to facilitate a local update at a non-coordinator site k that violates the current local constraint. This is done by finding a set of sites θ, containing site k, and trying to create a new (partial) compact split that would satisfy the new database state (i.e., new update). We propose a primitive RESOURCE-TRANSFER (a distributed transaction involving two sites) to deal with inter-site communication. We proved that under standard transaction management properties, any protocol that uses exclusively this primitive guarantees local and global consistency (i.e., current $C_1, \ldots, C_M$, and Ω), it is resilient to failures, and produces an optimal decomposition for sites in θ. Finally, we exemplify an instance of our distributed framework for the single coordinator case.

Chapter 3 introduces a generic optimization framework to decide optimally materialized views. First, we extend the expression-DAG (Direct Acyclic Graph) [RSS96] as

a mechanism to represent equivalent view evaluation plans. We characterize paths and the transitive closure of an expression-DAG (i.e., an expression-DAG with all possible equivalent view evaluation plans). We show that, under certain conditions, expression-DAG and AND-OR graphs are equivalent. However, while the size of a standard AND-OR graph is defined in terms of its nodes and arcs, the size of an expression-DAG is defined in terms of the cardinality of its operational nodes. Second, the optimization framework is formulated in terms of the expression-DAG structure. Thus, under certain objective function conditions, the problem of optimal selection of materialized views can be formulated as the constrained shortest path in an expression-DAG, i.e., a complete expression-path or AND-path. For this case, a linear-time algorithm (in terms of the expression-DAG size) is presented. Note that this special case can handle important applications such as inventory control and logistics support. We have performed some experiments, and the results suggest that our approach is feasible. Third, for the general optimization problem and if the expression-DAG has all possible view evaluation plans, then the linear-time algorithm can be applied to obtain a solution. Note that this algorithm does not evaluate all possible equivalent evaluation plans, because all those which are subsumed by others are eliminated earlier. Finally, if the expression-DAG with all possible view evaluation plans is not available, a local search algorithm is presented. However, further research is necessary in this area.

In Chapter 4, we formalize and extend considerably the notion of quasi-view (a view with explicit re-materialization conditions, called *refresh conditions*) to multi-databases

and create an optimization framework to design them. First, we show that the optimal quasi-view decomposition problem is not a separable problem, i.e., it has to be considered as both view decomposition and refresh condition evaluation together. Second, a general solution strategy is proposed, which introduces the notion of a conditional problem (i.e., optimization problems where some of its variables are fixed), where the optimization problem is reduced to the optimal view materialization and the constraint decomposition problems. Third, for the special case of disjunctive refresh conditions, we prove that the conditional refresh conditions decomposition problem is equivalent to finding a compact split (safe) decomposition, and therefore, all results from Chapter 2 can be applied.

Finally, this dissertation has presented an integrated framework for the analysis, design, and optimal decomposition of global database constraints, views, and quasi-views. This approach is a contribution to the state-of-the-art in constraint, view, and quasi-view management, in that the general decomposition problems can be formulated as optimization problems.

Future Work

This dissertation provides a framework to decompose optimally constraints, views and quasi-views, and it has been shown that this approach is formal, rigorous and feasible. However, there are many possible extensions to this work.

In the area of constraint decomposition, the most natural is to extend the results

to more generic numeric constraints, and to provide effective optimization algorithms. This is an important topic, since arithmetic linear constraints might be very restrictive for some domains. A second direction is to explore different objective functions, in which other criteria may be used. A third direction is to consider replicated data, i.e., relaxing the assumption that the variables in the constraint must belong to a partition; this is of special interest in practical problems.

The framework to manage constraints has based its results for local transactions (especially local updates). However, when a distributed transaction is processed the inter-site coordination problem must be solved. Another area is to implement effectively other architecture configurations, especially distributed ones. These instantiations could be used to solve problems related to multiple resource requirements.

In terms of materialized views (Chapter 3), the local search algorithm needs to be explicitly implemented and studied in terms of its complexity. A very important issue is related to the transitive closure of an expression-DAG, and whether it can be derived efficiently from an expression-DAG, the constrained shortest path problem represents a good alternative. Finally, research is needed to identify the classes of problems to which our approach can be applied efficiently, for example conjunctive views.

Finally, our quasi-view work can be extended directly to provide a more sophisticate semantics, i.e., extending the action part of the ECA paradigm from simply refreshing a quasi-view, to executing a set of rules. Also, this work can be extended to different types of refresh conditions, for instance, conjunctive ones.

