

ENABLING MULTI-PARTY COLLABORATIVE DATA ACCESS

A Dissertation
Submitted to
the Temple University Graduate Board

in Partial Fulfillment
of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY

by
Malek Athamnah
December, 2018

Examining Committee Members:

Dr. Krishna Kant, Advisory Chair, Dept. of Computer & Information
Sciences

Dr. Justin Shi, Dept. of Computer & Information Sciences

Dr. Chiu Tan, Dept. of Computer & Information Sciences

Dr. Saroj Biswas, Dept. of Electrical & Computer Engineering

©
Copyright
2018

by

Malek Athamnah

All Rights Reserved

ABSTRACT

Enabling Multi-party Collaborative Data Access

by

Malek Athamnah

Cloud computing has brought availability of services at unprecedented scales but data accessibility considerations become more complex due to involvement of multiple parties in providing the infrastructure. In this thesis, we discuss the problem of enabling cooperative data access in a multi-cloud environment where the data is owned and managed by multiple enterprises. We consider a multi-party collaboration scheme whereby a set of parties collectively decide accessibility to data from individual parties using different data models such as relational databases, and graph databases. In order to implement desired business services, parties need to share a selected portion of information with one another. We consider a model with a set of authorization rules over the joins of basic relations, and such rules are defined by these cooperating parties. The accessible information is constrained by these rules.

Specifically, the following critical issues were examined: Combine rule enforcement and query planning and devise an algorithm which simultaneously checks for the enforceability of each rule and generation of minimum cost plan of its execution using a cost metric whenever the enforcement is possible; We also consider other forms of limiting the access to the shared data using safety properties and selection conditions. We proposed algorithms for both forms to remove any conflicts or violations between the limited accesses and model queries; Used graph databases with our authorization rules and query planning model to conduct similarity search between tuples, where we represent the relational database tuples as a graph with weighted edges, which enables queries involving "similarity" across the tuples. We proposed an algorithm to exploit the correlations between attributes to create virtual attributes that can be used to

catch much of the data variance, and enhance the speed at which similarity search occurs; Proposed a framework for defining test functionalities their composition, and their access control. We discussed an algorithm to determine the realization of the given test via valid compositions of individual functionalities in a way to minimize the number of parties involved.

The research significance resides in solving real-world issues that arise in using cloud services for enterprises After extensive evaluations, results revealed: collaborative data access model improves the security during cooperative data processes; systematic and efficient solving access rules conflict issues minimizes the possible data leakage; and, a systematic approach tackling control failure diagnosis helps reducing troubleshooting times and all that improve availability and resiliency. The study contributes to the knowledge, literature, and practice. This research opens up the space for further studies in various aspects of secure data cooperation in large-scale cyber and cyber-physical infrastructures.

ACKNOWLEDGEMENTS

During my journey in this thesis, I feel that I'm indebted to many people for their help along the way. First and before mentioning those people, I'm indebted to Allah who started it all, without his will, none of this would be possible, his response to my prayers grant me with a lot of confidence and hope to continue my way. I would like to express my deepest gratitude and thanks to my supervisor, Dr. Krishna Kant. My journey with Dr. Krishna started four years ago, during these years he was a great supervisor, his style of dealing with others, encouragement, support, and enthusiasm can't be expressed in few words. I will never forget his patience, and insistence to make impossible things happen. He has always a major source for feedback, comments, and road map planning for our journey in this thesis. Many thanks for my family. Their love, care, patience, sacrifices, and confidence were the map of the road which I followed. I would like also to thank my friends, colleagues, and all who have contributed directly or indirectly to accomplish this work.

DEDICATION

To my late father, who unfortunately passed away during this journey toward my degree and didn't stay in this world long enough to see his son become a doctor. May God rest your soul in peace.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	v
DEDICATION	vi
LIST OF FIGURES	x
LIST OF TABLES	xi
CHAPTER	
1. INTRODUCTION	1
1.1 Cooperative data access	1
1.1.1 The Goal	2
1.1.2 Motivation	2
1.1.3 Contributions	2
1.2 Thesis Outline	5
2. MINIMUM COST RULE ENFORCEMENT FOR COOPERATIVE DATABASE ACCESS	
2.1 Introduction	6
2.2 Related Work	10
2.3 Authorization Model	12
2.3.1 A Running Example	12
2.3.2 Assumptions and Definitions	13
2.3.3 Authorized Queries Across Parties	18
2.3.4 Inadequacy of Classical Query Planning	21
2.3.5 Complexity of Cooperative Query Planning	23
2.4 Combined Rule Enforcement and Query Planning	26
2.4.1 Enforcement Cost Considerations	27
2.4.2 Consistent query planning	28
2.4.3 Algorithm Performance	32
2.5 Query Planning with a Third Parties	36
2.5.1 Minimum Cost Enforcement Through Join Service	37
2.5.2 Enforcement with Multiple Third Parties	42
2.6 Conclusion	45

3. GENERALIZED INTER-CLOUD STRUCTURED DATA SHARING	47
3.1 Introduction	47
3.2 Multiparty Model and Related Work	48
3.2.1 Fundamental Issues in Multiparty Models	49
3.2.2 Related Work	50
3.3 Extended Collaboration Model	51
3.3.1 Model Queries and Safety Properties	51
3.3.2 Enabling Conditional Accessibility	53
3.3.3 Running Example	53
3.4 Weakening of Model Queries	54
3.4.1 Weakening with and without Selections	56
3.4.2 Combined Safety Properties and Selections	59
3.5 Rule Enforcement under Conditional Access	60
3.6 Performance Evaluation	66
3.6.1 Evaluation of Access Rule Generation Algorithm (<i>ARG</i>)	66
3.6.2 Evaluation of Enforcement with Selections	68
3.6.3 Evaluation in a Real Cloud Environment	70
3.7 Conclusion	70
4. COLLABORATIVE SIMILARITY SEARCH ACROSS MULTI-PARTY REPOSITORIES	72
4.1 Introduction	72
4.2 Background	75
4.2.1 Multiparty Data Sharing Model	75
4.2.2 Representing Tuple Relationships	77
4.2.3 Collaborative Databases with Similarity Queries	79
4.3 Related Work	82
4.4 Similarity Search Algorithm	84
4.4.1 Similarity Search	84
4.4.2 Enhanced Algorithm	85
4.4.3 Exploiting Dominating Attributes	88
4.5 Performance Evaluation	91
4.5.1 Enhanced vs. Simple Scan	91
4.5.2 Enhanced Scan with PCA	94
4.6 Conclusion	95
5. A FRAMEWORK FOR MISCONFIGURATION DIAGNOSIS IN INTERCONNECTED MULTIPARTY SYSTEMS	97
5.1 Introduction	97

5.2	An Example of Multiparty Service	100
5.3	Diagnosis in Multiparty Systems	102
	5.3.1 Generating Tests from Probes	103
	5.3.2 Defining Cross-Party Access Rights	104
	5.3.3 Testing Issues	106
5.4	Problem description and Solution Method	107
	5.4.1 Preliminaries and notations	107
	5.4.2 Problem Definition and Complexity of MPT	109
	5.4.3 Proposed solution	111
5.5	Performance evaluation	113
	5.5.1 Effect of total number of probes	114
	5.5.2 Effect of total number of parties	115
	5.5.3 Effect of K	116
5.6	Related Work	117
5.7	Conclusion	118
 6. CONCLUSION AND FUTURE WORK		120
	6.1 Conclusion	120
	6.2 Future work	121
 BIBLIOGRAPHY		123

LIST OF FIGURES

Figure

2.1	Illustration of Query Planning	22
2.2	Illustration of Enforcement Complexity	25
2.3	Relevance graph for Local Enforcement	29
2.4	Illustration New vs. Old Algorithm (1)	35
2.5	Illustration New vs. Old Algorithm (2)	35
2.6	Numer of candidate rules	41
2.7	Time comparison between two algorithms	41
2.8	Minimal communication costs found by two algorithms	41
4.1	An Illustration of Query Type in Supply Chain	80
4.2	An Illustration of Query Type in e-commerce	80
4.3	Performance of Enhanced-Scan vs. Simple-Scan	91
4.4	PCA for partition	94
5.1	Typical Email Flow	100
5.2	Email Configurations	101
5.3	Combination of probes in (a) series, and (b) parallel.	103
5.4	Illustration of Access Possibilities	105
5.5	Controller graph	109
5.6	Illustration of the MPT problem.	111
5.7	Illustration of the (a) serial and (b) parallel test graphs.	113
5.8	Comparison of the minimum number of parties with different number of probes in G for (a) serial and (b) parallel test graphs.	114
5.9	Comparison of the minimum number of parties with different number of parties in G for (a) serial and (b) parallel test graphs.	115
5.10	Variation of number of parties required for conducting the tests with different K	116

LIST OF TABLES

Table

2.1	Auth. rules for running Example	14
2.2	Rule Enforcement and Planning Results	34
2.3	Auth. rules for running Example	43
3.1	Model Queries for the Running Example (<i>i</i> indicates that both parties 1&2 have the rule)	55
3.2	Rule Enforcement and Safety Properties	67
3.3	Access rules with new party	67
3.4	Auth. rules with selection condition on three attributes	69
3.5	Rule enforcement with selection	69
3.6	Rule enforcement time (in secs) in Amazon Cloud	70
4.1	Graph Similarity on one relation	90
4.2	Rules for the Running Example	91
4.3	Similarity queries	91
4.4	Similarity queries results	95

CHAPTER 1

INTRODUCTION

1.1 Cooperative data access

Advanced technologies offer new opportunities to the industry. Cloud computing is one of these technologies. It provides computing power and storage resources in distributed environment, while abstracting underlying infrastructure, depending on a given service. Several enterprise level organizations deploy highly scalable cloud computing solutions for an internal data sharing and collaboration. Moreover, some companies offer their cloud services for commercial use and act as a Cloud Service Providers (CSP) in the market. Enterprises increasingly need to collaborate to provide rich services to clients with minimal manual intervention and without paper documents. However, from data security point of view, sharing mechanisms releases more information than needed, and it is not desired by the data owners in many cases. Usually, data owners release their data to other parties only based on their collaboration requirements. In such environments, data must be released only in a controlled way among cooperative parties, subject to the authorization policies established by them. The data authorization should be flexible and fine-grained so that the data owners can easily determine which portion of the information is released to which party. On the other hand, the rest of the information that is not released needs to be protected.

1.1.1 The Goal

The goal of this thesis is to examine the problem of cooperative query execution in a multi-cloud environment where the data is owned and managed by multiple enterprises. Each enterprise maintains its own data using cloud services. In order to implement desired business services, parties need to share selected portions of information with one another. The accessible information is constrained by rules. The authorization rules are envisioned based on business needs and agreements may suffer from several issues.

1.1.2 Motivation

Data accessibility is considered a backbone for enterprises when they need to provide business services through the cloud, over the last few years, cloud computing has become more and more popular. It brings revolutionary innovation with regards to cost, resource management and utilization. Cloud computing offers nearly unlimited resources, highly reliable on-demand services with minimal infrastructure and operational cost. For these reasons, we now see the wide use of cloud computing in organizations as well as by end users. Data privacy and security is the big concern for the enterprises on cloud environment, the main focus of this thesis is on security and privacy issues concerning data accessibility in cooperative environments, which requires particularly strict control models and solutions.

1.1.3 Contributions

The scope of this dissertation includes the study of the following main issues.

Minimum cost rule enforcement for cooperative database access Parties are sharing their data to provide some set of online services, each party assumed to store its data in private relational databases, and is given a set of mutually agreed

set of authorization rules that specify access to attributes over individual relations or joins over relations owned by one or more parties. The access restrictions introduce significant additional complexity in rule enforcement and query planning as compared with a traditional distributed database environment. In our work Le, Kant, Athmnah, & Jajodia (2016), we examined the problem of minimum cost rule enforcement which simultaneously checks for the enforceability of each rule and generation of minimum cost plan of its execution.

Allowing conditional access using selection conditions and handling nega-

tive rules A model query is intended for a party, say P, and gives this party query ability over either an individual table; or over a join of two or more tables from the same or different parties. The model queries express what queries the parties have agreed to allow. Allowing parties access to entire columns is appropriate in many situations; since many frequently needed restrictions can be imposed through joins and projections (e.g., a hospital shares data with an insurance company for only those patients that subscribe to that insurance company). In other cases, it is adequate to share data for attribute values are in a certain range (e.g., share data only for senior citizens or children). Another form of limiting the given access to parties is through applying safety properties. They explicitly express prohibitions to ensure that sensitive information is not disclosed. In our work Athamnah & Kant (2016a), we included selection conditions to allow conditional access for rule enforcement and query planning. We also addressed the issue of resolving conflicts, between query plans and safety properties where the safety properties are stronger than model queries, for this reason, we proposed an algorithm that weakening of the model queries.

Querying in collaborative databases using graph property A graph is a natural way to represent the entities and complex relationships between them. Graph databases have been widely used recently Hwang & Li (2010), we extended our model

to cover graph databases, and how to use its capability in modern databases applications. In our work Athamnah & Kant (2017) we proposed a new technique to do similarity search between tuples, where we represented the relational database tuples as a graph with weighted edges, which enables queries involving "similarity" across the tuples.

Querying using attribute correlations on graph databases In practice, it is common for data in relational databases to have correlations between its attributes. If these correlations can be identified it may be possible to speed up multi-attribute searches at the cost of result accuracy. For example, in e-commerce data sets high-end features come with high prices, we used this feature to identify what we call dominating attributes. A dominating attribute is an attribute whose value gives us a strong indication of the range of values that other attributes may take. In Athamnah & Kant (2017) we defined an algorithm to exploit the correlations between attributes to create virtual attributes that can be used to catch much of the data variance, and enhance the speed at which similarity search occurs.

Control system testing for configurations in Multi-Party environment Misconfigurations are known to be responsible for an overwhelming percentage of failures, poor service, and exploitation by hackers for cyberattacks. Large-scale cyber and cyberphysical infrastructures are naturally composed of multiple administrative domains, each managed or controlled by a party with potentially unique designs, policies and configurations that are not shared with other parties. This does not only make diagnosis of problems difficult, but is itself often a cause of configuration mismatches and problems Butler et al. (2010); Caesar & Rexford (2005); B. Zhang & Al-Shaer (2011). In Athamnah et al. (2018) we proposed a framework to limit, and control the diagnosis process in case of misconfigurations using a well-defined model controlled by a given access rules.

1.2 Thesis Outline

In this thesis, the first chapter 2 discusses authorization model for secure cooperative data access scenarios, the problem of combining rule enforcement and query planning. In chapter 3, we study the problem of providing limited access to the accessible data and how to weaken the model queries if we have conflicts with safety property or selection conditions. In chapter 4, we discuss how to extend our model to cover graph databases, where we provide a mechanism to conduct similarity search technique. In chapter 5, we study the issue of controlling the diagnosis process and we propose a framework for misconfiguration diagnosis in multiparty environment. Chapter 6, conclusion and future work.

CHAPTER 2

MINIMUM COST RULE ENFORCEMENT FOR COOPERATIVE DATABASE ACCESS

2.1 Introduction

Providing rich services to clients with minimal manual intervention or paper documents requires the enterprises involved in the service path to collaborate and share data in an orderly manner. For instance, to enable automated shipping of merchandise and status checking, the e-commerce vendor and shipping company should be able to exchange relevant information, perhaps by enabling queries to retrieve data from each other's databases. Similarly, in order to provide integrated payment and payment status services to the client, the e-commerce vendor needs to share data with the credit card companies or other vendors that specialize in payment processing. There may even be a need for some data sharing between the payment processing and shipping companies so that the issue of payment for shipping can be smoothly handled.

Traditionally, such cross enterprise data access has been implemented in ad hoc ways. In particular, incoming queries may not be allowed to directly access the databases maintained by a company, and instead handled via some intermediate mechanism. More significantly, cross-enterprise data access is typically driven by bilateral agreements between the two parties that no other party knows anything about. While attractive from isolation perspective, such bilateral agreements introduce a high degree of cost, complexity, and inefficiency into the processes. In

particular, bilateral agreements may require more data to be exposed to other parties so that it is possible to answer complex queries that require composition of data from multiple parties. Bilateral agreements also rule out possibilities of sharing computation results between parties. For instance, if the e-commerce company needs to get information involving join of data over three parties (e.g., the e-commerce company itself, a warehouse, and a shipping company), under bilateral agreements, we have to bring the relevant data from the other two parties to the e-commerce company first and then do joins. With multiparty interactions enabled, such data may already be available. The purpose of this work is to explore the general *multi-party collaboration* model and to develop algorithms for safely implementing the authorization rules so that only desired data can be accessed by authorized parties.

We expect the multi-party data sharing to be driven by twin consideration of business need and privacy; therefore, the rules should grant sufficient privileges for answering the agreed upon set of queries but no more. We assume that the collaborating parties generally trust one another and play by the rules. Typically, this would be enforced through legal and financial provisions in the agreements, but there may still be a need to take the “trust-but-verify” approach. The verification issue is beyond the scope of this chapter and will be addressed in future work. The purpose of this chapter is to focus on efficient mechanisms for executing queries in what amounts to a distributed database with access restrictions. To the best of our knowledge this is the first work of its kind, even though query planning in distributed databases has been considered extensively.

Although the enterprise data may appear in a variety of forms, this chapter focuses on the relational model, with *authorization rules* specifying access to certain attributes over individual relations and some restricted joins over them. In our model, we only allow certain joins because we consider joins among data from different entities for collaboration, the joinable attributes among these data must be agreed among

the data owners along with suitable mapping to make the value syntactically and semantically compatible. In particular, we acknowledge that data sharing across parties brings in the well-known and long-standing problems of differing syntax, semantics, and schemas. We do not presume to solve this very difficult problem, and instead take the view that since the parties are interested in collaboration, they have the incentive to provide suitable interfaces to allow for meaningful composition of data across parties.

For instance, a hospital may join its attribute “patient_id” with the attribute “insured_id” in the insurance company’s data. We assume that the two parties know that these attributes refer to the same entity (identity of a person) and they each provide a “stub”, if necessary, to convert the ID’s to a form where every individual will have the same representation. However, arbitrary joins on non-key attributes will mostly generate useless information for the collaborating entities. Thus, we always assume the join attribute should be at least key of one of the relations, and collaborating parties should pre-define allowed/expected joins and suitable interfaces for them. For simplicity and schema level treatment, we do not consider tuple selections as part of the rules in this chapter. The problem then is to find ways of enforcing the rules and constructing efficient query plans.

Since each party is likely to frame rules from its own perspective, the rules taken together may suffer from inconsistency, unenforceability, and other issues. The *consistency* refers to the property that if a party is provided access to two relations, say R and S , then it must have access to its composition $R \bowtie S$. As mentioned above, only the compositions on certain join attributes make the results useful in collaboration. Therefore, we don’t consider the results of any random cartesian products between R and S on attributes that are not semantically compatible as data authorizations among the parties, because such results won’t convey any useful information for the party collaboration. Also, such random combinations are unlikely to cause significant

information leakage concerns. Consistency is introduced for practical reasons – it is very difficult to know, much less control, what a party does with the data that it can access. As discussed in detail in Le et al. (2012a), consistency can be enforced by taking the given rules and generating all valid compositions of them. This leads to an *explicit representation* of all rules and implies that any valid access will be authorized by exactly one rule.

The enforceability issue can be illustrated as follows: If a party P is given access to $R \bowtie S$ but it and no other party has access to both R and S , it is not possible to actually compute $R \bowtie S$. Alternately, some other party may be able to compute $R \bowtie S$, but not get access to all the attributes that party P is given access to. We considered the issue of enforceability and potential changes to rules to ensure enforceability in Le et al. (2013a). In addition to direct changes to the rules, another way to enforce rules is by introducing *trusted third parties* (TP) that can obtain the necessary data and do the required manipulations. In Le et al. (2013b), we studied the special case of a single TP trusted by all parties. In this chapter, we extend the analysis to multiple TPs with partial trust relationships.

Once the rules have been established to be enforceable, the next question is that of generating efficient plans for queries. A valid query can always be broken up into subqueries each of which is authorized by one of the rules. For each subquery, we ideally want an optimal plan, i.e., a plan with smallest overhead of data transmission among parties and joins at a party. We have examined this problem in Le et al. (2014) where we proposed a fast heuristic algorithm, since the problem of optimal query planning is not only NP-hard (as in traditional query planning literature) but also substantially more complicated because of the possibilities of exchange of partial results among parties.

In this chapter we seek a single combined enforcement and query planning algorithm instead of the two different ones studied in Le et al. (2013a) and Le et al. (2014).

For this, we consider the rules as queries also – which they are. Basically, the algorithm checks for enforcement of each rule, but uses a cost metric to choose a minimum cost enforcement plan whenever the enforcement is possible. It is important to note here that our focus here is not on formulating the most appropriate cost metric, but rather on efficient enforcement given a suitable cost metric. The key advantage of a combined algorithm is that the plans of all rules can be precomputed and stored, and then simplified for specific queries (by removing retrieval of unnecessary attributes). The algorithm also has some key enhancements so that it is less likely to generate suboptimal query plans. The chapter also provides more insights into the single TP case discussed in Le et al. (2013b) and extend the analysis to multiple third parties.

The rest of the chapter is organized as follows. Following the related work in Section 4.3, the problem is defined formally in Section 5.4. This section also illustrates why our problem is more complex than classical query planning and studies its complexity. Section 2.4 then describes a heuristic algorithm for combined enforcement and query planning and discusses its performance. Section 2.5 considers the issues in involving single and multiple third parties for rule enforcement and query planning. Section 5.7 then concludes the discussion.

2.2 Related Work

The problem of collaborative data access has been considered in the past, and this has inspired our multi-party collaboration approach. In particular, De-Capitani, et.al. De Capitani di Vimercati et al. (2008) consider such a model and discuss an algorithm to check if a query with a given query plan tree can be safely executed. However, this work does not address the problem of how the given rules are implemented and how the query plan trees are generated. The same authors have also proposed a possible architecture for the collaborative data access in di Vimercati et al. (2011) but this work does not address query planning. As we shall show shortly,

regular query optimizers cannot be used here since they do not comprehend access restrictions and may fail to generate some possible query plans.

There are also existing works on distributed query processing under protection requirements Cali & Martinenghi (2008); Li (2003) which consider a limited access pattern called binding pattern. It is assumed that the accessible data is based on some input data. For instance, if a party can provide names and ID's of some individuals, it may be allowed to access their medical records. This is a completely different model from ours. There are also many classical works on query processing in centralized and distributed systems Bernstein et al. (1981); Kossmann (2000a); Chaudhuri (1998), but they do not deal with constraints from the data owners, which differs from our work.

Answering queries that takes advantage of materialized views is another well investigated research direction. Some of these works focus on query optimization Goldstein & Larson (2001) which use materialized views to further optimize existing query plans. In our case, we need to generate a query plan from scratch. Some works use views for maintaining physical data independence and for data integration Pottinger & Halevy (2001). They assume the scenario where data is organized in different formats and comes from different sources, and accessing data via views may not provide the complete information to answer the queries. Using authorization views for fine-grained access control is discussed in Rizvi et al. (2004), and Z. Zhang & Mendelzon (2005) analyzed the query containment problem under such access control model. Similarly, conjunctive queries are used to evaluate the query equivalence and information containment, and the work Halevy (2001) presented several theoretical results. Compared to these works, our data model is homogeneous across the parties, and our authorization model not only puts constraints based on relational views but also the interactions among collaborating parties. Consequently, generating a query plan in our scenario is even more complicated. Some results from these works can be complementary to our work and can be used to further optimize the query plans

generated by our approach. However, this is out of the scope of this chapter.

In addition, there are services such as Sovereign joins Agrawal et al. (2006) to provide third party join services; we can think this as one possible TP model in our scenario. There is also some research Aggarwal et al. (2005); Ciriani et al. (2009); Sion (2005) about how to secure the data for out-sourced database services.

2.3 Authorization Model

In this section we discuss our basic model for collaborative access control and query processing involving relational databases owned by a number of parties. The model is based on many of the issues discussed informally in Section 5.1. We will illustrate various concepts using a simple running example, and thus we start with the example first.

2.3.1 A Running Example

This example describes an e-commerce scenario with five parties: (a) *E-commerce*, denoted as E , is a company that sells products online, (b) *Customer_Service*, denoted C , provides customer service functions (potentially for more than one Company), (c) *Shipping*, denoted S , provides shipping services (again, potentially to multiple companies), (d) *Warehouse*, denoted W , is the party that provides storage services, and (e) *Supplier*, denoted as P , that supplies the product to the warehouse. To keep the example simple, we assume that each party owns but one table described as follows. In reality, each party may have several tables that are available for collaborative access, in addition to those that are entirely private and thus not relevant for collaborative query processing.

1. E-commerce (order_id, product_id, total) as E
2. Customer_Service (order_id, issue, agent) as C

3. Shipping (order_id, address, delivery_type) as S
4. Warehouse (product_id, supplier_id, location) as W
5. Supplier (supplier_id, supp_name, factory) as P

The relations are self-explanatory, with underlined attributes indicating the key attributes. In the following, we use *oid* to denote *order_id* for short, *pid* for *product_id*, *sid* for *supplier_id*, *addr* for address, and *delivery* for *delivery_type*. Relations *E*, *C*, *S* can join over their common attribute *oid*; relation *E* can join with *W* over the attribute *pid*, and *W* can join with *P* over *sid*. Assuming an underlying universal relation in this example, it is easy to see that all relations are in BCNF with only lossless joins allowed. (More on this in the model discussion).

Beyond access to its owned table, each party may be given additional authorizations, henceforth called rules, that involve data from one or more parties. A rule r_t can be considered as a triple $[A_t, J_t, P_t]$, where P_t is the party that is given the access, A_t is the set of accessible attributes, and J_t is the sequence of joins among tables over which the authorization is given. For brevity, we call the latter as a *join path*. Table 4.2 describes all the rules with A_t , J_t , and P_t appearing in columns 2-4. In order to make the rule set explicit and complete, the specification must include (a) rules that merely state accessibility of the owned table by a party (e.g., rule #1 in Table 4.2), and (b) rules that represent composition of more basic rules to satisfy the consistency property. For example, in Table 4.2, specifying basic rules (1) and (2) obligates us to add the derived rule (3) that gives P_W access to $E \bowtie_{pid} W$.

2.3.2 Assumptions and Definitions

We consider a group of collaborating parties each with a sharable relational database (SDB) with collectively known schema. These SDBs are unlikely to be the internal databases maintained by the parties for their private operations; instead

#	Authorized attribute set	Auth. Join Path	To
1	{pid, sid, location}	W	P_W
2	{oid, pid}	E	P_W
3	{oid, pid, sid, location}	$E \bowtie_{pid} W$	P_W
4	{pid, sid, factory}	$P \bowtie_{sid} W$	P_W
5	{oid, pid, sid, location, factory}	$E \bowtie_{pid} W \bowtie_{sid} P$	P_W
6	{oid, pid, total}	E	P_E
7	{oid, pid, total, issue, agent}	$E \bowtie_{oid} C$	P_E
8	{oid, pid, location, total, addr, delivery}	$S \bowtie_{oid} E \bowtie_{pid} W$	P_E
9	{oid, pid, sid, total, factory}	$E \bowtie_{pid} W \bowtie_{sid} P$	P_E
10	{oid, pid, sid, total, factory, location, addr, delivery}	$S \bowtie_{oid} E \bowtie_{pid} W \bowtie_{sid} P$	P_E
11	{oid, pid, issue, total, agent, addr, location, delivery}	$S \bowtie_{oid} E \bowtie_{oid} C \bowtie_{pid} W$	P_E
12	{oid, pid, sid, issue, total, agent, factory, location, addr, delivery}	$S \bowtie_{oid} E \bowtie_{oid} C \bowtie_{pid} W \bowtie_{sid} P$	P_E
13	{oid, addr, delivery}	S	P_S
14	{oid, pid, total}	E	P_S
15	{oid, pid, total, addr, delivery}	$E \bowtie_{oid} S$	P_S
16	{oid, pid, total, location}	$E \bowtie_{pid} W$	P_S
17	{oid, pid, location, total, addr, delivery}	$S \bowtie_{oid} E \bowtie_{pid} W$	P_S
18	{oid, pid}	E	P_C
19	{oid, issue, agent}	C	P_C
20	{oid, pid, issue, agent}	$E \bowtie_{oid} C$	P_C
21	{oid, pid, issue, agent, total, addr, location}	$S \bowtie_{oid} C \bowtie_{oid} E \bowtie_{pid} W$	P_C
22	{sid, sname, factory}	P	P_P
23	{pid, sid}	W	P_P
24	{pid, sid, sname, factory}	$W \bowtie_{sid} P$	P_P

Table 2.1: Auth. rules for running Example

they represent “sanitized views” with following characteristics:

1. Only the columns (attributes) intended to be shared with other parties are included in the SDBs.¹

¹It is also possible that a party wishes to share only those tuples that satisfy certain selection conditions on the shared attributes, but an explicit treatment of this aspect is beyond the scope of this chapter.

2. Each SDB is in a standard form such as BCNF or 3NF.
3. The schema of SDBs along with syntax and semantics of each column and the functional dependencies (FD's) are published in a central repository.
4. Sharing implies that the SDBs across parties will have certain attributes that are intended to represent same semantic entities (e.g., customer ids, part numbers, diagnosis, ...). These *corresponding attributes* are assumed to be explicitly identified across all parties.
5. Although corresponding attributes could have different names in different SDBs (e.g., "patient_id" for a hospital vs. "insured_id" for insurance company), we can assume, without loss of generality, that they are mapped to identical names.
6. For any pair of corresponding attributes, say A_1 and A_2 across SDB₁ and SDB₂, queries or rules involving their comparison are allowed only if (a) there are published functions f_1 and f_2 such that $f_1(A_1)$ and $f_2(A_2)$ convert them to a common format, and (b) Identical converted values mean identical instances. For example, if A_1 and A_2 refer to customer ids, it is essential that after conversion, the same value means the same customer.

Two important subcases where the last assumption is required are (a) join across corresponding attributes (since equijoin requires a value comparison), and (b) selections that compare values. For the purposes of this chapter, only (a) is relevant. In our running example, this means, for example, that "order_id" in tables E, C, S refer to the same entities and can be treated as having identical representation.

Although the issue of how the representation of and operations on SDBs relate to the operations on internal DBs of parties is an important issue, we do not address it here.

Access Rules Across SDBs

With the above assumptions (including #6), we can allow access rules involving joins on corresponding attributes. Joins on other attributes are likely meaningless and not of much value. We further require that (a) at least one of the two corresponding attribute sets in a join is a key attribute set in its SDB², and (b) all joins are inner equijoins. We call these as “meaningful joins” and all rules are restricted to such joins. Such joins have properties similar to “lossless join” within a party where universal relation can be assumed, and is adequate for our purposes. If a party has access to two corresponding attributes neither of which is a key into its SDB, it is free to join them to glean some information, but it will not have the advantage of assumption #6, and the join may produce incorrect associations. Concern regarding information leakage in such situations can be partly addressed via judicious design of SDB schema and access rules, but we do not consider this aspect in this chapter.

Henceforth we only consider meaningful joins and refer to them as simply joins. We assume that the (meaningful) join schema is known across all parties. The join schemas can be flat or hierarchical, but we do not allow cyclic schemas. Of course, depending on the access rules, only some of the joins may be possible for a given party. We assume that the set of accessible attributes of a relation (single or one obtained via a meaningful join over other relations) always includes its key since access to a relation without access to key is generally not very useful. We also assume all keys to be primary keys; even though our model can handle foreign keys properly, it is unlikely that the collaborating parties will impose foreign key restrictions on one another, and thus we do not explicitly consider foreign keys. Now we can formally define the notion of join path as follows:

Definition 1. A **join path** $J_t = \{JR_t, JA_t\}$ represents a series of *equi-joins* over the re-

²Since a relational key could, in general, include multiple attributes, we need to speak of set of attributes forming the key, or *key attribute set* for short. For the same reason, we need to speak of *join attribute set* rather than a single join attribute.

lational schemas $JR_t = [R_1, R_2 \dots R_n]$ with join predicates $JA_t = [(A_{l1}, A_{r1}), (A_{l2}, A_{r2}) \dots (A_{l,n-1}, A_{r,n-1})]$ respectively, where (A_{li}, A_{ri}) are the join attributes from the i th and $(i + 1)$ st relation schemas R_i and $R_{(i+1)}$. Here n can be regarded as the **length** of a join path.³

As stated earlier, the rules in our model specify accessible attributes over a join path. We specifically disallow tuple selection conditions in rule definitions in this chapter; we are currently working on extending our results to include Selections, which will be reported in future work. Our queries will also be limited to simple select-project-join (SPJ) queries, with selections further replaced by access to the attributes mentioned in the selection conditions.

Given the rules that involve join to relations from multiple SDBs, the parties would need to store the relations obtained from such joins. While the intermediate results during the rule enforcement may be stored only temporarily, the one required to enforce the rules may be stored for longer periods instead of recalculating them every time. Such extra relations could be considered as part of the SDB of the party that obtains them. Therefore to distinguish the entire SDB from the original part put up by a party, we called the latter as **original SDB**.

The relational data model allows several other operations beyond SPJ (select-project-join). A few common operations are union, difference, and intersection of two relational schemas. Since we assume that the schemas are shared, it is trivial to determine what accesses a party has; however, such operations are unlikely to be very useful in a multiparty environment. The same applies to tuple level operations. For example, if we allow a party access to the union of tuples from relations R and S , it is no different than allowing access to R and S individually. If the access is to the intersection, we would first need that R and S have identical schema in the sense of assumptions 1-6 in section 2.3.2. This is unlikely to be the case in general,

³According to this definition, $n = 1$ corresponds to the case of no join.

although one could easily consider intersection operation for defining consistency and enforceability.

Based on the discussion above, we collect together the key assumptions here to make them more explicit.

1. Only “meaningful joins” are addressed in this chapter; parties could, of course, do other types of joins, but we make no claim about their usefulness in retrieving information, or the additional information leakage produced by them.
2. The schema concerning meaningful joins is known to all collaborating parties.
3. The set of accessible attributes to a party always includes the key attributes.
4. All parties involved are considered cooperative and non malicious in this chapter; therefore, issues of cheating or holding back information are not considered here.

2.3.3 Authorized Queries Across Parties

A query can originate from any collaborating party, but will be allowed only if the originating party has adequate accesses to authorize the query. A query q in our model can be represented by a pair $[A_q, J_q]$, where A_q is the set of attributes appearing in the Selection and Projection predicates. For instance, the SQL query

q: `Select oid, total, addr From E, S On E.oid = S.oid Where delivery = 'ground'`
 can be represented as the pair $[A_q, J_q]$, where J_q is the join path $E \bowtie_{oid} S$ and A_q is the set $\{oid, total, addr, delivery\}$ requested on this join path.

Now let us consider the query authorization. Since we represent all rules explicitly (including the derived rules), the query must have the same join path as some rule, and request no more attributes than the rule allows. We formalize this notion with equivalent join path:

Definition 2. We say that two join paths J_i and J_j are **equivalent**, henceforth denoted as $J_i \cong J_j$, if J_i and J_j involve the same set of relations and join attributes, and J_i is identical to some valid permutation of joins in J_j .

Now we can define an authorized query as follows:

Definition 3. A query q is **authorized** (\succeq) if there exists a rule r_t such that $J_q \cong J_t$ and $A_q \subseteq A_t$.

To be clear, we are not considering query rewriting here. The queries are still evaluated on the original set of relations. Answering an authorized query requires a sequence of operations (i.e., projection, join, and data transfer), each of which must be consistent with the given rules. The specific consistent operation requirements are as follows:⁴

1. For a projection (π) to be consistent with the rule set \mathcal{R} , there must be a rule r_p that authorizes (\succeq) the input information.
2. Join (\bowtie) is a binary operation where two input relations R_{i1} and R_{i2} (possibly obtained via other operations) produce the resulting plan $R_o = R_{i1} \bowtie R_{i2}$. For a join operation to be consistent with \mathcal{R} , all the three relations need to be authorized by rules for some party.
3. Data transmission (\rightarrow) involves an input relation R_i produced by a party P_i and an output relation R_o desired by a party $P_o \neq P_i$. If there are rules $r_i, r_o \in \mathcal{R}$ with equivalent join paths (i.e., $J_i \cong J_o$), and $r_i \succeq R_i, r_o \succeq R_o$, then the data transmission operation is consistent with \mathcal{R} .⁵

Definition 4. A *query plan* for an authorized query $q = [A_q, J_q]$ is a sequence of operations (projections, data transfers, joins) starting with individual SDBs that

⁴The notion of an operation being *consistent with rules* should not be confused with the earlier notion of *consistency of the rule set itself*. The latter refers to the rules being closed under all meaningful joins.

⁵If P_i is sending information with attributes not in A_o , P_i should do a projection operation $\pi_{A_o}(p_i)$ first.

ultimately results in retrieving the attribute set A_q over the join path J_q . Furthermore, if each operation is consistent with the given rule set, we call the plan as a **consistent query plan**.

A query plan is naturally organized as an hierarchy starting with the original SDBs of each party and moving up in join path length order. We can consider the plan at intermediate steps as a *sub-plan*. While such a recursive procedure applies to classical query planning as well, the unique issues in our case include: (a) data transfers among parties to retrieve “missing” attributes, and (b) ensuring that each operation is consistent with the rules. These issues makes our query planning different and more complex than classical distributed query planning as discussed later in section 2.3.4. Therefore, our focus is on issues like least expensive ways of retrieving the required attributes rather than the classical issues of indexing, disk accesses, join orders, etc. It is certainly possible to integrate the traditional data access details with our query plans to solve real world problems, but we do not discuss that aspect in this chapter.

Since a query is ultimately authorized by a rule, there is a close relationship between queries and rules in our model. In particular, rules can be viewed as model queries themselves, and thus generating efficient plans for enforcing rules is one way of handling queries and is the approach taken in the chapter. It allows the plans to be generated ahead of time, and then suitably modified by skipping the retrieval of the attributes that the query does not need.

As an example, consider enforcement of rule r_3 in Table 4.2. This involves a join over two subplans based on rules r_1 and r_2 respectively. The subplan for r_1 is to access table W on P_W . The subplan for r_2 is an access plan reading table E by P_W which requires data transfer from P_E to P_W . The example plan authorized by r_3 has the $J_{pl} = E \bowtie_{pid} W$, and $A_{pl} = \{oid, pid, sid, location\}$.

In the context of enforceability, we call a rule as *Target Rule* r_t expressed as

$r_t = [A_t, J_t, P_t]$ where A_t is the *Target Attribute Set*, J_t is the *Target Join Path*, and P_t is the *Target Party*. There are two important aspects in enforcing r_t : (a) enforcement of the target join path J_t , and (b) enforcement of the target attribute set A_t . Enforcement of J_t means that party P_t has a consistent plan such that it can obtain the desired joins on the join path J_t . This would require, at a minimum, the access all key attribute set at each step of the join sequence. In some cases, a rule does not have a total enforcement plan but only some partial plans. A **partial enforcement plan** means the join path can be enforced, but the attribute set of the plan is a proper subset of the desired set A_t . We say that an attribute set is a **maximal enforceable attribute set** for a rule, if it is enforced by a plan of the rule, and there is no other plan for the same rule that can enforce a superset of these attributes. If the maximal enforceable attribute set is equal to rule attribute set A_t , the rule is **totally enforceable**.

On the same party, we call a join path as a **sub-join path** of J_t if it contains a proper *subsequence* of relations of JR_t . Rules *not* on the sub-join paths are not relevant to r_t since any composition with these rules results in information more than what r_t authorizes. At party P_t , a plan that is on a sub-join path of J_t is a **relevant plan**, and the rule authorizing it is a **relevant rule** of the target rule. Parties having rules defined on the equivalent join path of J_t are called **J_t -cooperative parties**, and information regulated by J_t is allowed to be exchanged only between these parties.

2.3.4 Inadequacy of Classical Query Planning

Generating a consistent plan that answers an authorized query in our scenario is much more complex than the well-studied problem of query planning for distributed databases (without any access restrictions). We illustrate this by an example. Suppose that there are two collaborating parties P_R and P_S with database schemas $R(\underline{A}, B, C)$, and $S(\underline{A}, D, E)$ respectively (A is the key attribute for both relations).

The party P_R has an authorization rule $r_R = \{A, B, C, D\}, R \bowtie S$ (in addition to access to its own data). The party P_S has two authorization rules: $r_{S1} = \{A, B\}, R$ and $r_{S2} = \{A, B, C, D, E\}, R \bowtie S$. Let us now consider how to generate a consistent plan to answer a query for $\{A, B, C, D, E\}$ over the join path of $R \bowtie S$.

In classical query planning, we will generate a query plan tree and try to assign the appropriate operations to different parties. There is no constraint of data access in classical case. Therefore, either party P_R or P_S can retrieve the other relation and do the join to answer the query. From performance considerations, semi-joins Kossmann (2000a) are usually used in the distributed query processing. However, in our case, even a semi-join is not enough to generate the consistent query plan for the query. It is clear that neither P_R and P_S can obtain the desired result with just one join. If we use the semi-join method, the only possibility is that P_R sends $\{A\}$ to P_S ; P_S does the join and ships $\{A, D\}, R \bowtie S$ back to P_R , which then computes $\{A, B, C, D\}, R \bowtie S$ by doing another join. This, in turn is passed back to party P_S , which then obtains the desired result. In contrast, if we use regular join, then party P_S can have at best the attributes $\{A, B, D, E\}, R \bowtie S$ through one join operation.

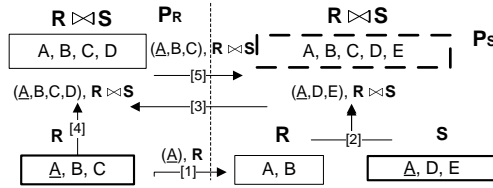


Figure 2.1: Illustration of Query Planning

To generate the consistent plan for answering the query, it is required that we do the semi-join first, and party P_R again sends the $\{A, B, C\}, R \bowtie S$ to party P_S . Another join operation at party P_S could then give the required query results. Figure 2.1 illustrates the situation. Each box is a rule, with attributes inside the box and the join path on top of the box. (To avoid clutter, the attributes involved in the join are not identified.) The dotted line in the middle separates the parties P_R and

P_S and the arrows across represent data transfers along with indication of operation sequence. The authorization rule that authorizes the *final* query is in dashed box. The numbers on the arrows indicate the ordered steps for the consistent query plan. It is clear that generating a consistent query plan under the data access constraints can be lot more complicated than for distributed query planning. In the following section, we show the complication of query processing in cooperative data access environment.

2.3.5 Complexity of Cooperative Query Planning

It is well known that the optimal query planning in the context of single party database access is NP-hard. The most significant aspect of this classical problem is the order in which the joins should be performed, since the number of tuples in a joined relation depends on the *selectivity* among the constituent relations. Join Selectivity is a number between 0 and 1.0 that provides an estimate for the size of the joined relation Kossmann (2000a) as a fraction of the size of their cartesian product. While this aspect remains unchanged in our model whenever there are multiple join order possibilities, our main focus is how to provide an optimal enforcement (with respect to joins, data transfers, and attribute retrievals) in the presence of rather complex accessibility constraints. However, as the proof of the next theorem shows, even if we simplify the problem to an extent that is straightforward in a classical query planning scenario, it is still NP-hard with the constrained access in our model.

Theorem 1. With any fixed per operation cost metric, finding the optimal query plan to answer an authorized query is *NP-hard*.

Proof. Consider two basic relations R and S which can join together, a set of attributes $U = \{A_1, A_2, \dots, A_n\}$, and a distinct attribute A_0 that we will use as key for join. We define a relation R with the schema $\{\underline{A_0}, A_1, A_2, \dots, A_n\}$, S has the schema

$\{A_0, A_x\}$ where A_x is not in U . We also have $m > 1$ sets $\{S_1, S_2, \dots, S_m\}$, where each S_i is a set of elements from U .

1. Party P_0 is given a rule r_0 that authorizes it to retrieve the entire set U over the join path $R \bowtie S$. Note that P_0 cannot enforce the join path as it has no access to R nor S directly.
2. Each of the other parties P_i , $i = 1 \dots n$, has a rule r_i on the join path $R \bowtie S$ with attributes $S_i \cup \{A_0\}$. On these parties, access to basic relations R and S is also given so they can enforce their own rules locally.

P_0 cannot locally do the joins in join-path but other parties can enforce their rules r_i locally, and their costs are known. Therefore, for P_0 to answer the query, it needs a plan bringing attributes from other parties and merging them at P_0 (multi-way join on attribute A_0) to answer the query. The optimal plan needs to choose the rules with minimal costs, and the union of their attribute sets must cover the query attribute set. Assuming the same cost for doing the join locally and sending the particular results to A_0 , we effectively have the classical set covering problem which is known to be NP-hard. In other words, the set covering problem can be reduced to a special case of our problem, which means that our problem must also be NP-hard.

□

We note that the above proof maps the set covering problem to a very simple version of our problem – one without complex costs or access restrictions. While these additional complexities would not change the theoretical complexity, they make it more complex to enumerate all feasible solutions. We discuss these and other challenges in the following.

To generate a consistent plan for a query, we first need a plan that enforces the query join path. This can be further joined with other plans to get all the requested attributes. Obviously, in order to consider a join path of length n , one needs to

consider all top level join subpaths of lengths k and $n - k$ for suitable values of k . Moreover, since a longer join path will generally produce relations with fewer tuples, it is usually desirable to consider joins of overlapping relations. For instance, generating a join path of $R \bowtie S \bowtie T$ may be better done as $(R \bowtie S) \bowtie (S \bowtie T)$ instead of, say, $(R \bowtie S) \bowtie (T)$. The ability to do this depends not only on the relation sizes and join selectivity, but also on authorization rules and data locations (since data transfers represent additional cost and delay). All such choices need to be considered for an optimal result.

An added difficulty is that we cannot just pick the subpaths based on the join or data transfer cost – we also need to pay attention to the attributes that can be accessed by doing the join. For instance, if the goal is to answer $\{A, B, C, D\}$ on join path $R \bowtie S \bowtie T$, we may have two ways of getting it: (a) A subplan pl_1 that yields that attribute set $\{A, B\}$, and (b) A higher cost subplan pl_2 that yields the attribute set $\{A, C, D\}$. Since we need more work to get the missing attributes, at this stage we cannot even pick one of these, and instead must keep both. Thus, in general, we need to maintain many partial enforcement plans. For each such plan, we then need to consider the problem of retrieving the missing attributes. This, in turn, requires checking all possible combinations of relevant rules, followed by a recursive procedure to find enforcement plan for the chosen relevant rules. It is clear that the exhaustive enumeration to find the globally optimal answer can be extremely expensive.