Bibliography

Bibliography

[ABGM90] R. Alonso, D. Barbará, and H. Garcia-Molina. Data caching issues in an information retrieval. *ACM Transactions on Databases*, 15(3):359–384, 1990.

[ASA+95] M. Abrams, C. Standbridge, G. Abdulla, S. Williams, and E. Fox. Caching proxies: Limitations and potentials. In *WWW-4, Boston Conference*, 1995.

[Bea96] B. Bueler and et al. Exact volume computation for polytopes: A practical study. Technical report, IFOR, Switzerland, 1996.

[BGM92] D. Barbará and H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Arithmetic Constraints in Distributed Database Systems. In *Proc. of the 3rd International Conference on Extending Data Base Technology, EDBT'92*, pages 373–388. Springer-Verlag, 1992.

[BLCea94] T. Berners-Lee, R. Cailliau, and et al. The world-wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[BLT86] J. Blakeley, P. Larson, and F. Tompa. Efficiently updating materialized views. In *Proceeding of the ACM SIGMOD Conference on Management of Data*. ACM Press, 1986.

[BPT97] E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multidimensional database. In *Proceeding of the International Conference on Very Large Data Bases*, 1997.

[BS79] M. Bazara and C. Shethy. *Nonlinear Programming, Theory and Algorithms*. John Wiley and Sons, 1979.

[Cer91] S. Ceri. Deriving production rules for incremental view maintenance. In *Very Large Data Bases*, 1991.

[CH79] J. Cohen and T. Hickey. Two algorithms for determining volumes of convex polyhedra. *Journal of the ACM*, 26(3):401–414, 1979.

[CI97] Pei Cao and Sandyn Irani. Cost-aware www proxy caching algorithms. 1997.

[CW91] S. Ceri and J. Widom. Production rules form incremental view maintenance. In *Proceeding of the International Conference on Very Large Databases, Barcelona, Spain*, 1991.

[CZ96] E. Chong and S. Zak. *An Introduction to Optimization.* John Wiley and Sons, 1996.

[Geo72] A. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[GHRU97] G. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index selection in olap. In *Proceeding of the International Conference on Data Engineering*, 1997.

[GL95] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proceeding of the ACM SIGMOD Conference on Management of Data.* ACM Press, 1995.

[Glo89] F. Glover. Tabu search, part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.

[GM91] H. Garcia-Molina. Global consistency constraints considered harmful. In *Proc. First International Workshop on Interoperability in Multidatabase Systems (IMS 91)*, pages 248–250, 1991.

[GM95] H. Gupta and I. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):3–18, 1995.

[GM98] H. Gupta and I. Mumick. Selection of views to materialize under a maintenence cost constraint. Technical report, Computer Science Department, Stanford University, 1998.

[GMS93] A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In *Proceeding of the ACM SIGMOD Conference on Management of Data.* ACM Press, 1993.

[GSE+97] S. Grufman, F. Samson, S. Embury, P. Gray, and T. Risch. Distributing semantic constraints between heterogeneous databases. In *13th International Conference on Data Engineering (ICDE'97), (IEEE), Birmingham, England*, 1997.

[Gup97] H. Gupta. Selection of views to materialize in a data warehouse. In *Proceeding of the International Conference on Data Base Theory*, 1997.

[GW93] A. Gupta and J. Widom. Local verification of global integrity constraints in distributed databases. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 49–58, Washington, D.C., 1993. ACM.

[HJLL] T. Huynh, L. Joskowicz, C. Lassez, and J-L. Lassez. Practical tools for reasoning about linear constraints. Technical report, IBM T.J. Watson Research Center.

[HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *Proceeding of the ACM SIGMOD Conference on Management of Data*. ACM Press, 1996.

[Huy97] N. Huyn. Maintaining global integrity constraints in distributed databases. *Constraint: An Internation Journal*, 2(3–4):377–399, 1997.

[Jea91] D. Johnson and et al. Optimization by simulating annealing: An experimental evaluation; PART II, graph coloring and number partitioning. *Operation Research*, 39(3):378–406, 1991.

[JK97] S. Jajodia and L. Kerschberg. *Advanced Transaction Models and Architectures*. Norwall, MA, Kluwer Academic Publishers, first edition, 1997.