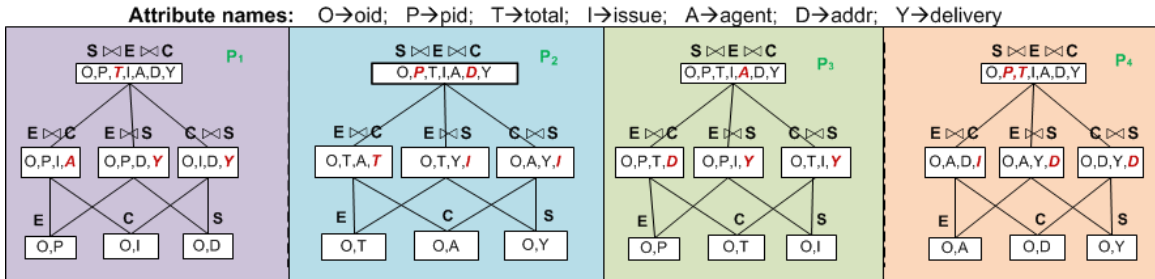


Figure 2.2: Illustration of Enforcement Complexity

In order to show the high complexity of optimal enforcement, Fig. 2.2 shows a

situation crafted based on but not limited to our running example. It considers 3 relations, namely, S , E , and C that can join over the same attribute oid . We also consider the same set of attributes, but abbreviate them to single letter as given by the legend in the Figure. We introduce 4 parties namely P_1, P_2, P_3, P_4 , and give them rules for basic relations (level 1), single joins (level 2) and two-way joins (level 3). For example, P_1 has access to attributes (O, P) (oid and pid) on relation E , (O, I) on C , and (O, D) on S . Because of the *consistency* assumption, lower level rules can combine to give higher level rules as shown by the edges; however, we have chosen higher level rules such that there is always one missing attribute as shown in red and bold-italic font. Thus, for example, P_1 has A missing on $E \bowtie C$ and must get it from another party, which in this case could be party P_2 or P_4 . Similarly, missing attributes for other parties and other join paths can be obtained from 1 or two other places. This pattern continues at level 3 as well, as shown. Given a suitable cost structure (not shown), one needs to examine a large number of possibilities to check for availability of attributes and then evaluated the corresponding costs.

2.4 Combined Rule Enforcement and Query Planning

In this section we discuss a combined enforcement/query planning algorithm for all rules based on suitable cost assignments. In this process, if we find that some rules cannot be enforced, our algorithm will simply mark them as unenforceable. We assume that the enforcement plans for all rules are derived in advance and stored in a repository. Then at run-time when a query q is received, a simple match over the join path can identify the rule r_t , if any, that can authorize it (since there is at most one such rule). Now if q is asking for attributes not included in r_t , the query is rejected. Otherwise, if the query is asking for fewer attributes, we retrieve the stored plan for r_t and update it in three steps. In the first step, we remove all non-key attributes from data transfers and joins that q does not request. This removal could, in some

cases, result in cross-party data transfer of only key attributes that are not used for any joins. These can be removed. Finally, even if such transfer is used for a join, but the join path does not change, the transfer and join can be removed. While such a plan can sometimes be higher cost than one specifically derived for query q , it makes query plan generation extremely fast.

2.4.1 Enforcement Cost Considerations

There are two major cost factors in our collaborative data sharing model: (a) Cost of data transfer from one party to another, denoted as C_T , and (b) Cost of a Join (or semijoin) denoted as C_J which includes local IO and computational costs. Other costs such as the cost of doing a projection of a relation are less significant and can be ignored. Many cost models are possible for data transfer and join costs. The simplest one, that focuses on number of operations, assumes fixed values for C_T and C_J irrespective of the table sizes involved. We take such an approach in this section, since cost modeling for joins and other relational operations is a very well studied problem in the literature.

As an illustration of our simple cost model, suppose that parties P_1 and P_2 “own” relations R and S respectively. Suppose that party P_3 is given access to R , S , and $R \bowtie S$. Then we consider the cost of enforcing $R \bowtie S$ as $2.C_T + C_J$ since it involves 2 data transfers and one join. Allowing different values of C_T and C_J allows the algorithm to bias the enforcement towards less inter-party data transfer (by choosing $C_T > C_J$) or less computation (by choosing $C_T < C_J$).

It is certainly possible to consider a more sophisticated cost model that depends on data size. We describe such a model here but use it only in Section 2.5.1 in the context of minimum information transfer to third party. For this, we assume that the number of tuples in the relations are known. Assuming that we have the historical statistical information of the tables, so we can estimate the join selectivity and hence

the number of tuples. The join cost also depends on a number of other factors including how the matching tuples are found, the size of the index, and whether it involves IO, etc. The data transfer costs include the cost of sending input relations (with suitable attribute projections) to the third party and retrieving the join result. The data transfer costs depend on the number of attributes sent, attribute sizes, and number of tuples, and can often be reduced by using compression techniques. While it is possible to consider all these aspects in our cost model, this will perhaps obscure the insights that are possible with a very simple cost model. Besides, these aspects have been studied amply in the literature.

It is worth noting that even a good estimate of actual transfer/compute sizes may be inadequate if the third party is allowed to do some caching of data. It is reasonable to assume that the TP will retain data for the duration of a query, but may be allowed additional capabilities as well.

2.4.2 Consistent query planning

Due to the difficulties in enumerating all possible ways of answering a query, we develop a greedy algorithm that simultaneously checks for enforcement and generates an efficient query plan for the rules.

The basic approach is to start with rules involving individual parties, i.e., rules of join path length (JPL) of 1, and then systematically go to longer join path lengths. Enforcement of rules with JPL=1 only involve transfer of relations to parties that have access to them but don't own them. In order to enforce a target rule r_t at a higher level, we first find the lower level "relevant rules" that can be enforced locally by the target party P_t . If this is inadequate, we involve *remote parties* (i.e., parties other than P_t) for enforcement and determine the maximal enforceable attribute set of the rule.

As the algorithm iterates, it also builds a graph structure capturing the enforceable

information and the relationships among the rules. Each node in the graph is an enforceable rule with its maximal enforceable attribute set. All non-enforceable rules and attributes are discarded. Two nodes on the same party are connected if one is relevant to the other. Among different parties, nodes can be connected if they have equivalent join paths. Fig. 2.3 shows part of the built graph for our running example. In particular, party P_W can compute $E \bowtie_{pid} W$ (allowed by rule r_3) using rules r_1 and r_2 . However, the attributes $\{oid, pid, location\}$ on this join path are also accessible to party P_S because of rule r_{16} , which means that P_S can receive these from P_W . Note that P_S does not have access to W , and so it cannot compute $E \bowtie_{pid} W$ directly. On the other hand, P_S has access to both S and E (rules r_{13} and r_{14}), and so can get $E \bowtie_{oid} S$ (authorized by rule r_{15}) directly. S can further join $E \bowtie S$ and $E \bowtie W$ to obtain $E \bowtie_{oid} S \bowtie_{pid} W$ which is authorized by rule r_{17} .

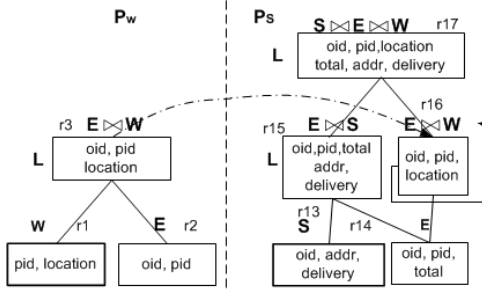


Figure 2.3: Relevance graph for Local Enforcement

The local enforcement attempt of rule r_t by party P_t typically only enforces the join path J_t , since it has missing attributes. In order to retrieve them we need to consider the relevant rules of basic relations on all **J_t -cooperative parties** (cooperative parties that have authorization rules on join path J_t). This can be done through semi-join operations. In such cases, the party P_t can send only the join attributes to its J_t -cooperative party, and the receiving party does a local join to get these attributes and send it back. P_t then performs another join to add these attributes to the query plan. If the r_t is enforceable, the remaining missing attributes can always be found in the relevant rules on J_t -cooperative parties and can again be obtained

via semijoins as needed.

Algorithm 1 Minimum Cost Rule Enforcement

```

1: void Min_Cost_Enforcement ( $C_J, C_T$ ) //  $C_J, C_T$  are Join data transfer costs
2: Sort rules according to join path length (JPL)
3: for  $p=1$  to #parties & each rule  $r_t$  w/  $JPL==1$  do
4:    $Cost(r_t) =$  if owns ( $p, Relation(r_t)$ ) then 0 else  $C_T$ ;
5:   Mark  $r_t$  as totally enforced and add to Graph;
6: for JPL = 2 to  $JPL_{max}$  do
7:   for each rule  $r_t$  of this JPL do
8:      $Att(r_t) =$  Set of attributes in rule  $r_t$ ;  $p = Party(r_t)$ ; // Party that has  $r_t$ 
9:      $RRS(r_t) =$  Rules of length JPL-1 or JPL-2 relevant to  $r_t$ ;
10:     $A_m = Att(r_t)$ ; // All attributes missing initially
11:    Add_node( $r_t$ ); // Add node  $r_t$  to the Graph
12:    if  $r_t$  can be maximally enforced by  $p$  using  $r_1, r_2 \in RRS(r_t)$  then
13:      Add_connections ( $r_t, r_1, r_2, C_T, C_J$ );
14:      // Add connections to  $r$  from  $r_1, r_2$  and update  $r$ 's cost
15:       $A_m = A_m \setminus [Att(r_1) + Att(r_2)]$ ;
16:      if  $A_m(r) \neq NULL$  then {Add ( $r_t, A_m$ ) to Queue};
17:    for each entry [ $r_t, A_m(r_t)$ ] in Queue do
18:       $Orig\_A_m(r_t) = A_m(r_t)$  // Remember original missing attributes
19:       $att\_enforced = 0$ ;
20:      for each rule  $r_s$  with same Join_Path as  $r_t$  do
21:        if  $already\_enforced(r_s)$  then
22:          if  $\#Att(r_s) > att\_enforced$  then
23:             $att\_enforced = \#Att(r_s)$ ;  $r_{new} = r_s$ ;
24:            if  $att\_enforced == \#att(r_t)$  then { retain  $r_{new}$  w/ min cost};
25:          Add_connection( $r_t, r_{new}, C_T$ );
26:           $A_m(r_t) = A_m(r_t) \setminus Att(r_{new})$ ;
27:          if  $A_m(r_t) \neq NULL$  then
28:            Retrieve the max subset of  $A_m(r_t)$  using semijoin w/ rules of  $Party(r_t)$ ;
29:            Update  $A_m(r_t)$ ;
30:          if  $A_m(r_t) \neq NULL$  then
31:            Mark  $r_t$  as unenforceable and remove it from Queue;
32:          else
33:            Remove [ $r_t, A_m(r_t)$ ] from Queue and update its cost in Graph;
34:          for each entry [ $r_t, A_m(r_t)$ ] in Graph do
35:            If there is another entry [ $r_s, A_m(r_t)$ ], choose one with lower cost; Update Graph;
36: end

```

Algorithm 5 sketches the overall process. The first for loop initializes the cost and marks the rules with JPL (join path length) of 1 as totally enforced. The next “for” loop then steps through each JPL in increasing order. Each target rule r_t of this JPL is then picked up and checked for local enforceability (partial or total) by the party by considering joins from relevant rules of JPL-1 and JPL-2. (With JPL=1, there is no JPL-2, of course.) For this, we initialize the set of missing attributes for rule r_t , denoted $Att(r_t)$, as the entire attribute set. That is, all of these attributes for rule r_t

are initially missing. If the join path of r_t can be enforced, perhaps in more than one way, we choose the relevant rules r_1 and r_2 that can join to provide the maximum number of attributes for r_t . We add node r_t to the Graph along with its updated cost and edges to r_1 and r_2 . We then update the missing attributes by subtracting out the attributes provided by r_1 and r_2 . (Note that the operator \setminus is the set theoretic subtraction operator.) It is possible that this partial enforcement does not provide all attributes for r_t . In this case, we need to obtain missing attributes from other parties in the next step. For this reason, we add the rule along with missing attributes to a Queue.

The next for loop, starting at line 17, then tries to obtain missing attributes from other parties for each rule r_t . Here we look through all already enforced rules in the Graph that have the same join path as r_t and check if they provide more attributes than what we have already. If there are many such rules, we choose the one that provides us with the maximum number of missing attributes (see lines 22, 23). Furthermore, if there is more than one rule that can provide all missing attributes, we choose the one with minimum cost (see line 24). We add a connection to this node in the Graph along with updated cost and update the missing attributes. If there are still some missing attributes, we try to obtain the largest subset of those (on the correct join path) by a semijoin with a rule of $\text{Party}(r_t)$. At this stage if there are still missing attributes, we conclude that they cannot be obtained; otherwise, we update its cost in the Graph. In either case, we remove the rule from the Queue at this point.

The last for loop (starting at line 34) is executed after we have gone through all entries in the Queue and thus done with the current JPL value. At this point we do a scan through all the totally enforced rules for current JPL, and check if any rule can be enforced with a lower cost. Note that we no longer consider rules that do not provide all the desired attributes.

The key quantity in the running time of the algorithm is N_{qt} , the number of rules locally relevant to rule q , and N_{qt} , the number of relevant rules on all J_t -cooperative parties. Both of these stay small (i.e., in the range of 10's) even with large number of rules and parties. The overall worst case complexity of the greedy algorithm is $O(N_{qt} * N_{qt}^2)$.

Theorem 2. A query plan generated by Algorithm 5 is consistent with the set of access rules R .

Proof. Based on our defined notion of consistent operations, the basic operations to enforce join paths are consistent. Each join operation step in the plan is added according to a legitimate local join over the relevant rules, and each data transmission operation happens only between J_t -cooperative parties. So, these operations are consistent with the access rules. In the iteration of join path lengths, there are join and semi-join operations between the already enforceable information which is built from the basic relations (JPL is 1) that are always consistent. A join operation between a rule and its local relevant rule is always consistent. A semi-join between a rule and a relevant rule on its J_t -cooperative party is also consistent. It is because the attributes in the relevant rule can be obtained by the rule with J_t on the same party, and the data transmission between two parties is consistent as they are J_t -cooperative parties and the matching authorizing rule gives accesses to these missing attributes. Since each operation in such a plan is consistent, the plan generated by Algorithm 5 is consistent with the access rule set R .

□

2.4.3 Algorithm Performance

In order to avoid exhaustive search and hence the exponential complexity of the exact solution, the algorithm takes two short cuts that could potentially lead to suboptimal results in large examples. First, when looking for local enforcement of a

rule at level n , the algorithm only considers relevant rules at levels $n - 1$ and $n - 2$ rather than at all levels. Second, after generating all enforcements at level n , the algorithm checks all *total* enforcements for the rule and chooses one with minimum cost and updates the graph accordingly. In general, it is possible that some of the partial enforcements can be useful in the following way: the missing attributes in these plans are obtainable via a semijoin with suitable parties and the extra join still yields a cost lower than the one that was chosen by the algorithm. Adding this enhancement can be quite expensive, and thus we do not include it in the algorithm.

Now let us turn to the experimental evaluation of the algorithm to compare its performance against the optimal solution. Ideally, this would involve implementing a brute-force algorithm that examines all (exponential number of) possibilities and thereby determines the optimal solution. Unfortunately, as already illustrated above, the brute force solution gets out of hand very quickly. Therefore, we took the following approach for evaluation. We generated 10 different variants of our running example by changing some rules (i.e., increasing or decreasing the attributes allowed by the rule), deleting/adding some rules, or changing the costs. Then we considered the algorithm generated minimal cost enforcement and optimal cost enforcement for each of the 24 or so rules in each of the 10 variants. In the following we list some of these rules and the costs. To allow for easy interpretation of costs, we assumed that $C_T = C_J = 1$, meaning that the overall enforcement cost is simply the number of operations.

The rule changes listed in the following are as follows:

R3-a: P_W given {oid,pid,total,addr,delivery} over JP $E \bowtie_{oid} S$

R3-b: P_W given {oid,pid,sid,total,factory,location,addr,delivery} over JP $S \bowtie_{oid} E \bowtie_{pid} W \bowtie_{sid} P$

R20-a: P_C given {oid,location,pid,total,addr} over JP $S \bowtie_{oid} E \bowtie_{pid} W$

case	Rule	Changes in Rules	Enforcement Plan	Cost	Opt. cost
1	r_{12}	None	$(r_7 \bowtie (r_8 \bowtie r_9 \rightarrow r_{10})) \rightarrow r_{12}$	20.0	20.0
2	r_{12}	Add r_{3-a}, r_{3-b}	$(r_7 \bowtie (r_{3-b} \rightarrow r_{10})) \rightarrow r_{12}$	15.0	15.0
3	r_{12}	Add r_{3-a}, r_{3-b} (W/o factory attribute)	$(r_7 \bowtie (r_8 \bowtie r_9 \rightarrow r_{10})) \rightarrow r_{12}$	20.0	20.0
4	r_{11}	Remove r_{22}, r_{23}, r_{24}	$(r_7 \bowtie r_8) \rightarrow r_{11}$	12.0	12.0
5	r_{11}	Remove r_{22}, r_{23}, r_{24} , Add total attribute to (r_2, r_3)	$(r_7 \bowtie r_8) \rightarrow r_{11}$	11.0	11.0
6	r_{11}	Remove r_{22}, r_{23}, r_{24} , Add r_{20-a}	$r_{21} \rightarrow r_{11}$	11.0	11.0
7	r_{11}	Remove r_{22}, r_{23}, r_{24} , Add $r_{20-a}, C_T = 2$	$(r_7 \bowtie r_8) \rightarrow r_{11}$	17.0	17.0
8	r_{17}	Remove r_{22}, r_{23}, r_{24} , Add r_{20-a}	$(r_{15} \bowtie r_{16}) \rightarrow r_{17}$	7.0	7.0
9	r_{16}	Remove r_{22}, r_{23}, r_{24} , Add r_{20-a}	$(r_3 \bowtie r_{14}) \rightarrow r_{16}$	4.0	4.0
10	r_{20}	Remove r_{22}, r_{23}, r_{24} , Add r_{20-a}	$(r_{18} \bowtie r_{19}) \rightarrow r_{20}$	3.0	3.0

Table 2.2: Rule Enforcement and Planning Results

Table 4.1 shows query plans for 10 different cases. Note that in several cases, we are showing the enforcement of the same rule but in the presence of changes to some other (usually lower level) rules as listed. For example, the first 3 rows of the table consider enforcement of Rule r_{12} . The changes listed are all relative to the first row which uses the original set of rules in Table 4.2. The first 3 cases involve a 4-way join, next 4 have 3-way join, and then one 2-way and two one-way joins. The table lists the optimal cost and the cost obtained by the algorithm. The optimal cost was obtained by an exhaustive analysis of all possible situations.

It is clear from the results that the algorithm provides the optimal results in all cases, which means that: (a) the algorithm was able to find total enforcement of rules in all cases, and (b) the enforcement cost was optimal. This is true not only for the 10 cases shown in Table 4.2, but for all 190 rule enforcements across these variants.

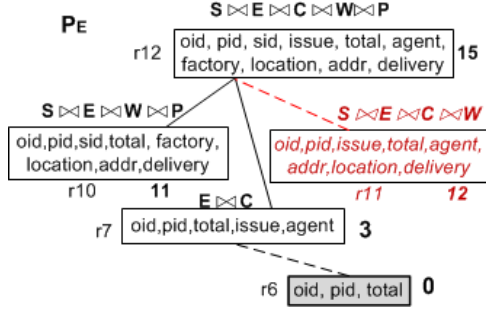


Figure 2.4: Illustration New vs. Old Algorithm (1)

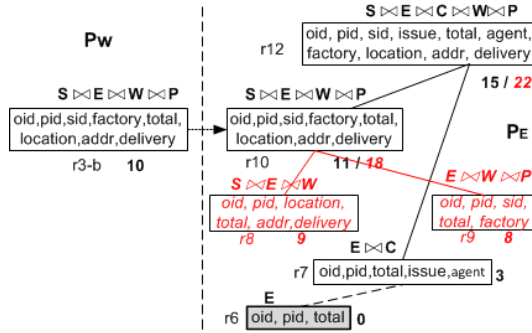


Figure 2.5: Illustration New vs. Old Algorithm (2)

In all cases, the algorithm runs in less than 0.1 seconds.

We note that our current algorithm is more thorough than the algorithm in Le et al. (2014). That algorithm only considers rules at level $n - 1$ (instead of levels $n - 1$ and $n2$). It also does not try to check for a lower cost enforcement through semijoins following the total enforcement at level n . To illustrate the impact of these differences, consider the enforcement of rule r_{12} (Case 2 of Table 4.1). One difference is shown in Fig. 2.4. Whereas the algorithm in Le et al. (2014) would choose r_{11} (shown in red italics) and join it with r_{10} , our current algorithm joins r_{10} with r_7 and thereby gets the much lower (and optimal) cost of 15. The second difference is shown in Fig 2.5. At first both algorithms will enforce r_{10} by using r_8 and r_9 (shown in red italics) with a cost of 18. However, our present algorithm will then look around and realize that it can instead enforce r_{10} at the cost of 11 by importing r_{r3-b} from party P_w .

2.5 Query Planning with a Third Parties

It was discussed earlier that some rules may be unenforceable. As discussed in section 2.3.2 there are 3 situations with respect to a rule: (a) The rule is *totally enforceable* by the regular parties, (b) The rule is only *partially enforceable*, meaning that it is possible to generate the desired join path but some attributes on this join path remain unavailable, and (c) The rule is *unenforceable* in that even the desired join path cannot be generated. In Le et al. (2013a) we devised a scheme to minimally enhance the rules with additional attributes so as to make partially enforceable rules totally enforceable. We assume here that such a procedure is already used so that there are no partially enforceable rules. Although in theory one could add additional rules to handle unenforceability as well, this is highly undesirable as it may violate the privacy policies of the regular parties. Instead, we examine the use of trusted third parties to provide the additional accesses necessary for enforcing the unenforceable rules.

It is possible to have many models for TPs, largely depending on the trust issues. The simplest model is a single TP that is trusted by all parties and thus acts as a conduit for doing the joins that cannot be done otherwise. Such a service is intended to be free from any side effects – it simply takes the input relations, returns the join, and then erases everything. If the TP is considered “honest but curious”, it is possible to encrypt the fields – perhaps using an order preserving encryption mechanism – but this does not materially affect the mechanisms explored here.

A more complex model is that of multiple TPs such that each of them is trusted by some (but not all) regular parties. In either case, the TPs may be either used only for enforcing unenforceable rules (the original intent), or given an expanded role where some otherwise enforceable rules may be enforced through TP because the latter can do so more cheaply. We will examine both of these scenarios. TPs can be involved even more deeply such as being instructed or allowed to cache data to reduce data

transfer overhead, although we do not address such scenarios in this chapter.

2.5.1 Minimum Cost Enforcement Through Join Service

Consider a target rule r_t that is unenforceable among regular parties and we use a third party (TP) join service to enforce it. We assume that this join service is trusted by all the regular parties. This generally leads to several ways for enforcing a rule. Thus the goal is to find a minimal cost enforcement for r_t assuming that the cost of the join service is proportional to the amount of data that it needs to work with. Such a minimal cost enforcement can also be used for any query authorized by r_t and hence for generating a minimal cost query plan.

In order to enforce r_t , the TP will receive suitable relations from certain regular parties. Such a relations obtained from a regular party, say r_p , could be either the basic relation that r_p owns, or a result of an enforceable rule given to r_p . In either case, we can think of the TP receiving a suitable set of attributes of an enforceable “rule” of r_p .

Suppose that the join path of the target rule r_t involves $K - 1$ joins (for some $K > 1$) among basic relations. Then depending on which joins are already reflected in the lower level rules received by the TP, it may perform $K - 1$ or fewer joins. As a simple example, if the TP needs to obtain $R \bowtie S \bowtie T$, it may obtain 3 rules with individual relations R , S , and T and do a 3-way join. As another possibility, it may obtain $R \bowtie S$ and a rule containing T (such as T , $R \bowtie T$, or $S \bowtie T$), and do a single join to get $R \bowtie S \bowtie T$. In order to avoid unnecessary use of TP, we will only send highest level enforced rules to the TP. There may still be many choices depending on the attributes that r_t is trying obtain over the desired join path, and we will base the choice on the cost considerations discussed next.

Let r_1, r_2, \dots, r_k for some $k \leq K$ denote the rules sent to TP for a $k - 1$ way join. Let J_i denote the join path for rule r_i , $i = 1..k$. (If r_i involves a basic relation,

the join path is null.) Then the cost for rule r_i can be computed as $w(J_i) * \pi(r_i)$, where $w(J_i)$ is the number of tuples in join path J_i and $\pi(r_i)$ is the per tuple cost of choosing rule r_i . If J_i is null, the number of tuples (in the basic relation) are known; otherwise, we need to estimate them. As discussed above, we assume join selectivity is known when generating the query plans. Thus $w(J_i)$ is a known quantity.

In selecting the rule attributes to send to the TP, we need to avoid duplications since rules will generally have several common attributes. However, when to avoid duplicates is a bit complicated. Consider two relations R and S available at parties P_1 and P_2 that are to be sent to TP for join over the attribute set $R.A$ and $R.B$ (where we use the dot notation to identify the relation to which the attribute set belongs). Here R and S may be either individual relations in the original SDBs, or those stored during our hierarchical rule enforcement process. First, it is clear that both $R.A$ and $R.B$ must be transmitted to TP – duplicate removal does not apply to join attributes. Now, suppose that R and S have corresponding attributes $R.B$ and $S.B$ that are not join attributes. If R and S are relations in the original SDBs, the two attributes need not be identical and thus we cannot consider one as duplicate. This holds in spite of Assumption #6 – ensuring identity would require additional declaration and coordination among parties. However, R and S are generated relations in the process of enforcement, $R.B$ and $S.B$ could be truly identical since they originate from the same party.

Thus, in general, the per tuple cost $\pi(r_i)$ is not constant, but is governed by whether duplicate removal should be done or not. Let U denote the attribute set for the target rule and S_i the set of attributes in rule r_i . Since we are selecting the rules in a specific sequence, by the time we get to rule r_i only the attribute set $T_i = U \setminus \bigcup_{j=1}^{i-1} S_j$ are left to be covered. Thus we can get the set $S_i \cap T_i$ from S_i except for those attributes that must be duplicated. We represent the latter as $Dup(S_i)$, and this includes the key of S_i and possibly others. Therefore, $\pi(r_i)$ can be defined

as follows:

$$\pi(S_i) = |S_i \cap (U \setminus \bigcup_{j=1}^{i-1} S_j)| + |Dup(S_i) \cap (\bigcup_{j=1}^{i-1} S_j)| \quad (2.5-1)$$

With this, the overall cost of a particular sequence of data transfers to TP is given by $cost(C) = \sum_{i=1}^k w(S_i)\pi(S_i)$.

Algorithm 2 Selecting Minimal Relevant Data For TP

Require: The set R of candidate rules of r_t on cooperative parties

Ensure: Find minimal amount of data being sent to TP to enforce r_t , the minimal cost and the rules to choose.

- 1: **for** Each candidate rule $r_i \in R$ **do**
 - 2: Do projection on r_i according to the attributes in r_t
 - 3: Assign r_i with its estimated number of tuples $w(S_i)$
 - 4: The set of selected rules $C \leftarrow \emptyset$
 - 5: $Cost(C) \leftarrow 0$
 - 6: Target attribute set $U \leftarrow$ merged attribute set of r_t
 - 7: **while** $U \neq \emptyset$ **do**
 - 8: Find a rule $r_i \in R$ that minimizes $\alpha = \frac{w(S_i)*\pi(S_i)}{|S_i \cap (U \setminus \bigcup_{j=1}^{i-1} S_j)|}$
 - 9: $R \leftarrow R \setminus r_i$
 - 10: $U \leftarrow U \setminus S_i$
 - 11: $cost(C) += \pi(S_i) * w(S_i)$
 - 12: $C \leftarrow C \cup r_i$
 - 13: **Return** C and $Cost(C)$
-

Let us now consider the complexity of the problem. Note that our goal is to choose rules so as to cover all required attributes at minimal cost. This is similar to the well-known weighted set covering problem where each rule can cover certain subset of the required attributes and has a certain cost. Such a problem is known to be NP hard, and the best known greedy algorithm finds the most effective subset by calculating the number of missing attributes it contributes divided by the cost of the subset. That is the heuristic algorithm selects the subset S_i using the one with minimal $\frac{w(S_i)}{|S_i \setminus U|}$. Accordingly, we select the rule with the minimal value of $\frac{w(S_i)*\pi(S_i)}{|S_i \cap (U \setminus \bigcup_{j=1}^{i-1} S_j)|}$, where $\pi(S_i)$ is defined in equation (2.5-1).

In our problem, with one more rule selected, the TP needs to perform one more

join operation, and possibly one more join attribute needs to be transferred to the TP. Therefore, when selecting a candidate rule, we examine the number of attributes this rule can provide and the costs of retrieving these attributes. Note that each time we select a new rule, we need to retrieve the key attribute set to do the join (in addition to the required attributes). Since this increases cost, the algorithm prefers rules providing more attributes and results in fewer selected rules which is consistent with our goal. We present our *Greedy Algorithm* in Algorithm 2.

We now show some results obtained via simulations. Fig. 2.6 show the relationship between the total number of rules and the number of top level relevant rules that any algorithm would need to consider for enforcement. In the figure, legend “len7,node12” means the target join path length is 7 and there are 12 cooperative parties in total. The most important result is that the number of relevant rules remains rather limited even if the total number of rules grows substantially. The significance of this result is that the complexity of the enforcement is primarily governed by the number of relevant rules, and thus remains constrained. Significantly, this result holds whether or not third parties are used for enforcement. Nevertheless, since the BruteForce algorithm is exponential, the Greedy algorithm presented here is still required. Fig. 2.7 shows this result. While the complexity of greedy algorithm does not change much, the BruteForce algorithm takes off beyond about 15 top level relevant rules.

In order to compare the performance of Greedy and BruteForce algorithms, we considered a join schema with 8 parties. The number of tuples in a rule is defined as a function of the join path length, basically $w(J_i) = 1024/2^{\text{length}(J_i)}$. In other words, we assume as the join path length increases by one, the number of tuples in the results decreases by half. We tested with randomly generated target rules with join path length of 3 and 6. Fig. 2.8 shows the comparison between two algorithms with respect to communications costs. It is seen that the two algorithms generate almost identical results. In Fig. 2.8, the legends of “BruteForce4” and “Greedy4” indicate the target

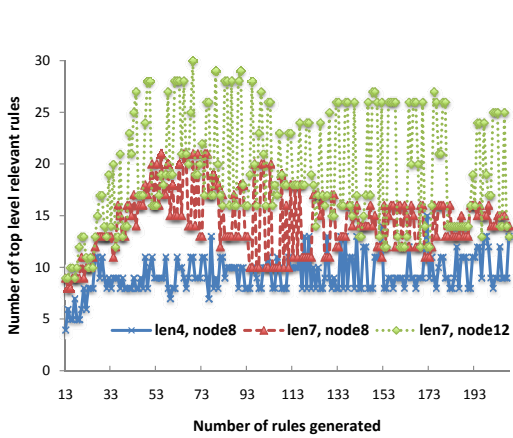


Figure 2.6: Numer of candidate rules

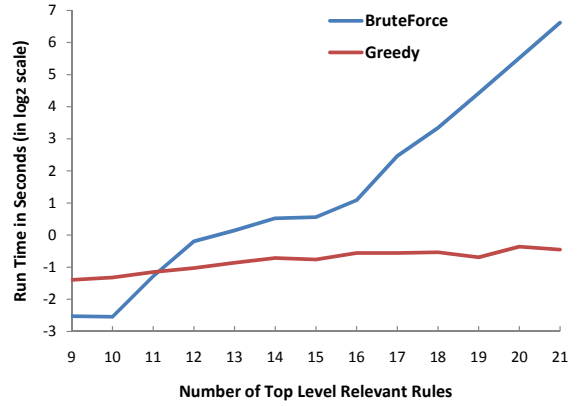


Figure 2.7: Time comparison between two algorithms

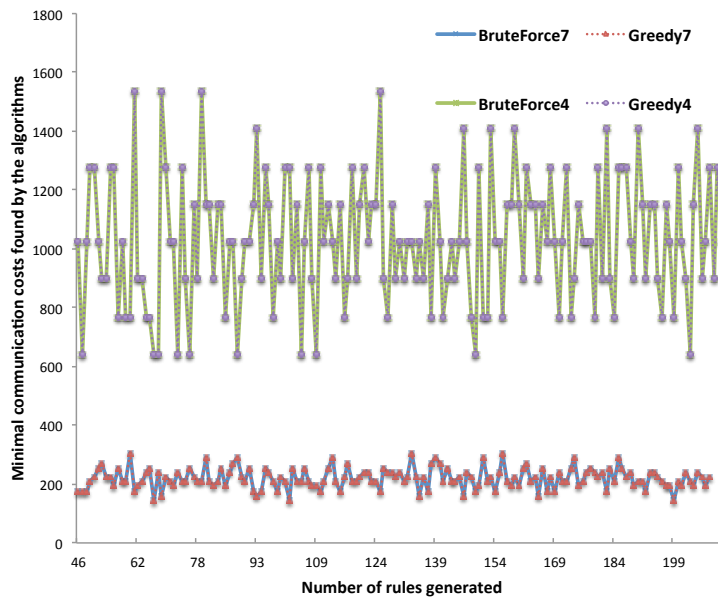


Figure 2.8: Minimal communication costs found by two algorithms

rule has the join path length of 3 for the two algorithms. Similarly, “BruteForce7” and “Greedy7” indicate join path length of 6. Among all these solutions, in less than 2% of the cases the two algorithms produce different answers. In addition, the maximal difference between them is just 5%. This shows the high quality of results achieved by the greedy algorithm.

2.5.2 Enforcement with Multiple Third Parties

There are several reasons why considering multiple TPs may be sensible. First, some regular parties may have established working relationship with certain TPs and thus prefer to involve them. (Obviously, each regular party introducing its own TPs is not helpful; instead, we want much fewer TP's than regular parties, with the latter trusting 2 or more TPs.) The second reason for multiple TPs is risk reduction – allowing a single TP to access data from many regular parties makes the TP a good target for hacking. The third reason may be related to proximity and cost considerations. Transferring large amounts of data across geographic regions may be slow and undesirable, and different TPs may have different cost structures.

We assume that we have K TP's, denoted TP_i , $i = 1..K$. We denote by $\Gamma(TP_i)$ the set of regular parties that trust it. We assume that $\Gamma(TP_i)$'s for all i are distinct, since if two TP's are trusted by exactly the same set of regular parties, we simply use only one of them. We also assume that $|\Gamma(TP_i)| > 1$ for all i , since a TP that is trusted by only one regular party is useless. Finally, we assume that every regular party trusts at least one TP_i . Even with these restrictions, it is possible that some TP_i are superfluous and not needed, and/or all TP_i collectively are still unable to enforce all the rules. Thus, there are two related problems to solve: (a) Check if the given set of TP's are adequate to enforce all the rules, and if so, (b) Find minimum cost enforcement plans for all rules. As in Section 2.4.2, we will use a combined algorithm for both.

In Section 2.5.1, we focused on the problem of choosing the highest level rules to send so that the data transfer to TP is minimized. With multiple TPs, there are two additions issues to consider: (a) choosing among TPs if more than one TP can enforce a rule, and (b) handling situation where a single rule enforcement requires multiple TPs. One way to address Problem (a) is to check for each TP if it can enforce the rule, and then use the cost minimization from the previous section. Problem (b)

TP cost	#TP	cost(4P)	cost(5P)
0.5	0	11.0	20.0
0.5	1	8.0	14.0
0.5	2	8.0	14.0
1.0	0	11.0	20.0
1.0	1	11.0	19.0
1.0	2	11.0	19.0
2.0	0	11.0	20.0
2.0	1	11.0	20.0
2.0	1	11.0	20.0

Table 2.3: Auth. rules for running Example

really does not arise as stated, since it is generally undesirable to allow a direct TP to TP communication. Instead, if a TP should simply return the enforced rule result to the regular party (or parties) that are given access to it. Thus, if these results are needed by another TP to enforce a higher level rule, that TP will still get the rule from a regular party. In other words, if the rules are enforced in the join path length order (as they are), there is no question of going through more than one TP.

If each TP is trusted by only a few regular parties, case (a) is also unlikely. In fact, if several TPs can enforce a rule, that probably implies that we are being indiscriminate about the use of TPs. Thus, instead of extending the TP handling approach of section 2.5.1 to multiple TPs, we use a different approach. First, we only consider a simple operation count based cost metric as discussed in section 2.4.1. Second, instead of first enforcing all rules that can be enforced without TPs, we consider minimal cost enforcement with TPs also included among the existing parties. Although the TPs are algorithmically treated similar to regular parties in this case, they do not own any data and their costs could be quite different from those for regular parties.

Our algorithm for query planning with multiple TPs starts out by including all TPs and regular parties. The rules given to TP_i are all the rules given to the regular parties in the set $\Gamma(TP_i)$. We then run a minimum cost enforcement algorithm of

section 2.4.2. If the TP cost is zero, our algorithm can handle it as well.

Table 2.3 shows sample results for our running example with 4 parties only (i.e., Party P removed) and with all 5 parties. The unit cost of regular party join/data transfer is 1, but the unit cost for the TP is chosen as 0.5, 1.0 and 2.0. The unit cost of 0.5 would give preference to TP and cost of 2.0 would avoid TPs. The total cost reported is the minimum cost for enforcing the highest level rule. The number of third parties is assumed to be 0, 1, or 2. The rules are chosen such that it is possible to enforce them without any third party, but the third party may help as shown in the results. In this example, more than one TP does not help in the 4 party or 5 party case. In general, how much multiple TP's help depends on how many parties trust them. One would expect that more TP's would help up to a point, and any additional TPs will only increase the cost. This is because with many TPs, we are likely to have more fragmented trust relationships, and hence it takes more steps to enforce the rules. The higher cost may still be desirable since in these situations we are accommodating the interests of regular parties in terms of using the parties that they can trust. Also, in Table 2.3, a TP cost of 2.0 does not help at all. The total cost in last 3 rows remains 20.0 since the optimal enforcement does not use any TP's at all.

In the multiple TP model, we have implicitly assumed that if a two regular parties P_1 and P_2 trust a common TP, say T_c , they allow T_c to retrieve any data allowed by the rules of either party. It is possible to make this more granular. For example, P_1 and P_2 may *delegate* only some of their rules to T_c , but not all. This means that T_c will be able to do certain operations only. Specifically, the delegations can be done to allow enforcement of unenforceable rules, but no more. It is even possible that if two distinct unenforceable rules are to be enforced, P_1 and P_2 choose the services of two different TPs and provide them with delegations such that each TP enforces exactly one of the two rules. Such a mechanism can be used to provide Chinese Wall

like isolation between the enforcements Brewer & Nash (1989). All of these cases are easily handled by our algorithm by defining appropriate rules for TPs.

2.6 Conclusion

In this chapter we considered the problem of efficiently enforcing rules and planning queries in a collaborative database environment both without and with the help of third parties. Because of the complexity of the problem, practical algorithms must necessarily be heuristic. We presented such algorithms and showed that they can solve the problem efficiently and effectively.

In this chapter we considered rule enforcement with and without third parties. It is possible to explore further variations on this in terms of what sort of operations are outsourced to third parties and which ones are done collaboratively among the original parties. Another issue is that of materializing certain popular joins and allowing the queries to be rewritten using those. In addition to deciding which joins to materialize and how to best exploit them, we also need to consider issues of updating them and accessing them efficiently.

As stated earlier, this chapter specifically leaves out the consideration of selection operations in the access rules. This is reasonable in most cases, particularly since a party could decide to share tuples only in certain ranges. However, in general, it may be desirable to share tuples over joins of two or more relations only under certain conditions. For example, consider a party providing customer service in some province (or state) X . Then it may be useful to limit this party to the orders coming from customer belonging to state X as well. It turns out that with suitable restrictions, it is possible to extend the analysis to such cases, provided that the enforceability of the rules can still be determined at schema level.

Another issue glossed over in this chapter is the relationship between internal DBs of a party and what it chooses to expose to other parties. This aspect needs a more

careful study with respect privacy of individual party's data vs. collective benefit of sharing. In particular, if parties were free to arbitrarily withhold tuples from sharing, this may cause inconsistencies and seriously limit the value of sharing. As a trivial example, if two parties share tuples in disjoint ranges of some corresponding attribute, a join on this attribute is useless. We plan to address this issue carefully in the future. A closely related issue is to examine ways of detecting compliance of a party to its declared sharing policies.

Finally, we plan to study the cooperative relationships among enterprises in various real world scenarios, and test our mechanisms under these cases.

CHAPTER 3

GENERALIZED INTER-CLOUD STRUCTURED DATA SHARING

3.1 Introduction

The rapid deployment of cyber and cyberphysical systems has resulted in increasingly rich data repositories that are used by product and service providers to support their operations and to provide rich online services to clients based on mash-ups and analytics over data available from other collaborating parties. As a concrete example, consider the health-care ecosystem involving hospitals, insurance companies, diagnostic labs, drug companies, nursing homes, etc. In this environment, answering a query regarding payments by patients requires matching patient id from the hospital database and corresponding customer id from insurance database – effectively a “join” operation if the data is stored in the relational form. Similarly, in order to smoothly distribute products to retailers, it is necessary to have coordination and restricted data sharing across trucking companies, 3PL (3rd party logistics) operators, cold-chain suppliers, distribution center operators, etc. Numerous other examples abound, one of which, namely e-commerce, will be covered in detail in the chapter. The increasing penetration of automation and cloud computing means that the collaboration happens across private or semiprivate clouds owned by various parties, each hosting its own data. We expect the richness of services to rise rapidly in the near future, driven by the push of big data analytics across willing parties.

Although the current data sharing practices across private parties depend on

undisclosed 1-on-1 agreements between them, we have argued in our prior research that a more direct *multi-party model* offers many advantages, including less information leakage Le et al. (2013a). In our multi-party model, each party is explicitly provided a set of mutually agreed *access-rules* for accessing data either directly from another party, or one that represents a composition over data from two or more parties. For example, with relational databases, a party may be allowed access to $R \bowtie S[A, B]$, meaning, attributes A and B over the relational join of relations R and S that belong to two different parties. The explicit nature of such accesses leads to better control over sharable information and its more efficient access, as discussed in our earlier work Le et al. (2013a). The main advancement of this chapter over the prior work is two fold: (a) to present an user-friendly way of describing the accesses, and from there to automatically derive the “access rules”, and (b) to support conditional access to data in specification, access rule derivation, and enforcement of the rules.

The rest of the chapter is organized as follows. In section 3.2 we describe our multiparty model and review some key issues essential for understanding the contributions of this chapter. Section 3.3 then discusses the extended collaboration model studied in this chapter. Section 3.4 discusses details of rule derivation and Section 3.5 concerns the rule enforcement under conditional accesses. Section 5.5 presents a comprehensive evaluation of the proposed algorithms. Finally, Section 5.7 concludes the discussion.

3.2 Multiparty Model and Related Work

We consider a group of collaborating parties, each hosting their relational databases in their private clouds, but willing to work together on a mutual data access plan. Accessing data across parties can be technically challenging due to varying format and semantics, but these are not the focus of this work. Thus we proceed with the

simple assumption that each party creates a “stub” for its data for uniform access by others. For example, in order to do a “join” of records from hospital and insurance company, we need stubs to ensure that `patient_ids` and `insured_id`’s translate to the same ID for the same person. The full assumptions are discussed in our earlier work Le, Kant, Athmna, & Jajodia (2016), that defines the notion of *meaningful joins* across parties, similar in spirit to the traditional notion of lossless joins within a single party. We also assume that the parties are not malicious and will correctly provide the agreed upon data; it is surely desirable to have some trust-but-verify mechanisms in place, but that too is beyond the scope of this chapter.

As discussed above, it is often necessary to provide access to a party to data that is composed from that belonging to two or more parties (or private clouds). Here “composition” could refer to meaningful operations over relations from multiple parties, such as join, union, intersection, difference, etc. Of these, join is invariably the most important and challenging to handle operation. Other operations across parties often are not useful, and in any case, they are easy to handle and thus not discussed further here.

3.2.1 Fundamental Issues in Multiparty Models

Providing access to the results of arbitrary compositions over relations poses the following crucial problem, that we call *enforcement problem*: Suppose that a party P is given access to $R * S$ where R and S are two pieces of data (or relations in our model) from different parties and “*” is a suitable composition operator. The question is then whether $R * S$ can be enforced; i.e., whether some party has enough privileges to actually compute $R * S$ (and thereby provide it to P)? If not, P ’s supposed access to $R * S$ cannot be implemented. Enforceability can be quite difficult to check, especially if we wish to do it with minimal cost. It is easily shown to be NP-hard even in very simple settings, and thus effective heuristics are a must Le et al.

(2013a). Distributed query planning that respects such access restrictions is the key problem and novelty of our work. We have studied efficient algorithms for minimal cost query planning in Le, Kant, Athmna, & Jajodia (2016); however, our prior work made two significant simplifying assumptions that we remove in this chapter, as discussed below.

First, we assumed that the access rules do not involve any conditional data accessibility (i.e., “selections” in the language of relational algebra). This could make the access rules to be unnecessarily promiscuous but easy to handle. *Allowing conditional access is one contribution of this chapter.* Second, we assumed that the “access rules” for each party were given directly; however, it is often quite difficult for parties to formulate a coherent set of access rules. *We address this shortcoming by splitting the access specification into two parts:* (a) a set of *model queries* that a party wishes to execute on the collective data of all parties, and (b) a set of accesses that it is not allowed to gain (presumably at the behest of other parties). We call the latter as *safety properties*. Since the purpose of these is to protect sensitive information, they are considered stronger than model queries. Thus the key idea is to derive access rules automatically from the model queries by respecting the restrictions imposed by safety properties. The access rules are then used for enforcement and query planning in the presence of selection conditions.

3.2.2 Related Work

Our work on the topic was inspired by that De-Capitani, et.al. De Capitani di Vimercati et al. (2008) and di Vimercati et al. (2011) on collaborative databases; however, they do not address the difficult problem of rule enforcement. It is important to note that despite tremendous amount of work on query planning Jajodia et al. (2014), the access restrictions rule out the use of regular query optimizers, as discussed in more detail in Le, Kant, Athmna, & Jajodia (2016). There are a number of other

works on somewhat related topics, e.g., distributed query processing under protection requirements Cali & Martinenghi (2008); Li (2003), use of authorization views for fine-grained access control Rizvi et al. (2004); Z. Zhang & Mendelzon (2005), but these do not address the model considered in our research. In particular, Rizvi et al. (2004) considers query rewriting, but in our case the queries are still evaluated on the original set of relations, and hence there is no rewriting. Rewriting could work in very limited cases of access restricted but in the general model considered here, it would result in too much overhead.

As stated above, our prior work handles the key issues of rule enforcement and query planning in the multiparty environment Le et al. (2013a); Le, Kant, Athmna, & Jajodia (2016). We have also examined a crucial property called *consistency*, which means that if a party P is given access to data items R and S that can be composed meaningfully (denoted by the operation “*”), then P should also be given access to $R * S$. The reason for imposing this requirement is that there is no practical way to prevent P from doing the composition locally in this case. Efficient algorithms to ensure consistency by adding additional rules, as necessary, is explored in Le et al. (2012a). We have also has considered use of *trusted third parties* (TP) for enforcement of rules that cannot be enforced using regular parties Le et al. (2013b).