[Kam84] N. Kambo. *Mathematical Programming Techniques*. Affiliated East-West Press PVT Ltd., 1984.

[KGea96] L. Kerschberg, H. Gomaa, and et al. Data and information architectures for large-scale distributed data intensive information systems. In *Proc. of the Eighth IEEE International Conference on Scientific and Statistical Database Management, Stockholm, Sweden*. IEEE Computer Society Press., 1996.

[Las] J-L. Lassez. From LP to LP: Programming with constraints. Technical report, IBM T.J. Watson Research Center.

[Las83] J.B. Lasserre. An analytical expression and algorithm for the volume of a convex polyhedron in Rn. *Journal of Optimization Theory and Applications*, 39(3):363–377, 1983.

[LH96] M. Laurent and P. Van Hentenryck. Localizer: A modeling language for local search. 1996.

[LM92] Jean-Louis Lassez and Michael Maher. On Fourier's algorithm for linear arithmetic constraints. *Journal of Automated Reasoning*, 9:373–379, 1992.

[LMSS95] J. Lu, G. Moerkotte, J. Schue, and V. Subrahmanian. Efficient maintenance of materialized mediated views. In *Proceeding of the ACM SIGMOD Conference on Management of Data*. ACM Press, 1995.

[Maz93] S. Mazumdar. Optimizing distributed integrity constraints. In *Proc. Third International Symposium on Database Systems for Advanced Applications (DASFAA-93)*, pages 327–334, Taejon, Korea, 1993.

[MY98] S. Mazumdar and Z. Yuan. Localizing global constraints: A geometric approach. In *In Proceedings of the 9th International Conference on Computing and Information. ICCI'98*, 1998.

[NKOD95] S. Nural, P. Koksal, F. Ozcan, and A. Dogac. Query decomposition and processing in multidatabase systems. Technical report, 1995.

[Qia89] X. Qian. Distributed desing of integrity constraints. In L. Kerschberg, editor, *Proc. Second International Conference on Expert System Database Systems*, pages 417–425, Redwood City, California, 1989. Benjamin Cummings.

[QS87] X. Qian and D. Smith. Constraint reformulation for efficient validation. In *Proc. Thirteenth International Conference on Very Large Databases*, pages 622–632, 1987.

[RSS96] K. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *SIGMOD, Montreal, Canada*. ACM Press., 1996.

[Sel94] L. Seligman. *A Mediator for Approximate Consistency: Supporting 'Good Enough' Materialized Views.* PhD thesis, School of Information Technology and Engineering, George Mason University, Fairfax, VA., 1994.

[SK95] L. Seligman and L. Kerschberg. Federated knowledge and database systems: A new architecture for integrating of AI and database systems. *Advances in Databases and Artificial Intelligence*, 1, 1995.

[SK97] L. Seligman and L. Kerschberg. A mediator for approximate consistency: Supporting 'good enough' materialized views. *Journal of Intelligent Information Systems*, 8(3):203–225, 1997.

[SS90] N. Soparkar and A. Silberschatz. Data-value partitioning and virtual messages. In ACM, editor, *Proc. 9th ACM SIGACT-SIGMOD-SIGART Simposium on Principles of Database Systems*, Nashville, Tennessee, 1990.

[SV86] E. Simon and P. Valduriez. Design and implementation of an extendible integrity subsystem. In *Proc. Nineteenth Hawaii International Conference on System Sciences*, pages 622–632, 1986.

[TS97] D. Theodoratos and T. Sellis. Data warehose configuration. In *Proceeding of the International Conference on Very Large Data Bases*, 1997.

[Ull88] J. Ullman. *Principles of Database and Knowledge-base Systems: The New Technologies.* Computer Science Press, 1988.

[Vah97] Amin Vahdat. Transparent result caching. 1997.

[YKL97] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceeding of the International Conference on Very Large Data Bases*, 1997.

Curriculum Vitae

Samuel E. Varas was born on June 25, 1964, in San Vicente de T.T., Chile. He graduated in Industrial Engineering from the University of Chile in 1989, and he received his Master of Science in Industrial Engineering (minor in Economy) from the University of Chile in 1989. He is a full time academic at University of Chile since 1989.

Permanent Address:	República 701 Santiago Chile
Electronic Address:	svaras@dii.uchile.cl http://www.dii.uchile.cl/~svaras/

This dissertation was typeset with LaTeX by the author. LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.