3.3 Extended Collaboration Model

In this section, we present an extended collaboration model, and we also introduce a running example for later illustration of various algorithms.

3.3.1 Model Queries and Safety Properties

The *model queries* can be considered as the most permissive form of queries that a party wishes to execute (i.e., ones with all desired attributes and most generous selection criteria) over a given *join path*. A join path is simply a sequence of mean-

ingful joins involving the desired relations and join attributes. As stated above, we primarily focus on relational join as a composition mechanism because it is the most useful and challenging. A model query (or access rule) intended for a party, say P can be specified as a SPJ (select-project-join) fragment, represented as the 4-tuple $(P, \mathcal{J}, \mathcal{A}, \mathcal{C})$ where \mathcal{J} is the desired *join path*, \mathcal{A} the authorized set of attributes, and \mathcal{C} is the conditional accessibility which we call it selection condition. We assume that the join keys are included in the authorized set \mathcal{A} so that it is possible to carry out further meaningful operations on the joined relations. If no selection condition is involved, we will represent the model query simply as $(P, \mathcal{J}, \mathcal{A})$.

Safety properties can take the following two forms:

Unconstrained: This form does not explicitly constrain the denied set of attributes by the join path; i.e., a set of attributes, say \mathcal{A}_T , is denied to the target party T on every join sequence that can retrieve them. That is:

$$\text{SP}_{\text{unc}} = \{T, \mathcal{A}_T, \mathcal{C}\}$$

Note that the set \mathcal{A}_T would typically come from two or more tables, possibly from different parties, and obtainable via some sequence of meaningful joins (though we do not specify them).

Constrained: This form is identical to that of model query. That is, \mathcal{A}_T is denied over a specific join path \mathcal{J}_T .

$$\text{SP}_{\text{cons}} = \{T, \mathcal{J}, \mathcal{A}_T, \mathcal{C}\}$$

With either form, all parties involved in \mathcal{A}_T must agree to the safety rule. The key issue is to resolve any conflicts between safety properties and model queries via a minimal “weakening” of the model queries. Following this, the safety properties are no longer needed, and the weakened model queries become like the *access rules* of our earlier model, which can be used for enforcement and query planning. Section 3.4 discusses the details of the weakening.

3.3.2 Enabling Conditional Accessibility

Allowing access to parties to entire columns is actually appropriate in many situations, since many frequently needed restrictions can be imposed through joins and projections (e.g., a hospital shares data with an insurance company for only those patients that subscribe to that insurance company). In other cases, it is adequate to share data for attribute values are in a certain range (e.g., share data only for senior citizens or children). Arbitrary selection conditions are possible, but are both difficult to handle and also generally not necessary.

Earlier we spoke of the importance of ensuring the *consistency* of access rules, and consistency implies unavoidable leakage of information. A strategic use of selection conditions can lessen such leakage. For example, if a party has access to R under (selection) condition C_1 and to S under C_2 , it can correctly compute $R \bowtie S$ only for the much stricter condition $C_1 \wedge C_2$. In other words, the individual accesses do not reveal much information about $R \bowtie S$.

We differentiate between two types of selection conditions. The first type is the *elementary selections* where the selection condition consists of comparison of attributes against constants that can be numeric, string, Boolean, etc. We denote such a condition as C_e . We regard all other types of selections as *advanced selections*, and these may include direct comparison among attributes or use of aggregate functions like sum or average. We denote these as C_a . We will focus mostly on C_e 's in this chapter and consider C_a 's only simplistically. In both cases, the condition may involve multiple terms joined by Boolean operators such as AND/OR.

3.3.3 Running Example

In this section, we introduce a small running example to illustrate the model. It consists of five parties: (a) *E-commerce*, denoted as E , is a company that sells products online, (b) *Customer_Service*, denoted C , provides customer service functions

(potentially for more than one Company), (c) *Shipping*, denoted S , provides shipping services (again, potentially to multiple companies), (d) *Warehouse*, denoted W , provides storage services, and (e) *Supplier*, denoted as P , supplies the product to the warehouse. Each party owns but one table described as follows.

1. E-commerce (order_id, product_id, total) as E
2. Customer_Service (order_id, issue, agent) as C
3. Shipping (order_id, address, delivery_type) as S
4. Warehouse (product_id, supplier_id, location) as W
5. Supplier (supplier_id, supp_name, factory) as P

The relations are self-explanatory, with underlined attributes indicating the key attributes. In the following, we use oid to denote *order_id* for short, pid for *product_id*, sid for *supplier_id*, $addr$ for address, and $delivery$ for *delivery_type*. Relations E , C , S can join over their common attribute oid ; relation E can join with W over the attribute pid , and W can join with P over sid . All of these joins are considered as “meaningful” in that the join attribute is a key attribute for at least one of the two joining relations. Table 4.2 lists all rules in form of authorized attribute set, join path and the party to which the rule is given.

3.4 Weakening of Model Queries

Before plunging into the mechanics of weakening, let us first briefly discuss its impact on the shared data accessibility. Obviously, the weakening may remove certain attributes that a party expected to be available, which means that the allowed accesses may no longer meet the business needs of this party. There is no fully automated way of addressing this, but the analysis algorithms can be used to do “what-if” analysis

Table 3.1: Model Queries for the Running Example (i indicates that both parties 1&2 have the rule)

#	Authorized attribute set	Auth. Join Path	To
1	{pid, location}	W_i	P_{W_i}
2	{oid, pid}	E_i	P_{W_i}
3	{oid, pid, location}	$E_i \bowtie_{pid} W_i$	P_{W_i}
4	{pid, location}	$W_2 \bowtie_{pid} W_1$	P_{W_2}
5	{oid, pid, total}	E_i	P_{E_i}
6	{oid, pid, total, issue}	$E_i \bowtie_{oid} C_i$	P_{E_i}
7	{oid, pid, location, total, addr}	$S_i \bowtie_{oid} E_i \bowtie_{pid} W_i$	P_{E_i}
8	{oid, pid, issue, total, agent, addr}	$S_i \bowtie_{oid} E_i \bowtie_{oid} C_i \bowtie_{pid} W_i$	P_{E_i}
9	{oid, pid, total}	$E_2 \bowtie_{oid} E_1$	P_{E_2}
10	{oid, addr, delivery}	S_i	P_{S_i}
11	{oid, pid, total}	E_i	P_{S_i}
12	{oid, pid, total, addr, delivery}	$E_i \bowtie_{oid} S_i$	P_{S_i}
13	{oid, pid, total, location}	$E_i \bowtie_{pid} W_i$	P_{S_i}
14	{oid, pid, location, total, addr, delivery}	$S_i \bowtie_{oid} E_i \bowtie_{pid} W_i$	P_{S_i}
15	{oid, addr, delivery}	$S_2 \bowtie_{oid} S_1$	P_{S_2}
16	{oid, pid}	E_i	P_{C_i}
17	{oid, issue, agent}	C_i	P_{C_i}
18	{oid, pid, issue, agent}	$E_i \bowtie_{oid} C_i$	P_{C_i}
19	{oid, pid, issue, agent, total, addr}	$S_i \bowtie_{oid} C_i \bowtie_{oid} E_i \bowtie_{pid} W_i$	P_{C_i}

so that the parties can ultimately come to some agreement over the desired queries and safety guarantees that they are willing to live with.

A more interesting consequence of weakening is that certain rules may not be enforceable; i.e., a party has access to result of a composition (join), but there is no way to actually do the join. One way to resolve this problem is to bring in one or more trusted third party that is given suitable access rights to actually compute the desired joins and supply them to authorized parties. The alternative is again to ask the parties to amend the specification. This can be assisted by guidance regarding how to change the specification (e.g., a minimal perturbation or maximal benefit changes to rules). However, a formal consideration of this is beyond the scope of this chapter.

3.4.1 Weakening with and without Selections

Let $\mathbb{Q} = \{Q_1, Q_2, \dots, Q_m\}$ denote a set of model queries, where $Q_i, i \in 1, \dots, m$ is expressed as $\{P_i^{(Q)}, J_i^{(Q)}, A_i^{(Q)}, C_i^{(Q)}\}$ following the notation given earlier. Let $\mathbb{S} = \{S_1, S_2, \dots, S_k\}$ denotes the safety properties where $S_j, j \in 1, \dots, k$ is of the form $SP_{j,unc} = \{P_j^{(S)}, A_j^{(S)}, C_j^{(S)}\}$ or $SP_{j,cons} = \{P_j^{(S)}, J_j^{(S)}, A_j^{(S)}, C_j^{(S)}\}$.

Conflict Definition: A conflict requires that a subset of tuples allowed by query Q_i is denied by the SP SP_j for a specific party. That is, if $P_i^{(Q)} = P_j^{(S)}, A_j^{(S)} \subseteq A_i^{(Q)}$, and $C_i^{(Q)} \implies C_j^{(S)}$, we have a conflict for unconstrained safety properties. For constrained safety properties, we additionally need $J_i^{(Q)} = J_j^{(S)}$.

We will disallow redundancies in safety properties. This can be enforced by ensuring the following:

1. For any pair of safety properties $S_u, S_v \in \mathbb{S}$, and $P_u^{(S)} = P_v^{(S)}, A_u^{(S)} \not\subseteq A_v^{(S)}$ and $A_v^{(S)} \not\subseteq A_u^{(S)}$ – Cannot have subset relationship to avoid redundancy.
2. If $A_1^{(S)} = \{a_1, a_2, X\}$, $A_2^{(S)} = \{a_1, a_3, X\}$, $A_3^{(S)} = \{a_2, a_3, X\}$, and $A_4^{(S)} = \{a_1, a_2, a_3, X\}$ where X is a set of attributes other than a_1, a_2, a_3 , we discard $A_4^{(S)}$ because it's a weaker safety property.

It is then easy to see that:

Lemma 3.1. *The Conflict can be removed by removing an attribute $a \in A_j^{(S)}$ from $A_i^{(Q)}$.*

Lemma 3.2. *The weakening set W contains exactly one attributes from each safety property.*

Lemma 3.3. *The problem of determining the minimal (or minimum cost, if we use an explicit cost metric) weakening set W_{\min} is NP-hard.*

Proof: *The weakening set can be considered as a set-cover for the set of safety properties. Thus, determining W_{\min} is equivalent to determining minimal set-cover.*

Algorithm 3 *ARG*: Access Rule Generation

```
1: void Model_Query_Checking(Q,S)//Q is the set of model queries, S is the set of safety
   property
2: for  $p=1$  to #parties do
3:    $Q(p)$ = Set of model queries of party  $p$ 
4:    $SP_A$  = Set of SP of party  $p$  that have attributes only;
5:    $SP_G$  = Set of SP of party  $p$  that have both joins and attributes;
6:   Query_Weakening( $Q(p)$ ,  $SP_A$ );
7:   Query_Weakening( $Q(p)$ ,  $SP_G$ );
8: void Query_Weakening(Q,SP)
9:  $Att$  // Set of attributes
10:  $L$ =NULL // Set of cost,related attribute, and number of covered SP
11:  $W$ =NULL // Set of attributes that have to be weakened from all Q
12: for each attribute  $att$  in  $Att$  do
13:    $F=0$  // Cost function for attribute  $att$ 
14:    $N=0$  // Counter for the attribute where it appears in queries
15:    $M=0$  // Counter for the attribute where it appears in safety properties
16:   for each safety property  $sp$  in  $SP$  do
17:     if  $att \in Att(sp)$  then  $M++$ ;
18:   if  $M > 0$  then
19:     for each query  $q$  in  $Q$  do
20:       if  $SP == SP_G$  then
21:         if  $JP(q) == JP(sp)$  then
22:           if  $att \in Att(q)$  then  $N++$ ;
23:         else if  $att \in Att(q)$  then  $N++$ ;
24:        $F = N/M$ ;  $l_{att}=\{F, att, M\}$ ;  $L = L \cup l_{att}$ ;
25:  $C=0$  // Number of covered safety property
26: for all  $l$  in  $L$  do
27:   if  $C < |SP|$  then
28:      $W = W \cup att(l)$  Choose a set  $l$  with minimal  $F$  cost
29:      $L = L \setminus l$ ;  $C = C + M$ 
30: if  $SP == SP_G$  then
31:   for each safety property  $sp$  in  $SP$  do
32:     for each query  $q$  in  $Q$  do
33:       if  $JP(q) == JP(sp)$  then
34:         if  $W \subset Att(q)$  then  $Att(q) \setminus W$ ;
35: else
36:   for each query  $q$  in  $Q$  do
37:     if  $W \subset Att(q)$  then  $Att(q) \setminus W$ ;
38: end
```

In view of the NP-hardness result, we develop a greedy algorithm that simultaneously checks for safety properties and weakens the given rules. The basic approach is to start with the set of attributes contained in all rules and safety properties for a particular party. Then for each attribute, i.e., $(A_i^{(Q)})$, determine the number of rules and safety properties in which it appears, say (n_i) and (m_i) , respectively. If $(m_i = 0)$, we don't consider this attribute, since it is not affected by the safety properties.

Otherwise, we calculate the cost of its removal as $f(n_i/m_i)$, where $f()$ is an identity function. After obtaining removal cost for all attributes we choose the minimal cost subset of these attributes such that all safety properties are covered. Algorithm *ARG* (Access Rule Generation) sketches the overall process.

The first function obtains a set of model queries and a sets of safety properties of both types(safety properties that have attributes only, and safety properties that have joins and attributes together) for a given party. Then, we call the next function with different parameters. The first call to weaken the model queries is for safety properties that have attributes only, and the second call is to weaken the model queries for safety properties that have joins and attributes. In the second function we generate the set of all attributes from the given model queries. The first for loop, starting at line 12, initializes a cost function and two counters for the current party; one for the attributes that appear in model queries and the other one for the attributes that appear in safety properties (see lines 14,15), and then systematically takes each attribute and determines the number of safety properties in which it appears and increases its counter (see line 17). If this attribute does not appear in any safety property we don't consider it, otherwise we determine the number of model queries in which it appears and increase its counter (see line 22 for safety properties that have joins and attributes or line 23 for safety properties that have attributes only).

After this, we calculate the cost based on an identity function (see line 24). After doing that for all attributes in the loop starting at line 26, we choose an attribute with minimal cost subset of these attributes such that all safety properties are covered using a greedy minimum set cover algorithm Young (2008). The last for loop (starting at line 31 for safety properties that have joins and attributes or line 36 for safety properties that have attributes only) is where we remove the obtained set of attribute from the previous steps. At this point we do a scan through all the model queries of the same particular party, and remove the attributes identified as part of the weakening set W

that conflict with safety property.

The key quantity in the running time of the algorithm is N_P , the number of all parties, N_{Att} , the number of all attributes for a particular party, N_Q , the number of all queries, and N_{SP} , the number of safety properties for the same party. The overall worst case complexity of the greedy algorithm is $O(N_P * N_{Att} * N_{SP} * N_Q)$.

3.4.2 Combined Safety Properties and Selections

The safety properties with selections can be specified as $\{P^{(S)}, A^{(S)}, C^{(S)}\}$ or $\{P^{(S)}, J^{(S)}, A^{(S)}, C^{(S)}\}$ where $P^{(S)}$ is the party, $J^{(S)}$ is a join path, $A^{(S)}$ a set of attributes, and $C^{(S)}$ is the selection condition where the attribute set is denied. Now, consider a model query Q_i , denoted $\{P_i^{(Q)}, J_i^{(Q)}, A_i^{(Q)}, C_i^{(Q)}\}$ that conflicts with SP S_j , denoted $\{P_j^{(S)}, A_j^{(S)}, C_j^{(S)}\}$ where $A_j^{(S)} \subseteq A_i^{(Q)}$. We resolve the conflict by splitting the model query Q_i into two rules as follows:

1. $r_{i1}:(P_i^{(Q)}, J_i^{(Q)}, A_i^{(Q)}, C_i^{(Q)} \wedge \neg C_j^{(S)})$ – part of the model query Q_i where SP S_j doesn't hold.
2. $r_{i2}:(P_i^{(Q)}, J_i^{(Q)}, A_i^{(Q)}, C_i^{(Q)} \wedge C_j^{(S)})$ – part of the model query Q_i where SP S_j holds.

For the first part where the condition doesn't hold we use the same approach as Algorithm *RES* (Rule Enforcement with Selection) which we will describe it later on, and for the second part we generate a rule using Algorithm *ARG*. We use the same approach if the model query conflict with SP that have both joins and attributes.

Algorithm *RESPS* (Rule Enforcement for SP with Selection) is a greedy algorithm that consolidates both the previous approaches. We take all rules and safety properties for a particular party, and then for each rule that has conflicted attributes with a given safety property we compare its selection condition with the selection condition on that safety property, and see whether it holds or not. If it holds, we

Algorithm 4 *RESPS*:Rule Enforcement for SP with Selection

```
1: void Rule_Enforcement_Safety_Selection( $\mathbb{Q}, \mathbb{S}, C_J, C_T$ ); //  $\mathbb{Q}$  is the set of model queries,  $\mathbb{S}$ 
   is the set of safety properties,  $C_J, C_T$  are Join data transfer costs
2:  $R_n = NULL$ ; // Set of rules where conflict doesn't hold
3:  $\mathcal{R} = NULL$ ; // Set of model queries where conflict holds
4: for  $p=1$  to #parties do
5:    $SP(p)$  = Set of safety properties of party  $p$ 
6:    $Q(p)$  = Set of model queries of party  $p$ 
7:    $Att(p)$ ; // Set of attributes for party  $p$ 
8:   for each  $sp$  in  $SP$  and each  $q$  in  $Q$  do
9:     if  $Att(sp) \subseteq Att(r)$  then
10:      if  $(C(r) \cap C(sp) \neq \phi)$  then
11:         $r_n = (Att(r), JP(r), C(r) \wedge \neg C(sp), p)$ ;  $R_n = R_n \cup r_n$ ;
12:         $r_c = (Att(r), JP(r), C(r) \wedge C(sp), p)$ ;  $\mathcal{R} = \mathcal{R} \cup r_c$ 
13:      else  $R_n = R_n \cup r$ ;
14: Call_Algorithm_(RES ( $R_n$ )); // We send all rules including the newly added rules to rule
   enforcement with selection Algorithm
15: Call_Algorithm_(ARG ( $\mathcal{R}$ )); // We send all conflicted rules to access rule generation Algorithm
16: end
```

generate a new rule with selection condition outside the boundary of the original rule and we put it in a set R_n . Then we put the original rule in another set called \mathcal{R} , which has all the rules that are ready to be weakened. If it doesn't hold, we also put it in a set R_n . Finally we enforce the rules in R_n using Algorithm *RES*, and we weaken the rules in \mathcal{R} using Algorithm *ARG*.

The key quantity in the running time of the algorithm *RESPS* is N_p , the number of parties in with they have rules, N_{sp} , the number of safety properties for the same party, and N_R the number of all rules for that party. The overall worst case complexity of the greedy algorithm is $O(N_P * N_{SP} * N_R)$.

3.5 Rule Enforcement under Conditional Access

A minimum cost enforcement of rules is essential for query planning. To see this, consider an authorized query, q , issued by a party. It could differ from the access rules (which are really model queries suitably modified to remove conflicts with safety properties) in two ways: (a) it may involve several rules along with further operations (e.g., union, intersection, etc.) over their results, and (b) it may request

fewer attributes or use more restrictive selection conditions (e.g., smaller ranges) than the access rule permits. In either case, once we have a mechanism for efficient enforcement of all rules, it is easy to apply them to actual queries. Note that because of (b), it may still be useful to explicitly plan individual queries as done in Le et al. (2014), however, this can be rather inefficient. Instead, the approach used in Le, Kant, Athmna, & Jajodia (2016) is to compute the enforcement plan for each access rule in advance, and then simply strip it down depending on unneeded attributes in the actual query. We will use this approach here as well and point out the changes required to accommodate selections.

Regardless of how the enforcement (or query planning) is done, the resulting enforcement/query plan pl is a sequence of operations that involve suitable SPJ operations and transfer of partial results across parties. We say that two join paths $J_i^{(Q)}$ and $J_j^{(Q)}$ are equivalent, or $J_i^{(Q)} \cong J_j^{(Q)}$, if $J_j^{(Q)}$ is a valid permutation of $J_i^{(Q)}$. A target rule $r_t = \{J_t^{(Q)}, A_t^{(Q)}\}$ can be *totally enforced*, if there exists a plan pl such that $J_t^{(Q)} \cong J_{pl}^{(Q)}$, $A_t^{(Q)} = A_{pl}^{(Q)}$, $P_t^{(Q)} = P_{pl}^{(Q)}$ r_t is *partially enforceable*. If it is not totally enforceable and there is a plan pl that $J_t^{(Q)} \cong J_{pl}^{(Q)}$, $A_t^{(Q)} \supset A_{pl}^{(Q)}$, $P_t^{(Q)} = P_{pl}^{(Q)}$. Otherwise, r_t is not *enforceable*. A join path $J_t^{(Q)}$ is enforceable if there is a plan pl that $J_t^{(Q)} \cong J_{pl}^{(Q)}$.

Let us start with elementary (range) selections. We shall restrict selections over join attributes – selections over other attributes are possible but difficult to handle in general. On a given join path, there can be several rules with different selection conditions. As an example, suppose that we have the following two authorization rules:

$$[P_{E_i}, E \bowtie_{oid} C, \{oid, total, issue, agent\}, pid < 200]$$

$$[P_{E_i}, E \bowtie_{oid} C, \{oid, pid, issue, agent\}, oid < 100]$$

In this case it suffices to consider a single rule:

$$[P_{E_i}, E \bowtie_{oid} C, \{total, issue, agent\}, pid < 200 \text{ or } oid < 100]$$

That is, when multiple rules are defined on the same join path, we can first merge their range selection conditions $C_e = (C_{e1}UC_{e2}\dots UC_{em})$ on the same join path into one statement. Then during the enforcement checking, while composing the lower level join paths into the join path of such a given rule, we need intersection of conditions of constituent rules. The functional dependencies (FDs) are then used to ensure that the conditions are updated correctly for all the attributes. Finally, we will get only one enforceable selection condition of the rule on one join path. So we can still maintain one rule at a join path.

Similarly, the advanced selection conditions on a particular join path can be merged together into one as $C_a = (C_{a1}UC_{a2}\dots UC_{am})$. Therefore, there are at most 2 rules on one join path, and a rule can be represented as $[P_r, J_r, A_r, C_e]$ and $[P_r, J_r, A_r, C_a]$.

An advantage of the elementary selections on join attributes is that we can work with them at a purely symbolic level in checking enforceability and query planning. For example, given two such ranges over some attribute, we can easily determine if one is contained within the other, or more generally determine their intersection. This is not possible for advanced selections in general. Therefore, when checking the rule enforceability from bottom up, only the elementary selection conditions can be used as *relevant rules* to evaluate the enforceability of the rules with longer join paths. Advanced selections can be allowed one time as local operations, since the selection can be assumed to create a new relation (or view) which does not involve any further selections.

In general, it is possible to allow for more general symbolic evaluation than just the ranges. For example, if party Q can access R_i with condition $A_1 < A_2$, and R_j with condition $A_1 < A_2/2$, then we know that if Q were given the rule $R_i \bowtie R_j$ with condition $A_1 < A_2/3$, it would be enforceable. Given a basic enforcement mechanism, such extensions are themselves straightforward – the challenging part is

properly defining the space of constraints that allow for purely symbolic reasoning.

Our minimum cost rule enforcement algorithm, reported in Le, Kant, Athmna, & Jajodia (2016), takes all the target rules and checks for their join path enforceability. It starts with target rules (r_t) of the smallest join path length (JPL =1), and marks them on a graph as totally enforced with initialized cost. Next, it steps through each JPL in increasing order. Each target rule r_t of a JPL is then picked up and checked for local enforceability (partial or total) by the party by considering joins from relevant rules of JPL - 1 and JPL - 2 (with JPL = 1, there is no JPL - 2, of course). If the join path of r_t can be enforced, perhaps in more than one way, we choose the relevant rules r_i and r_j that can join to provide the maximum number of attributes for r_t . We add node r_t to the graph along with its updated cost and edges to r_i and r_j . It is possible that this enforcement does not provide all attributes for r_t (partial enforcement). In this case, we need to obtain missing attributes from other parties through all already enforced rules having the same join path as r_t . If there are many such rules that can provide the missing attributes, we choose the one that provides the most (a greedy approach). We add an edge to this node in the graph along with updated cost and update the missing attributes. Finally, we scan through all the totally enforced rules for current JPL, and check if any rule can be enforced with a lower cost.

This algorithm can be easily extended to consider selection conditions as well, and for brevity we will not describe the changes. Instead, we simply present the enhancement in Algorithm *RES*. When evaluating a target rule on a join path, the algorithm only checks its relevant rules with range selections rather than checking all rules. If the join path J_r is enforceable, then we check the possible local enforcement of the rule, and update the range selections on all join attributes. After checking the local enforcement, the algorithm further checks the range selection conditions from **J_t -cooperative parties** of the target rule. We define these as the set of parties that have access rules over a join path J_t (or its equivalent). These parties can exchange

their common attributes defined over J_t . For instance in our running example, P_{E_i} and P_{C_i} are J_{19} -cooperative parties since $J_{19} \cong J_8$. Basically, when combining two pieces of enforceable information on the same join path, we can take the union of selections on the key attribute of the join path first and then update the range selections on other join attributes.

As an example, suppose that there is a target rule with range selection $[P_{W_i}, E \bowtie_{pid} W, \{oid, pid, location\}, \{oid > 100, pid < 200\}]$, and there are two local relevant rules, $[P_{W_i}, E, \{oid, pid\}, \{pid < 100\}]$ and $[P_{W_i}, W, \{pid\}, \{pid > 50\}]$. There is no selection condition on the attribute “oid”, so “oid > 100” can be totally enforced. For “pid”, the effective selection condition on $E \bowtie W$ is obviously $C = pid < 100 \wedge pid > 50$ (intersection of conditions over E and W). Thus all tuples with “pid < 200” are not admissible; instead only the intersection of “pid < 200” and C is, which is C itself in this case. Notice that the condition “oid > 100” could further limit the pid values, however, unless is a direct relationship between oid and pid , we need to maintain both expressions.

After the local enforcement checking, the algorithm also checks the possible enforcement by getting information from its J_t -cooperative parties. In the above example, if there is a J_t -cooperative party that has the enforceable rule $[P_E, E \bowtie W, \{pid\}, \{oid < 200 \wedge pid > 120\}]$, then the key attribute selection of the target rule can be merged as “oid < 200” \wedge “oid > 100” $\&$ “pid > 120” $\cap C$. Then we need to update again the enforceable pid information on the join path. That can give us the final selections for each attribute.

The heuristic Algorithm *RES* performs minimal cost enforcement of all rules in the presence of selections. This algorithm is an enhancement of our algorithm in Le, Kant, Athmnaah, & Jajodia (2016); therefore, we will only point out the significant changes. This algorithm also starts with baseline rules (i.e., rules giving each party access to its own individual relations) and proceeds in the order of increasing join

Algorithm 5 *RES*:Rule Enforcement with Selection

```
1: void Rule_Enforcement_Selection( $C_J, C_T$ ); //  $C_J, C_T$  are Join data transfer costs
2: Sort rules according to join path length (JPL)
3: for  $p=1$  to #parties & each rule  $r_t$  w/  $JPL==1$  do
4:   Mark  $r_t$  as totally enforced and add to Graph;
5: for  $JPL = 2$  to  $JPL_{max}$  do
6:   for each rule  $r_t$  of this JPL do
7:     Initialize  $Att(r_t), St(r_t)$  &  $p(r_t)$ ; // Attr, sel. cond, & party of rule  $r_t$ ;
8:      $RRS(r_t) =$  Rules of length JPL-1 or JPL-2 relevant to  $r_t$ ;
9:      $A_m = Att(r_t); S_m = St(r_t)$ ; // All attr & tuples missing initially
10:    Add_node( $r_t$ ); // Add node  $r_t$  to the Graph
11:    if  $r_t$  is enforceable by  $p$  using  $r_1, r_2 \in RRS(r_t)$  then
12:      Add arcs to  $r$  from  $r_1, r_2$  and update  $r$ 's cost.
13:       $A_m = A_m \setminus [Att(r_1) + Att(r_2)]$ ;
14:       $S_m = S_m \setminus [St(r_1) \cap St(r_2)]$  // Intersection of selections
15:      if  $A_m(r) \neq \text{NULL}$  then {Add ( $r_t, A_m, S_m$ ) to Queue};
16:    for each entry [ $r_t, A_m(r_t), S_m(r_t)$ ] in Queue do
17:      att_enforced = 0;
18:      for each rule  $r_s$  with same Join_Path as  $r_t$  do
19:        if already_enforced( $r_s$ ) then
20:          if #Att( $r_s$ )  $\not\subseteq$  att_enforced where  $S_m(r_s) \Rightarrow S_m(r_t)$  then
21:            att_enforced = #Att( $r_s$ );  $r_{new} = r_s$ ;
22:            if att_enforced = #att( $r_t$ ) then {retain  $r_{new}$  w/ min cost}
23:          Add_arcs to  $r_t$  from  $r_{new}$  w/ cost  $C_T$ ;
24:           $A_m(r_t) = A_m(r_t) \setminus Att(r_{new})$ ;
25:          if  $A_m(r_t) \neq \text{NULL}$  then
26:            Retrieve the max subset of  $A_m(r_t)$  using semi-join w/ rules of Party( $r_t$ ); Update
               $A_m(r_t)$ ;
27:          if  $A_m(r_t) \neq \text{NULL}$  then
28:            Mark  $r_t$  as unenforceable and remove it from Queue;
29:            else Remove [ $r_t, A_m(r_t), S_m(r_t)$ ] from Queue ;
30:          for each entry [ $r_t, A_m(r_t), S_m(r_t)$ ] in Graph do
31:            {If  $\exists$  entry [ $r_s, A_m(r_t), S_m(r_s)$ ] s.t.  $S_m(r_s) \Rightarrow S_m(r_t)$ , choose one with lower cost;
              Update Graph;}
32: end
```

path length. Whenever an enforcement step involves a join, the selection condition for the resulting rule is the intersection of the conditions associated with the input rules. Also, when the algorithm tries to obtain missing attributes from other parties for a target rule r_t , we look through all already enforced rules in the graph that have the same join path as r_t and a selection condition that implies the selection condition for r_t .

The key quantity in the running time of the algorithm is N_{qt} , the number of rules locally relevant to rule q , and N_{qt} , the number of relevant rules on all J_t -cooperative

parties. Both of these stay small (i.e., in the range of 10's) even with large number of rules and parties. The overall worst case complexity of the greedy algorithm is $O(N_{qt} * N_{qt}^2)$.

3.6 Performance Evaluation

3.6.1 Evaluation of Access Rule Generation Algorithm (*ARG*)

Randomly generating safety properties for the given model queries in Table 4.2 is unlikely to yield meaningful constraints and will be unable to stress test the algorithm. Therefore, we instead generated 10 different variants of our running example by changing some rules, (i.e., increasing or decreasing the number of attributes allowed by the rule), and adding some safety properties. Then, we weakened the rules according to Algorithm *ARG*. Next, we used our minimum cost rule enforcement Algorithm to generate the enforcement (query) plans for the rules. We also checked the rules are *totally enforceable* or *partially enforceable* in the presence of safety properties. Incidentally these cases were specially constructed to provide sub-optimal results. Please note that the end result is affected by both heuristic algorithms (*REG* and *RES*), which could amplify the errors. Nevertheless, as we shall see, the net error remains very small.

Table 4.1 shows the results. Here and subsequent tables, we have abbreviated non-key attributes further to save space. The key is: A: addr, D: delivery, F: factory, I: issue, L: location, and T: total. Each row of the table shows one case, and the columns show respectively (a) Set of safety properties used in that case, (b) "Weakening set $W - Alg$, or the set of attributes removed from the rules by algorithm *ARG* to comply with the given safety property, (c) "Weakening set $W - Opt$ " is the set of attributes that must be removed in order to enforce all rules with maximum possible number of attributes, (d) "# Rules-Alg" is the totally enforced rules after weakening rules using

Table 3.2: Rule Enforcement and Safety Properties

#	Safety Properties	Weakening Set W		# Rules	
		Alg.	Opt	Alg	Opt
1	$[P_{S_1}, \{pid, L\}], [P_{S_1}, \{oid, T\}]$	{L, T}	{L, T}	27	27
2	$[P_{S_1}, \{pid, A, D\}], [P_{S_1}, \{oid, T, L\}]$	{L, A}	{L, D}	26	28
3	$[P_{S_1}, \{oid, pid, A\}], [P_{S_1}, \{T, D, L\}]$	{L, A}	{L, A}	27	27
4	$[P_{S_1}, \{oid, pid, T\}], [P_{S_1}, \{A, D, L\}]$	{L, pid}	{L, T}	27	27
5	$[P_{S_1}, \{pid, A, T\}], [P_{S_1}, \{oid, D, L\}]$	{L, A}	{L, A}	27	27
6	$[P_{S_1}, \{pid, A, L\}], [P_{S_1}, \{oid, D, T\}]$	{L, D}	{L, D}	29	29
7	$[P_{E_1}, \{oid, pid, L\}], [P_{E_1}, \{sid, T, F\}]$	{L, sid}	{L, F}	36	37
8	$[P_{E_1}, E_1 \bowtie_{oid} C_1, \{oid, pid, I\}], [P_{E_1}, S_1 \bowtie_{oid} E_1 \bowtie_{pid} W_1 \bowtie_{sid} P_1, \{sid, T, A\}]$	{sid, I}	{sid, I}	36	36
9	$[P_{E_1}, \{sid, pid, F\}], [P_{E_1}, S_1 \bowtie_{oid} C_1 \bowtie_{oid} E_1 \bowtie_{pid} W_1 \bowtie_{sid} P_1, \{oid, L, A\}]$	{sid, L}	{F, L}	37	38
10	$[P_{E_1}, \{pid, T, A\}], [P_{E_1}, \{oid, F, sid\}]$	{A, sid}	{A, F}	34	35

Table 3.3: Access rules with new party

Rule	R Chan	Authorized attr. set	Auth. Join Path	To
r_1	Add "sid"	{pid, sid, L}	W_1	P_{W_1}
r_3	Add "sid"	{oid, pid, sid, L}	$E_1 \bowtie_{pid} W_1$	P_{W_1}
r_{3a}	New rule	{pid, sid, F}	$W_1 \bowtie_{sid} P_1$	P_{W_1}
r_{3b}	New rule	{oid, pid, sid, L, F}	$E_1 \bowtie_{pid} W_1 \bowtie_{sid} P_1$	P_{W_1}
r_{7a}	New rule	{oid, pid, sid, T, F}	$E_1 \bowtie_{pid} W_1 \bowtie_{sid} P_1$	P_{E_1}
r_{7b}	New rule	{oid, pid, sid, T, F, L, A}	$S_1 \bowtie_{oid} E_1 \bowtie_{pid} W_1 \bowtie_{sid} P_1$	P_{E_1}
r_{8a}	New rule	{oid, pid, sid, I, T, agent, F, L, A}	$S_1 \bowtie_{oid} E_1 \bowtie_{oid} C_1 \bowtie_{pid} W_1 \bowtie_{sid} P_1$	P_{E_1}
r_{14a}	New rule	{oid, pid, sid, T, L, A, D}	$S_1 \bowtie_{oid} E_1 \bowtie_{pid} W_1 \bowtie_{sid} P_1$	P_{S_1}
r_{19a}	New rule	{sid, sname, F}	P_1	P_{P_1}
r_{19b}	New rule	{pid, sid}	W_1	P_{P_1}
r_{19c}	New rule	{pid, sname, sid, F}	$W_1 \bowtie_{sid} P_1$	P_{P_1}

algorithm ARG , and (e) "# Rules-Opt" is the optimal number of totally enforced rules by the exact algorithm.

In cases 1 to 6, we tested the weakening rules using algorithm ARG and minimum cost rule enforcement algorithm with the 35 rules in Table 4.2. For cases from 7 to 10 we added another party P_{p_i} to the previous rules in Table 4.2 and we made some changes in the rules as shown in Table 3.3. As a result we get overall 44 rules. We ensure that every safety property has at least one non-key attribute, and only the non-key attributes are removed for conflict removal.

For example, in case 1 we are given two safety properties for party P_{S_1} and the 35 rules in table 4.2 as input in algorithm ARG , and we get weakening set $W = \{L, T\}$. We can see that this is the same result in exact weakening set, meaning the two attributes are the only attributes that have the minimal cost and they cover two safety properties. After removing those two attributes from all the conflicted rules in party P_{S_1} , we run the rules through our minimal cost rule enforcement algorithm. This results in 8 rules *totally enforced*, which is the same result in optimal solution if we used the exact weakening set.

In some cases, the set W given by algorithm ARG is different from the exact weakening set, but the overall number of rules enforced without weakening their attributes are the same. This is because in such cases the rules can still be *totally enforced* through the semi-join operations Kossmann (2000b).

We can see in overall cases the algorithm was able to enforce all the rules with very low error rate (1.6%), i.e. 5 cases out of 314 totally enforced rules. If we consider all 360 rules the error is even smaller (1.4%). In effect, the 10 different sets of safety properties considered here perturbs a T of 360 rules, which represents an extensive perturbation testing of our fairly representative running example.

3.6.2 Evaluation of Enforcement with Selections

As discussed in Le, Kant, Athmna, & Jajodia (2016), the basic algorithm (without selections) does extremely well in that we were unable to generate cases where it gives nonoptimal results. Thus, we only focus on how often the algorithm will generate nonoptimal results with selections. For the evaluation, we generated different rules sets by changing some rules and adding some selection conditions for rules in Table 4.2. Then we considered the algorithm RES and selection conditions on three attributes listed in Table 3.4. We checked the enforceability of the rules and determined whether other rules are *Totally enforceable* or *partially enforceable* given

Table 3.4: Auth. rules with selection condition on three attributes

Rule	change	Sel Condition	Authorized att set	Auth Path	Join	To
r_{7a}	Add cond	$oid > 50 \wedge sid < 1000 \wedge pid > 100$	{oid, pid, sid, T, F}	E_1 $W_1 \bowtie_{sid} P_1$	\bowtie_{pid}	P_{E_1}
r_{13}	Add cond	$oid < 200 \wedge pid > 100 \wedge sid < 700$	{oid, pid, T, L}	$E_1 \bowtie_{pid} W_1$		P_{S_1}

Table 3.5: Rule enforcement with selection

#	Rule	Enforcement Plan	Enforced Tuples	obs cost	opt cost
1	r_7	$[P_{S_i} \xrightarrow{r_7} P_{E_i}] \bowtie r_5$	$oid < 200 \wedge pid > 100$	7.0	7.0
2	r_{7b}	$r_7 \bowtie r_{7a}$	$50 < oid < 200 \wedge sid < 1000 \wedge pid > 100$	15.0	15.0
3	r_8	$r_6 \bowtie r_7$	$oid < 200 \wedge pid > 100$	11.0	11.0
4	r_{8a}	$r_8 \bowtie r_{7a}$	$50 < oid < 200 \wedge sid < 1000 \wedge pid > 100$	19.0	19.0
5	r_{14}	$r_{13} \bowtie r_{12}$	$oid < 200 \wedge pid > 100$	6.0	6.0
6	r_{19}	$[P_{E_i} \xrightarrow{r_{19}} P_{C_i}] \bowtie r_{18}$	$oid < 200 \wedge pid > 100$	12.0	12.0

the selection conditions.

Table 3.5 shows the result for 6 different variants of the rule set. Each row of the table shows one such case, and the columns are:(a) *Rule* used in that case, (b) *Enforcement plan* for that rule, (c) *Enforced tuples* is the applicable tuples implied from selection condition on enforcement plan, (d) *Cost* is the cost for this rule enforcement, and (e) *Optimal cost* is the optimal cost in which this rule will be enforced. In the enforcement plan the operation $[P_X \xrightarrow{r_j} P_Y]$ means that result of rule r_j is transferred from party P_X to party P_Y .

We see in these 6 cases the algorithm determined cost and optimal cost are equal. We generated many other cases with different number of attributes on selection conditions(10 cases for single attribute selections, 10 cases for two attributes selections, and 6 cases for three attributes selections). We found that in all cases, our algorithm gave optimal results both in terms of cost and the conditions enforced.

Table 3.6: Rule enforcement time (in secs) in Amazon Cloud

#	Rule	Plan	Description	Time
1	r_3	$r_1 \bowtie r_2$	$P_{E_1} \xrightarrow{r_3} P_{W_1}$, join with r_1 at P_{W_1}	251
2	r_3	$r_1 \bowtie r_2$	$P_{E_1} \xrightarrow{r_3} P_{W_1}$, semi-join with r_1 at P_{W_1}	216
3	r_{13}	$P_{W_1} \xrightarrow{r_{13}} P_{S_1}$	P_{W_1} does the join, sends the result to P_{S_1}	270
4	r_{13}	$P_{W_1} \xrightarrow{r_{13}} P_{S_1}$	P_{W_1} does the semi-join, sends result to P_{S_1}	243
5	r_7	$P_{S_1} \xrightarrow{r_7} P_{E_1}$	P_{S_1} does the join, sends the results to P_{E_1}	761
6	r_{14}	$r_{12} \bowtie r_{13}$	$P_{W_1} \xrightarrow{r_{13}} P_{S_1}$, join with r_{12} at P_{S_1}	693
7	r_{14}	$r_{12} \bowtie r_{13}$	$P_{W_1} \xrightarrow{r_{13}} P_{S_1}$, semi-join with r_{12} at P_{S_1}	726

3.6.3 Evaluation in a Real Cloud Environment

We experimented with our multiparty model in a real cloud environment using Amazon Web Services (AWS). Each party was allocated to a different server, hosted one normalized relation, and used Amazon RDS (Relational Database Service) to perform relational operations. The parties also needed to exchange data in order to execute the query plans. Table 3.6 shows the results for 7 cases of enforcements of rules. Each row of the table shows one case. The "rule" is the rule being enforced, the "Plan" is its enforcement plan and the "Time" is the measured enforcement time in secs. The operation $P_X \xrightarrow{r_j} P_Y$ means that result of rule r_j is transferred from party P_X to party P_Y . One point to note from the numbers is that different methods for enforcing a rule have different costs. Another point is that a systematic enforcement of our multiparty model allows intermediate results to be obtained by other authorized parties directly. For example, although row 3 includes cost of computing $r_1 \bowtie r_2$, party P_{S_1} could simply get it from party P_{W_1} .

3.7 Conclusion

In this chapter we considered a comprehensive model for collaborative data sharing among cooperating private clouds and examined the algorithmic issues of deriving access rules from higher level specifications involving tuple selections and the enforce-

ment of these rules. We showed that our algorithms can solve the problem efficiently and accurately in spite of its high complexity. In the future, we will consider various other models of data sharing via clouds and their impact on performance.

CHAPTER 4

COLLABORATIVE SIMILARITY SEARCH ACROSS MULTI-PARTY REPOSITORIES

4.1 Introduction

The rapidly increasing penetration of information technology in all areas of cyber and cyber-physical arenas, has led to increasingly rich data repositories that are used by producers and service providers to support highly optimized operations. There are at least two dominant trends in this regard, each of which poses its unique computing challenges. The first is the increasing amount and granularity of available data and analytics that drives richer queries and augmented services based on relationships between products, services, sensors, people, etc. The challenges here relate to the quality of data, and corresponding issues of speed and accuracy of online analytics using this data. The second trend is the increasing breadth of data available across different parties, which can be exploited for much more semantically rich queries and services. The challenges here relate to the thorny problems of interoperable data formats, schemas, access restrictions on data across parties, security/privacy issues, etc.

Access to cross-party data is becoming increasingly important not only because of what we could do with it, but also due to the increasing replacement of the vertical business model of the past by complex supply chains involving many vendors/providers that must collaborate in order to provide the same level of services as in a vertically integrated model. The situations we consider, are those where the

attributes of the entities are coarse or perhaps aggregated and not subject to large changes in a small time period.

A canonical example of collaboration is in the supply chain where parties such as manufacturers, shippers, retailers, etc. need to share information to function smoothly. Typically, the shared information would involve a composition (or “join” in relational terms) of information by doing key matching from two or more parties. For example, shipping needs a match on a manufacturers from shipper’s DB and retailers’ DB to get the respective addresses and details on order size and shipment requirements. A few other situations requiring collaboration are product distribution logistics Kant & Pal (2017) by 3rd parties (e.g., 3PL), power utilities operating in a region, etc.

In addition to the clear application to the supply chain, collaborative similarity search has application to domains where it may be common to have the elements of the tuples generated by different processes or stored away from their point of generation. This is a common scenario in Internet of Things (IoT) Gubbi et al. (2013) environments, where the nodes generating the data is usually not the nodes analyzing the data. For example in an Iot-based emergency and disaster relief system Ben Arbia et al. (2017), important consideration must be given to how wearable devices and other sensors can send collected information to be analyzed. A component of an alternative architecture may involve storing elements of tuples closer to the generation of data so that large amounts of data don’t have to be routed to a command center for analysis under unreliable network conditions. Under such an architecture if the analysis required searching for the queried tuples with similar values then the actual queries can be done at some edge nodes before the results are collected for further analysis.

Another domain that will see IoT related innovations is crime detection Byun et al. (2014). With expected pervasiveness of wearable devices Metcalf et al. (2016),

there may be many devices and a variety of sensors near an ongoing crime or other emergency. Vital information can be extracted from the sensor data if the query can be targeted to devices that may be relevant to an event. In this case, it is not desirable to return all the data to a central location and then querying it. Rather a similarity search can be performed on a node closer to point where the data is generated, restricting the similarity search to the relevant types of sensors. The different sensor data of a person don't need to be collected on one node before performing the search. Similar architectural patterns can be applied to other domains that will be experience an increase in IoT related innovations such as Smart Healthcare System (SHS) Gope & Hwang (2016), Smartgrids Sagiroglu et al. (2016), and Smart homes Mano et al. (2016).

There are many instances where a party may simply put out some data for potential use by others instead of collaborating explicitly. Correlating such data to derive intelligence from these sources is increasingly of interest as data mining techniques advance. There is, in fact, an effort called LOD (Linked Open Data) Sikos (2015) to cater to such needs. LOD represents data as triples (URI, property, value) in RDF format and makes it easy to take data from spreadsheets, configuration files, and such and easily allow for its participation in collaborative queries. For specificity, we do not consider such models in this chapter. .

Irrespective of the number of parties involved, it is often desirable to provide do some further processing based on “similarity”. This is a common need in the e-tailer industry. These may be similarity of products (e.g., products in the same price range or similar features), or similarity of customers (e.g., those living in the same city, or those that are friends of a given customer). For example, such processing can be used for additional purchase recommendations, or for further limiting the set of items returned as results. Ideally, we would like to do this in real-time so that it is possible to make them an integral part of the query capabilities. *To the best of our*

knowledge, ours is the first attempt to address this problem in general terms for any environment, not just e-commerce. It may be noted that many e-tailers do provide purchase recommendations, but those are based on accumulated user ratings and choices.

The overall goal of the chapter is to combine the above two aspects, i.e., support queries across multi-party relational databases that can also provide auxiliary information based on similarity between the tuples of a database table. Such similarity is best represented by a graph where tuples are nodes and edges (potentially weighted) represent similarities. We devise an efficient similarity search algorithm that can be further enhanced by making use of dominating attributes that often occur in real data. We also show how the algorithm can be applied to the collaborative database environments.

The rest of the chapter is organized as follows. Section 4.2 discusses the background and the challenges in solving the problem. Section 4.3 discusses the related work. Section 4.4 proposes an efficient solution to the similarity search problem and shows how it can be enhanced further by using dominating attributes. Section 4.5 then provides an evaluation of the scheme and section 5.7 concludes the discussion.

4.2 Background

In this section, we provide a brief background on our multiparty collaborative data sharing model and its extension with similarity searches.

4.2.1 Multiparty Data Sharing Model

Our multiparty data sharing model goes beyond the currently prevalent 1-on-1 sharing model, where each pair of parties that wish to share data execute a non-public agreement as to what attributes (for example, relational columns) of one party's database will be shared with the other. In contrast, our multiparty sharing model

involves an explicit and globally visible (at least on a need-to-know basis) set of “rules” that specify what accesses each party has on individual data and compositions (for example, relational joins) of data by doing key matching from two or more parties . We have shown that such a specification actually needs to expose less information than a 1-on-1 sharing model Le et al. (2013a); Le, Kant, Athmnah, & Jajodia (2016). Also, the global rule visibility allows formal analysis such as checking consistency among rules, resolving conflicts between rules and specified policies regarding non-exposure of certain data and its compositions, checking for enforceability of rules, and efficient query planning Le, Kant, Athmnah, & Jajodia (2016); Athamnah & Kant (2016a). Although many of these approaches are applicable to more general data models, our focus has largely been on relational databases and their variants (for example, key-value stores), and we shall continue that in this chapter.

The proposed multiparty data sharing model includes a set S of parties, where for $1 \leq i \leq |S|$, each party P_i owns a number of relational “tables” T_{i1}, \dots, T_{ik} in a standard form such as BCNF/3NF. In this model, parties give controlled access to specific data from their relational databases to one another. The specific access of each party on individual data and compositions (for example, relational joins) of data (key matching) from two or more parties is given by an explicit and globally visible (at least on a need-to-know basis) set of “rules”. A rule in our model is described by a 4-tuple, namely, $\{JP, A, SC, P\}$, where JP is the “join path” involving “meaningful joins” over a set of tables, namely \mathcal{T} , from one or more parties. A is the set of attributes of \mathcal{T} that are allowed to be accessed, SC is the set of tuple selection conditions specified, and P is the party to whom this access rule is given. The notion of “meaningful joins” is similar to “lossless join” for a single party; it is described in detail in Le, Kant, Athmnah, & Jajodia (2016) including the required constraints on attributes put up by a party for sharing and the “stubs” that may be needed for appropriate format conversions.

We assume that the rules are mutually consistent, do not violate the specified safety properties, and are *enforceable*. Consistency refers to the fact that if a party P is given rules to access two joinable tables T_1 and T_2 , then it should also have a rule that provides access to $T_1 \bowtie T_2$. Consistency is necessary since it is not possible to prohibit P in this case to obtain access to $T_1 \bowtie T_2$ whether or not this is allowed by the rules. Consistency can be achieved via a systematic rule composition procedure as discussed in Le et al. (2012a). The enforceability means that if P is given access to $T_1 \bowtie T_2$, then at least one party, say Q , is given access to T_1 and T_2 individually, so that it is possible to actually do the join (and ship the results to P if $Q \neq P$). These issues including efficient query planning have been addressed in Le et al. (2013a, 2014); Le, Kant, Athmnah, & Jajodia (2016); Athamnah & Kant (2016a) and continue to apply here.

4.2.2 Representing Tuple Relationships

We extend the basic model by allowing a graph, G_{ij} to be associated with T_{ij} of party P_i (not every table may need such a graph). The graph G_{ij} represents relationships across tuples of table T_{ij} . Such tuple level relationships are generally not easily expressible or workable in the relational context, and hence the use of associated graph. The nodes of the graph, being tuples, can be identified by their primary key. The nature of tuple relationships depends on what the tuples represent, some examples being similarities (in case of products, customers, or events), kinship/friendship (in case of people), physical proximity (in case of facilities), logical proximity (e.g., for IP addresses), temporal proximity (in case of events), etc.

For most types of relationships, such as proximity or affinity, there is a degree associated with the relationship, which is conveniently represented by the *edge weight*. For example, “similarity” between two nodes can be represented by a number in range $[0,1]$ where 0 means no similarity and 1 means that the nodes are identical. In such a

representation, edges with weights below some threshold may not be used at all. For certain relationships, such as friendship or kinship, a weight may not make sense and can be assumed to be 1 for all existing edges.

There are two ways of representing tuple relationships: (a) explicit, i.e., explicitly specified edges and their weights, and (b) implicit, where graph is specified in terms of values of attributes of the table. In the latter case, the graph will really be a multi-graph, with similarity defined by a subset of the attributes of the table. For example, if the table represents laptop attributes, the relevant attributes for similarity might be price, weight, processor type, and screen resolution. Note that the last two attributes here are not numeric, and a suitable conversion function is required before they can be used for similarity comparison. We do not delve into the issue of how such a function is defined; obviously, the definition is very much domain dependent. In some cases, such as the “processor type”, the conversion function may only produce categorical values which can be represented by enumerated values, e.g., “Intel Kabylake”=1, “Intel Skylake”=2, “AMD FX”=3, etc. In other cases, such as “screen resolution”, one could use either categorical values or numeric (e.g., number of pixels).

In either case, we also define a “similarity function” which maps these values to the range [0..1] with 1 meaning perfect similarity. In this regard, consider a graph with N nodes, and the vector $A = (A_1, \dots, A_K)$ of attributes that are relevant for estimating similarity. Then i th node has attribute value vector $V_i = (v_{1i}, \dots, v_{Ki})$, where v_{ki} is the value for the attribute A_k . (Note that we are dealing with converted values, which are either enumerated or numeric values). Then, we could then define the similarity between nodes i and j on attribute A_k , denoted ξ_{ij}^k , as $\xi_{ij}^k = f_k(v_{ki}, v_{kj})$ where $f_k(x, y)$ is the similarity function with respect to attribute A_k . For categorical values, f_k can be regarded as the indicator function, i.e., 1 if $v_{ki} = v_{kj}$ and 0 otherwise. For numeric values, it must be monotonically increasing as x and y get closer, where “Closer” is defined in terms of $\text{avg}(x, y)/\max(x, y)$.

Now, the tuples i and j are considered similar according to attribute k if $x_{ij}^k > \lambda_r$ where λ_r is some threshold value. For example, if we are looking for highly similar tuples, we may choose $\lambda_r = 0.95$. The threshold may be either constant for all queries, or allowed to be chosen on a per query basis.

Since we are interested in similarity relative to K attributes, we also need to define a mechanism to compose similarity across multiple attributes. The simplest interpretation is that we want the similarity test to succeed according to every one of the desired K attributes. In general, it is possible to define other methods as well, but we do not consider them here.

4.2.3 Collaborative Databases with Similarity Queries

Given the requirement of rule consistency (see above), any valid query Q is authorizable via a unique rule r . Suppose that r involves a “join path” $JP^{(r)}$, i.e., a sequence of meaningful joins over some tables, say, $T_1^{(r)}, T_2^{(r)}, \dots, T_k^{(r)}$. Let $\eta^{(r)}$ denote the set of tuples accessible over this join-path according to the rule R . Let $\eta_i^{(r)}$ denote the projection of $\eta^{(r)}$ over the attribute set of table $T_i^{(r)}$. Note that the query Q will typically have a selection condition that asks for fewer tuples than the rule allows. Accordingly, let $\eta_i^{(Q)} \subseteq \eta_i^{(r)}$ denote the tuples from table $T_i^{(r)}$ that the query wants included in the result. Similarly, Q may request only a subset of the attributes of each table $T_i^{(r)}$ that R authorizes.

Let us now define another query Q' that modifies the results of Q based on the relationships contained in the graphs for the attributes that are included in the results of query Q . Consider one such graph, say, $G_i^{(r)}$ that corresponds to the included attributes of table $T_i^{(r)}$. In the following, we adopt the convention that if a certain graph $G_i^{(r)}$ is null, it has no effect on Q' . There are two types of queries in this regard:

1. **Query Expansion:** Here Q' asks for inclusion of additional tuples that are related to those in Q according to the similarity property given by the graph

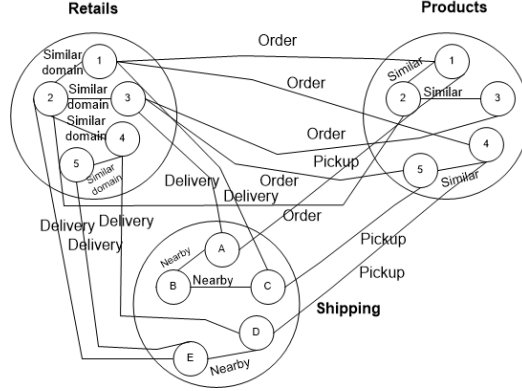


Figure 4.1: An Illustration of Query Type in Supply Chain

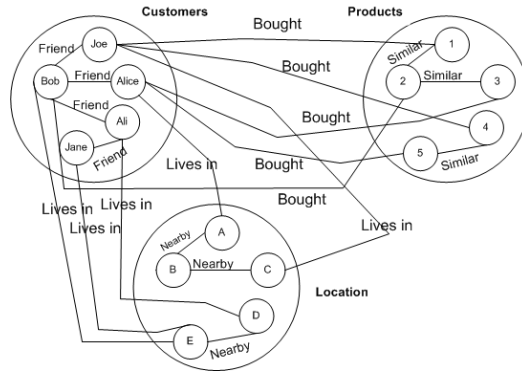


Figure 4.2: An Illustration of Query Type in e-commerce

$G_i^{(r)}$. That is, we “pull in” additional tuples (or nodes) in $\eta_i^{(r)}$ that similar to the ones in $\mathcal{G}_i^{(Q)}$. In the worst case, this operation may “pull in” all the tuples in $\eta_i^{(r)}$ (i.e., those that are authorized by the rule), but no more.

2. **Query Restriction:** Here the query Q' asks for a further filtering of results of Q based on the graph $G_i^{(r)}$. Let $\mathcal{G}_i^{(Q)}$ denote the subgraph of $G_i^{(r)}$ restricted to only the tuples in $\eta_i^{(Q)}$. Then the returned tuples can be further filtered based on the similarity (and possibly other) properties of this subgraph. For example, we may want to return the largest set of tuples that are similar to one another.

Query expansion and restriction provide the graph property based augmentation of collaborative database queries that we are looking for. Fig. 4.1 and Fig. 4.2 illustrates the query type in two different industries. Fig. 4.1 captures 3 entities: retailers,

products, and shipping and captures the relationship between them in three relational tables, namely pickup, delivery, and order. It also captures the relationship between the tuples. The supply chain query expansion allows us to answer queries such as: Given a set stores selling some category of perishable items, are there items that will last at least certain amount of time and transportation in the area with enough space that will be able to get them to these stores in time.

Fig. 4.2 show query type in an e-commerce scenario with 3 types of entities: customers, products, and locations. Though in this case there are 2 relational tables namely “Bought” and “Lives in”. In this example, we can answer the query restriction: if Bob bought a product, then also return all the products similar to this product that were bought by Bob’s friends. This will return product 1 since Joe, who is Bob’s friend, has product 1 which is also similar to product 2. We wish to execute such queries where the both the attributes on which similarity is expressed and the thresholds for similarity can be specified at run-time.

The key issue in answering expanded or restricted queries is the similarity search, a key topic that has been thoroughly researched in many data mining contexts. For example, in image processing and natural language processing, the search is based on a set of “features” (or attributes) Kulis et al. (2013); Mei et al. (2014); Zhao (2017).

However, in all of these applications, the set of features (or attributes) on which the similarity is evaluated are fixed and the threshold for similarity is predefined. Such an assumption allows for extensive preprocessing of data before the queries are executed, such as dimensionality reduction methods or filtering out irrelevant data. This is not possible for our problem except in special circumstances as discussed later. Furthermore, our problem requires response times without having to install an additional extensive search infrastructure in addition to that needed for the database and normal enterprise applications.

4.3 Related Work

Query expansion and restriction based on graph properties involves processing graphs with number of nodes equal to number of relational tuples. For certain tables, such a graph could become very large, and compact (or compressed) representation can be highly valuable. Compressed graph representations have been examined extensively in the literature Maneth & Peternek (2015). Also, since each attribute of interest represents a separate dimension, dealing with large number of dimensions could be of interest, although we do not believe that user queries would typically ask for similarity relative to more than a few attributes. Nevertheless, the problem of nearest neighbor in high dimensional data is somewhat relevant to our work Muja & Lowe (2014, 2009). Locality sensitive hashing based graph compression and nearest neighbor search are also amply discussed Slaney & Casey (2008); Andoni & Indyk (2008). Oliver, et al., present an algorithm called TLSH Oliver et al. (2013), which is claimed to be free from the disadvantages of previous algorithms such as Nil-sima Damiani et al. (2004), Ssdeep Kornblum (2006), and Sdhash Roussev (2010). All of these algorithms are designed to detect changes in text files such as spam emails, and TLSH shows a more consistent behavior in that the similarity metric goes down slowly as more noise is introduced.

Unfortunately, much of the graph compression literature considers only plain graphs; there is not much literature on graphs with node/edge attributes. One such exception is Álvarez et al. (2010) that proposes attributed graph compression based on extensions of k^2 trees; however, the proposed scheme creates a matrix involving each possible value of the attribute, which is useful only for those attributes that have a rather small number of possible values. It could work by dividing attribute values into a small number of buckets, but the resulting large bucket size makes the results unsatisfactory.

Recommendation algorithms are best known for their use on e-commerce web-

sites. Most such algorithms start by finding a set of customers whose purchased and rated items overlap the user's purchased and rated items. The algorithm aggregates items from these similar customers, eliminates items the user has already purchased or rated, and recommends the remaining items to the user. Two popular versions of these algorithms are *collaborative filtering* and *cluster models* Resnick et al. (1994). Amazon.com is best known e-tailer for providing purchase recommendations. Their algorithm does an item level collaborative filtering G. Linden et al. (2003) that works as follows: For each of the user's purchased and rated items, the algorithm attempts to find similar items, then combines those similar items into a recommendation list G. D. Linden et al. (2001). It then aggregates the similar items and recommends them. They use recommendation algorithm to personalize the online store for each customer to determine the most-similar match for a given item. The algorithm builds a similar-items table by finding items that customers tend to purchase together. Given a similar-items table, the algorithm finds items similar to each of the user's purchases and ratings, aggregates those items, and then recommends the most popular or correlated items.

Cluster models find customers who are similar to the user, then divide the customer base into many segments and treat the task as a classification problem. The algorithm's goal is to assign the user to the segment containing the most similar customers. It then uses the purchases and ratings of the customers in the segment to generate recommendations, the segments typically are created using a clustering or other unsupervised learning algorithm.

It may be noted that the computations above are done in the background with no direct specification by the user how the similarity issue is handled (i.e., which attributes are most important or what is the threshold of similarity). Most web-sites, including Amazon.com, do provide ability to select specific features of interest, but that's different from similarity and often not satisfactory.

4.4 Similarity Search Algorithm

The key to supporting the desired graph augmented queries is the ability to efficiently identify items similar to the given item(s) in the augmented graph. This capability may be required for an arbitrary set of attributes that are relevant for the query, and the similarity threshold used for each attribute could vary from query to query. For example, in our earlier example of laptops, one user may want to retrieve all laptops that are similar in price and weight at 90% level, whereas another one is more interested in weight and processor type at different similarity level.

4.4.1 Similarity Search

In the general case with queries able to use any set of attributes and thresholds for similarity, it becomes very difficult to devise specialized techniques since they would invariably require some preprocessing. Nevertheless, we show that there are several opportunities for gaining efficiency, and this is one key novelty of this work.

To start with, let us consider two simplistic approaches that allow for the attribute/threshold flexibility. The first approach is to create a complete graph for each attribute where the edges are labeled with the similarity metric between the any two nodes. With k attributes and n tuples, this would require $O(n^2k)$ space, which can be prohibitive. For example, with 10K tuples, each attribute would need at least 1.6 GB of memory. However, given such graphs, it is very efficient to navigate the graph starting with any node for which similarity search is desired. Also, instead of obtaining the similarity set with respect to each attribute and then obtaining intersection, it is rather trivial to switch back and forth between these graphs and obtain the final result. We do not consider this approach further due to space explosion.

An alternate approach is to maintain sorted tuple lists according to each attribute value. This would require only $O(nk)$ space, and expense of maintaining sorted lists is negligible unless the tuple values change very frequently. In this case, the edges

can be considered to exist only between adjacent nodes. For further compactness, we could have a single multi-list, each with different colored edges for each attribute type. The problem now is that given a set S of nodes (or tuples), how do we efficiently determine the containing node set S' such that for any $i \in S, j \in S'$, the similarity measure for k th attribute ($k = 1..K$), denoted, $\xi_{ij}^{(k)}$, is above a threshold $\theta^{(k)}$, where $0 \leq \theta^{(k)} \leq 1$.

A simple way to solve this problem is to consider one attribute at a time, and start scanning the nodes starting with a node in set S and proceeding until we get all the nodes with similarity within the specified threshold. This provides us with K “flows”, i.e., an ordered list of nodes that are relevant with respect to each attributes. Then the “Simple-Scan” algorithm (that we use as a baseline for comparison), then takes an intersection across all these flows to determine the answer. Though not necessarily optimal, a good heuristics is to do the intersection starting with the smallest flow and moving towards the larger ones.

4.4.2 Enhanced Algorithm

We now propose an enhanced algorithm, called “Enhanced-Scan”, that exploits the structure of the “flows” in order to compute the matchings faster than a straightforward comparison. All flows are represented through a multigraph, where the edges for each relevant attribute can considered to have a different color. During this graph construction, we estimate a *ratio*, say η_k for flow k that represents the number of nodes that do not share all the required attributes divided by the total number of nodes. This ratio is computed as we scan the attributes initially during flow construction. The ratios will be modified incrementally as we traverse the flows, and do not require any additional effort.

We then start traversal of the flow that has the highest ratio and start scanning to find the match. For the traversal, we maintain two separate pointers one for each

Algorithm 6 \mathcal{S} : Efficient Graph Similarity Determination

```
1: void Enhanced_Scan( $G, n, thres, Att$ )//  $G$  is the graph ,  $n$  is the query node,  $thres$  is the
   threshold,  $Att$  is the set of attributes
2:  $Similar = NULL$ ; // Set of similar nodes
3:  $Flow = NULL$ ; // Set of starting nodes, and ending nodes in the graph for each attribute in
   set  $Att$ 
4: for each  $att$  in  $Att$  do
5:    $f = \text{get\_flow}(n, thres, att)$ ; // get the flow for attribute  $att$  on threshold  $thres$ 
6:    $Flow = Flow \cup f$ ;
7:  $l = 0$ ; // Number of nodes in which they do not share multiple attributes
8:  $b = 0$ ; // Number of nodes in which they share multiple attributes
9:  $k = 0$ ; // To determine when to exit from the while loop
10:  $current\_node = NULL$ ; // This is a pointer for the current node in the flow
11: while  $k \neq -1$  do
12:    $Perc = NULL$ ; // Set of ratios for each flow
13:   for each flow  $f$  in  $Flow$  do
14:      $l = \text{get\_number\_shared}(f)$ ; // get the number of nodes that share attribute
15:      $b = \text{get\_overall\_number}(f)$ ; // get the overall number of attribute
16:      $p = l/b$ ;
17:      $P = \{p, f\}$ ;
18:      $Perc = Perc \cup P$ 
19:    $flow = \text{pick\_higher}(Perc)$ ; // Pick the flow that has higher ratio
20:    $current\_node = flow[\text{first\_node}]$ ; // update the current node for the flow
21:    $x = \text{len}[Flow]$ ; // return number of flows
22:    $y = 0$ ;
23:   for each flow  $f$  in  $Flow$  do
24:     if  $\text{search}(current\_node) == \text{True}$  then
25:        $y++$ ;
26:      $\text{update\_Flow}(flow[\text{get\_next}(\text{first\_node})])$ ; // remove one node from this flow
27:   if  $y == x$  then
28:      $Similar = Similar \cup current\_node$ ;
29:      $current\_node = \text{get\_next}(current\_node)$ ;
30:   if  $current\_node == \text{get\_last}(Flow)$  then
31:      $k = -1$ 
32: Return  $Similar$ 
```

flow starting from the first node that is included in the threshold boundary. We scan the nodes of all other flows (without moving their pointer) to find this node. If we find it, we put the id in the similarity list, and move the pointer to the next node that has all attributes (or colors). We recompute the ratios for all flows (since the nodes of one flow have changed), and pick the flow with the highest ratio. We keep going with the traversal using this approach, until we finish one of the flows.

Algorithm \mathcal{S} sketches the overall process. The first for loop (line 4) obtains the flows of nodes that are included for our threshold on each flow. Then the while loop (line 12) gets the number of shared nodes, the overall number of nodes, and computes

the ratio. Function `pick_higher` (line 19) picks the flow with higher ratio. After that, we start the traversal by choosing the flow's current node, and search the nodes of the other flows (see line 23). Finally, if we find the node, we add it to the similarity list, and move the flow's pointer to the next node that has both attributes. We keep doing this until we cover all nodes of the flows.

Let us illustrate the process with an example. Suppose we have two flows, $F_A = \{1, 3, 2, 4, 5, 6, 7, 8\}$, and $F_B = \{1, 9, 10, 5, 7, 8\}$. Note that these flows are constructed after the initial scan of the tuples, and typically will have much fewer nodes than the number of tuples. Also, the only nodes included are those that are already determined to be similar according to the similarity function and threshold, so, the only remaining problem is to match nodes across different flows. Now, suppose that we want to return the nodes similar for node 1 for both attributes (flows). We note that nodes 3, 2, 4, 6 from F_A do not appear in F_B ; this corresponds to a ratio of 0.5 (4 out of 8 elements). Similarly, 9 and 10 from F_B do not appear in F_A , which gives a ratio of 0.3 (2 out of 6). Since $0.5 > 0.33$, we have to start with flow F_A .

As the root node has both attributes, it will be added to the similarity set. We then move P_A (pointer in flow A) to the next node that has both colors, $P_A = 5$ (we can see in this flow that the jumps come in clumps, since nodes 2, 3, and 4 do not have both the attributes). We recompute the ratios after the change, which gives $0.25 < 0.3$. This means that we have to switch to order F_B . Now, $P_B = 1$, and since it is in the similarity set, we move the pointer to the next node that has both colors, i.e., $P_B = 5$. We recompute the ratios and scan, we keep going until one of the orders is terminated. After finishing the traversal we will have a similarity set that has all the nodes in which they are similar to node 1 with respect to both the attributes.

The key quantity in the running time of the algorithm is N_F , the number of flows, $N_{S_{all}}$, the number of comparisons for the entire flow, $N_{S_{half}}$, the number of comparisons for half of the flow, S_i , number of skips in the other flow, N_n , total

number of nodes on the other flow. The overall worst case complexity of the algorithm \mathcal{S} is $O(\prod_{i=1}^{N_F} N_{F_i} * (N_{S_{all_i}} + N_{S_{half_i}}) * (1 - S_i/N_{n_i}))$.

This algorithm gains its efficiency by not comparing each node in both flows. Without it, if F_A is of size N , and F_B is of size M , the complexity will be $N * M$. Instead, we exploit the structure of the flows so that we always choose the order that has the least possible number of comparisons.

4.4.3 Exploiting Dominating Attributes

We now propose another technique that exploits the relationships between the data. Most real data sets have relationships between their attributes. For example, in most of the e-commerce related product data, high-end features come with high prices. In the actual laptop data that we harvested from e-commerce sites, we found that the high priced laptops are lighter, have higher end processors, have more memory (e.g., Intel i5 or i7 processors), etc. Thus, certain attributes can be considered as dominant in the sense that they roughly define others. We try to take advantage of such relationships by finding the dominating attribute(s) by doing the *Principal Component Analysis* (PCA) of the data set, as described below.

The PCA essentially produces a sequence of *virtual attributes* in decreasing order to importance. Each of these virtual attributes is a weighted linear combination of original attributes. We then check how many of these virtual attributes are needed to catch much of the data variance (say, 80%). Following this, we can construct the “flows” on virtual attributes for our Enhanced-Scan algorithm, and use those for similarity search.

The success of PCA based techniques obviously depends on how the data is clustered. Ideally, we want the data to be clustered in an ellipsoid like form that is long in a few virtual dimensions (these are the dominant dimensions), and short in others. Now if the original data does not follow such a form, it may still be possible

to partition the data such that each partition is close to an ellipsoid. Two types of partitions are possible in this regard: (a) data driven clustering, and (b) product attribute driven clustering. While the data driven clustering was actually successful in our laptop example, there is an inherent problem arising due to the behavior of nearly all known clustering algorithms – the clustering algorithm may group the data in unexpected ways and thus destroy, rather than help in exposing, the ellipsoidal nature of data. Consequently, we focus on (b) only here. The rationale for (b) is that certain product attributes are known in advance to be crucial and often provided as filtering choices to the users. For example, for laptops, such partitions could include: (1) laptops with HDD vs SSDs, (2) small portable or “ultrabook” types (e.g., 13 inches or less), and normal ones, etc. In the following we discuss how to handle these in the analysis. Generally, we expect a very small number of partitions according to each criteria, and in fact it is not useful to use the following analysis if there are many partitions.

Consider the partitions $P = \{P_1, P_2, \dots, P_m\}$ for a given attribute. We first compute the PCA for each partition and check if we can merge them in order to reduce the total number of PCAs. For merging, we compute the centroid for each partition $P_i \in P$, denoted as C_i , and defined as the point such that the sum of Euclidean distance from all the points of the partition is minimized. This is easily shown to be the point with arithmetically averaged coordinate values. Now, consider two PCA components W_i and W_j (in partitions P_i and P_j) corresponding to the first PCA component, and let θ_{ij} denote the angle between them that is no larger than 90° . Let L_{ij} denote the line segment connecting the centroids of P_i and P_j . Let ϕ_i denote the angle between L_{ij} and P_i that is no larger than 90° . Similarly, let ϕ_j denote such an angle between L_{ij} and P_j . Then we claim the following:

Definition 4.1. Let ε denote a small fraction representing the tolerance. Then any two partitions, P_i and P_j , are said to be mergeable if (a) $\theta_{ij} < \varepsilon$ and (b)

$$\|L_{ij}\| \text{Sin}(\max[\phi_i, \phi_j]) < \varepsilon \min(\|P_i\|, \|P_j\|)$$

Using this criteria, we merge as many partitions as possible. Note that the merge order does matter – for example $\text{merge}(\text{merge}(P_i, P_j), P_k)$ is not the same as $\text{merge}(P_i, \text{merge}(P_j, P_k))$. Because of this, the problem is easily shown to be NP-hard, but a simple greedy heuristic (i.e., merge the best pair first), works reasonably well.

When the partitioning exists on multiple attributes, there is the natural question whether the partition according to the first attribute should be further partitioned according to the second attribute, etc. This again leads to an NP-hard problem and may create too many small partitions. Thus, we do not perform any cross-attribute partitions; instead, we do the partitioning and merger on each attribute separately, and then choose the set that yields the best PCA result.

Table 4.1: Graph Similarity on one relation

#	Query	thres	#nodes	Simple	Enh.
1	$S(G,945,0.50,["L", "W"])$	0.50	13	12706	5036
2	$S(G,945,0.60,["L", "W"])$	0.60	11	4577	1078
3	$S(G,945,0.70,["L", "W"])$	0.70	6	311	47
4	$S(G,945,0.80,["L", "W"])$	0.80	1	27	7
5	$S(G,945,0.90,["L", "W"])$	0.90	1	12	5
6	$S(G,819,0.50,["L", "W"])$	0.50	13	2077	597
7	$S(G,819,0.60,["L", "W"])$	0.60	10	1248	251
8	$S(G,819,0.70,["L", "W"])$	0.70	7	792	298
9	$S(G,819,0.80,["L", "W"])$	0.80	6	312	207
10	$S(G,819,0.90,["L", "W"])$	0.90	2	81	53
11	$S(G,732,0.50,["L", "S"])$	0.50	163	19598	8305
12	$S(G,732,0.60,["L", "S"])$	0.60	141	14978	8013
13	$S(G,732,0.70,["L", "S"])$	0.70	101	10412	7282
14	$S(G,732,0.80,["L", "S"])$	0.80	50	4964	4242
15	$S(G,732,0.90,["L", "S"])$	0.90	30	2868	2526
16	$S(G,833,0.50,["L", "S"])$	0.50	163	19598	8305
17	$S(G,833,0.60,["L", "S"])$	0.60	141	14978	8013
18	$S(G,833,0.70,["L", "S"])$	0.70	101	10412	7282
19	$S(G,833,0.80,["L", "S"])$	0.80	50	4963	4241
20	$S(G,833,0.90,["L", "S"])$	0.90	19	2626	2344

Table 4.2: Rules for the Running Example

#	Authorized attribute set	Auth. Join Path	To
1	{oid, pid, L, W, S}	E	P_E
2	{oid, pid, L, W, S}	$E \bowtie_{oid} C$	P_E
3	{oid, pid, L, W, S}	E	P_C
4	{oid, name, address}	C	P_C
5	{oid, pid, name, address, L, W, S}	$E \bowtie_{oid} C$	P_C

4.5 Performance Evaluation

4.5.1 Enhanced vs. Simple Scan

For this evaluation we used a data set called “Adventure Works” Microsoft (December.2014/n.d.). The data set has 70 different tables, and concerns e-commerce. Table 4.1 shows a number of cases for Algorithm \mathcal{S} using the e-commerce table from this data set, which describes products sold by the ecommerce company. This table has 26 attributes and more than 100K tuples. We use only 5 attributes here for similarity computations.

Table 4.3: Similarity queries

QR	TR	Sim attr set	Auth JP	Authorized attr set	Selection cond	To
1	0.50	L	E	{oid, pid, L, W, S}	None	P_E
2	0.50	L,W,S	$E \bowtie_{oid} C$	{oid, pid, L, W, S}	$pid > 100 \wedge L < 900$	P_E
3	0.75	L,W,S	$E \bowtie_{oid} C$	{oid, pid, L, W, S}	$pid > 100 \wedge L < 900$	P_E
4	0.90	L,W,S	$E \bowtie_{oid} C$	{oid, pid, L, W, S}	$pid > 100 \wedge L < 900$	P_E
5	0.50	L	E	{oid, pid, L, W, S}	$pid > 100$	P_C
6	0.50	address	C	{oid, name, address}	None	P_C
7	0.50	L	$E \bowtie_{oid} C$	{oid, pid, L, address}	$pid > 100$	P_C

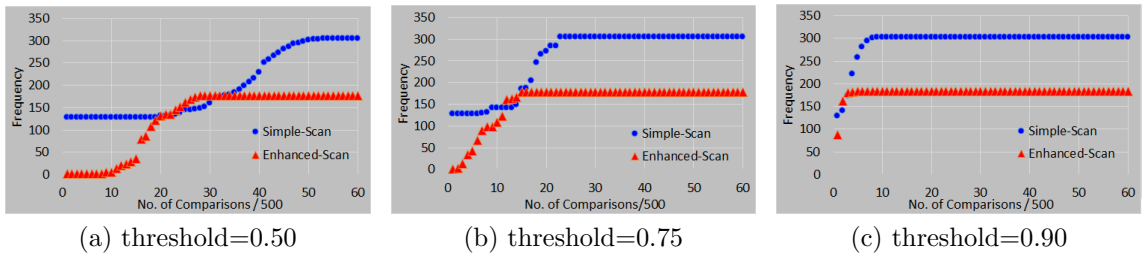


Figure 4.3: Performance of Enhanced-Scan vs. Simple-Scan

Each row in Table 4.1 shows one case, and the columns show respectively (a)

”Query” that includes the query node, and a set of attributes, (b) ”Thres”, the threshold for the query, (c) ”#nodes” that are similar to the query’s node, (d) ”Simple”, the number of comparisons using the Simple-Scan algorithm, and (e) ”Enh.”, the number of comparisons using the Enhanced-Scan algorithm.

For example, in case 1 we are given as input, node_id=945, and threshold=0.50, and a set of attributes [”L”, ”W”] which represent list-price, and weight. We want to query all the nodes that are similar to node_id 945 on both list-price, and weight within the threshold 0.50. We can see that the result for this query in the table is 13 similar nodes, and the total number of comparisons for Simple-Scan algorithm is 12706 while it is 5036 for Enhanced-Scan.

We can see that in all cases, the algorithm \mathcal{S} was able to retrieve the queries with significantly fewer comparisons as compared with the Simple-Scan algorithm. Of course, the retrieved similarity set is identical in both cases, as it should be since our method is exact.

For further evaluation, we use our running example in Athamnah & Kant (2016a) section C, and simplify the schema for the following two parties: (a) E-commerce, denoted as E, is a company that sells products online, (b) Customer Service, denoted C, provides customer service functions (potentially for more than one Company), each party owns one table described as follows. • E-commerce(order_id, product_id, list_price, weight, size) as E

• Customer_Service (order_id, customer_name, address) as C

The relations are self-explanatory, with underlined attributes indicating the key attributes. In the following, we use oid to denote order id for short, pid for product_id, L for list_price, W for weight, S for size, name for customer_name, addr for address. Table 4.2 lists all rules in form of authorized attribute set, join path and the party to which the rule is given.

We already discussed rule enforceability and query planning in our prior work on

the subject Le et al. (2012a, 2013a, 2014); Le, Kant, Athmnah, & Jajodia (2016); Athamnah & Kant (2016a). Accordingly, queries involving these rules can be easily checked for feasibility and a plan for them can be generated efficiently.

Table 4.3 shows a list of similarity queries that we use for this evaluation. The columns in this table are respectively: (query number, threshold, authorized join path (JP), authorized attribute set, selection condition, and party who is querying).

Query 1 and 6 are of **query restriction** type. First, the query returns tuples for this join path, then the similarity on the “L” attribute filters the tuples to get the tuples in which their similarity value is 0.50 and above. The other queries belong to **query expansion** type. For example, query 5 is the same as query 1 but it has selection condition which causes the query to return a subset tuples of query 1. The similarity on attribute “L” then pulls in additional tuples in whose similarity value is 0.50 or above.

We compared the two algorithms on queries 2, 3, and 4, as they have three different thresholds 0.50, 0.75, and 0.90. Fig. 4.3a shows the performance for threshold=0.50, where we return the resulting tuples of join for query 2, then for each tuple we return all the similar tuples on the attribute set L,W,S with threshold=0.50. As before, we divide the number of comparisons (ranging from 100 to 30K) into 60 ranges for plotting purposes. We can again see that Enhanced-Scan does significantly better – overall about 47% of the comparisons as compared with the Simple-Scan algorithm.

In Fig. 4.3b we repeat this on query 3 with threshold=0.75 for the same attribute set L,W,S. The ratio here is again 47%. Finally, Fig. 4.3c represents the chart for query 4 with threshold=0.90 for the same attribute set L,W,S. Here Enhanced-Scan does only 29% as many comparisons as Simple-Scan, which is significantly better.

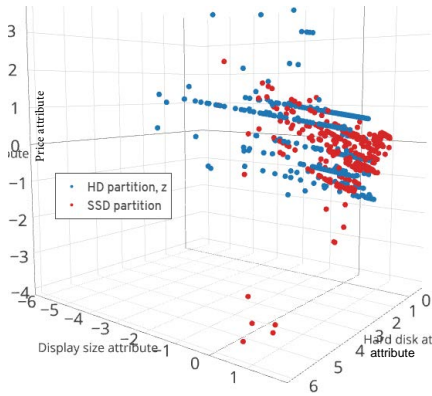


Figure 4.4: PCA for partition

4.5.2 Enhanced Scan with PCA

In order to evaluate the advantages of using PCA, we used a rather large data set collected from Amazon.com, Inc. for laptops. Of the many attributes, we considered the following in our analysis: hard disk type (HDD, SSD), hard disk size, screen size, RAM size, and price. As an illustration of partitioning based PCA analysis, we considered data partitioning based on the hard disk type and then computed the PCA for each partition for three attributes (hard disk size attribute, display size attribute, and price attribute) to check if we can find a dominating attribute among them. Fig. 4.4 shows the two data partitions in the original attribute space where the red points belong to the SSD partition, and the blue points belong to the HDD partition. The analysis shows that for the HDD partition, the dominant (virtual) attribute covers 90% of the variance whereas for SSD it covers 80% of the variance.

We then ran the merging algorithm on these PCAs to check if they are mergeable. The steps here are to compute the centroid on each partition, find the line segment between centroids, project the data on the PCA line segment, and check the merging conditions. It turns out that the two partitions cannot be merged in this case. That is, we get a better dominating virtual attribute if we keep the two partitions separate. This can also be seen by doing PCA on the entire data set, which gives on 60% variance coverage by the principal component.

Table 4.4 compares the results of the Enhanced-Scan algorithm applied directly (denoted as \mathcal{S}) and applied following the PCA analysis (denoted as \mathcal{M}). The advantages of the PCA analysis are clear. We see that in both cases we get approximately the same number of similar items, but the number of comparisons goes down substantially for \mathcal{M} . While fewer comparisons is largely related to the smaller data sets, in general, the fewer similar items may be retrieved, which is the key tradeoff of partitioning.

Table 4.4: Similarity queries results

#	Query Issued	thres -hold	#similar		#comp.	
			\mathcal{S}	\mathcal{M}	\mathcal{S}	\mathcal{M}
1	$\mathcal{S}(G,12,0.50,["H", "D", "P"])$	0.50	512	512	22530	7899
2	$\mathcal{S}(G,35,0.60,["H", "D", "P"])$	0.60	552	552	19877	6596
3	$\mathcal{S}(G,162,0.70,["H", "D", "P"])$	0.70	432	432	18675	6512
4	$\mathcal{S}(G,192,0.80,["H", "D", "P"])$	0.80	150	150	14596	4869
5	$\mathcal{S}(G,335,0.90,["H", "D", "P"])$	0.90	56	56	10089	3363
6	$\mathcal{S}(G,474,0.50,["H", "D", "P"])$	0.50	416	416	20598	8963
7	$\mathcal{S}(G,784,0.60,["H", "D", "P"])$	0.60	386	386	17693	5897
8	$\mathcal{S}(G,1209,0.70,["H", "D", "P"])$	0.70	253	253	16947	5659
9	$\mathcal{S}(G,86,0.80,["H", "D", "P"])$	0.80	198	198	15362	5129
10	$\mathcal{S}(G,115,0.90,["H", "D", "P"])$	0.90	44	44	10631	3549
11	$\mathcal{S}(G,78,0.50,["H", "D", "P"])$	0.50	229	223	128176	42727
12	$\mathcal{S}(G,100,0.60,["H", "D", "P"])$	0.60	284	280	112968	38439
13	$\mathcal{S}(G,368,0.70,["H", "D", "P"])$	0.70	84	84	24299	8632
14	$\mathcal{S}(G,424,0.80,["H", "D", "P"])$	0.80	36	34	12439	4349
15	$\mathcal{S}(G,515,0.90,["H", "D", "P"])$	0.90	183	182	108952	36327

4.6 Conclusion

In this chapter, we consider the problem of executing queries on multiparty repositories where the queries need to retrieve tuples based on similarity function defined via graphs. Such queries are extremely useful in providing additional information to the user based on the graph relationships that may represent similarities between products, relationships between resources, etc. We designed an algorithm that performs significantly better than the straightforward scan of the tuples while still retaining full flexibility in terms of the attributes and thresholds for similarity. The algorithm is further enhanced by exploiting dominating attributes and partitioning when possible.

Allowing attributes and thresholds to vary at query time arises in many other contexts as well and our algorithms can be exploited. For example, if the feature of interest is the skin color of facial images, we can make the threshold quite stringent if we are quite certain about the precise skin tone, but otherwise use a looser threshold. In the future, we will consider enhancements to our proposed methods to make them more efficient; for example, we will consider combinations of data and attribute partition driven clustering to get better results.

CHAPTER 5

A FRAMEWORK FOR MISCONFIGURATION DIAGNOSIS IN INTERCONNECTED MULTIPARTY SYSTEMS

5.1 Introduction

With increasing societal reliance on and complexity and diversity of automated services, the service outages and slowdowns become a major threat to business continuity and goodwill. A recent study of best practices in the Fortune 1000 companies Elliot (2014) has shown that the average total cost of unplanned application downtime per year is \$1.25 billion to \$2.5 billion overall, and on the average \$0.5 million to \$1 million per hour for critical application failures in large organizations.

It has been well recognized that inconsistent changes relating to configurations and IT production environments are the top root causes of system outages or performance problems Oppenheimer et al. (2003); Yin et al. (2011); Barroso et al. (2013). Past studies have indicated that up to 80% of downtimes of mission-critical applications are caused by mistakes, miscommunications or misunderstanding related to changes Connolly (2014) and up to 85% of performance incidents can be traced to changes Cappelli (2015). However, the root cause identification of performance problems usually takes one week on average Cappelli (2015), and often much longer due to misleading messages Connolly (2014). The ongoing “DevOps” transformation of IT, which melds the line between development and operations and thereby expects to provide *continuous deployment* Shahin (2015), will further exacerbate the problem of

misconfiguration related hiccups, unless effective solutions are found to quickly locate the root causes of the problem and fix them.

Large-scale cyber and cyberphysical infrastructures are naturally composed of multiple administrative domains, each managed or controlled by a party with potentially unique policies and configurations that are not shared with other parties. Yet user requests pass through the infrastructure of multiple parties. For example, in the typical situation of a customer in an organization accessing a remote web service, at least 3 parties are involved: the local organization, the internet service provider (ISP) that provides wide area network access, and the data center that hosts the web service. Each of these parties has its own configuration and test procedures to diagnose problems. However, in order to properly diagnose end to end problem, it is essential to run multiparty tests, and these would require cross-party access. We envision them as compositions of test probes provided by individual parties along with suitable cross party access rules. These access rules may not be statically granted, and instead granted specifically for testing purposes.

The multiparty access problem arises in a variety of scenarios, even including those involving a single enterprise. For example, large enterprises often have multiple business that do not freely share information and thus may use separate private clouds hosted in the same server farm. Even within an owned data center, there are often different teams with different expertise and responsibilities, (e.g., network team, security team, and database team), each of them has their own configuration and perhaps some limited access by other teams. In all these cases, when problems arise, a multi-party testing may be essential and such testing may grant more access rights than in the normal situation. In any case, given a basic test functionalities and access rights, one could ask several questions such as: Is the set of given access rights adequate to do a multiparty test? If a test can be carried out in multiple ways, what is the shortest test, or test involving fewest parties? These are the types of questions

that we are interested in addressing in this chapter.

It is important to point out the issues that we do not address in this chapter. First, we assume that the parties involved have a collaborative, rather than adversarial relationship. In other words, the parties do not attempt to circumvent the specified access rules, do not corrupt/alter the data provided to other parties, and do not hold back or filter any legitimate information. We assume that all collaborating parties are mutually authenticated using standard mechanisms, and the global controller tracks and limits the rate and number of concurrent tests in order to prevent DDoS scenarios.

Accessing data across parties can be technically challenging due to varying format and semantics of the data used by them, but these are not the focus of this work. Thus we proceed with the simple assumption that each party creates a “stub” for its data for uniform access by others.

Building a comprehensive misconfiguration diagnosis infrastructure involves many significant research, design, and implementation challenges. These include (a) exploring constraints to be exploited by the probes for diagnosis, (b) defining and handling access control across parties, (c) test design and selection, and overall testing infrastructure design, implementation, and evaluation. We note that this chapter is not concerned with the design of the probes provided by various parties or the design of suitable tests to uncover a specific issue. Instead our focus is on whether the given test can be implemented using the given probes, and if so, how it can be implemented most efficiently.

The outline of the chapter is as follows. Section 5.2 provides an example of a common multiparty system, namely the email system. Section 5.3 discusses various issues in building multiparty tests from individual probes provided by the parties. Section 5.4 defines the problem of test selection formally and proposes a solution. Section 5.5 then shows sample performance evaluation. Related works are discussed in section 5.6. Finally, section 5.7 concludes the discussion.

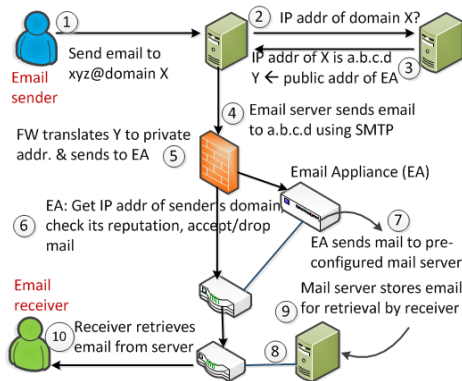


Figure 5.1: Typical Email Flow

5.2 An Example of Multiparty Service

In this section we discuss in some depth the configuration of Internet email, the inherent involvement of multiple parties, and how multiparty testing would be needed to diagnose the misconfiguration.

Fig. 5.1 shows the typical flow for email involving the email filtering appliance (EA). The E-mail server resolves the E-mail domain name to the public IP address of the domain using the DNS service. It then sends the E-mail using SMTP (Simple Mail Transfer Protocol) to the corresponding IP address. The email is intercepted by the enterprise firewall (FW), which translates the public IP address of EA to its (private) DMZ IP address and forwards traffic to the EA. The EA then does a DNS query on the sender domain name, compares the IP address of the sender to its own SensorBase database and determines the reputation score of the sender. It rejects the E-mail if it falls within a pre-configured reputation score, and sends a failure email back to the sender. The receiver then retrieves the email at its convenience.

The key resources here are obviously DNS, FW, EA, and Mail Server (MS). The DNS can be probed directly, but others generally do not provide any direct interface for querying. Thus, it is necessary to design and install Resource Probing Module (RPM) on others. With that, and assuming that FW, EA and MS are owned by the same party, it is adequate to have a single Mirror for this email system. Unfortunately,

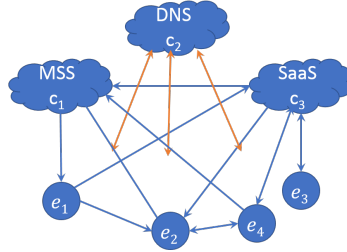


Figure 5.2: Email Configurations

the email setups are rarely this simple, as illustrated in Fig. 5.2, which shows 3 popular configuration methods and highlights the need for multi-party coordination.

In Fig. 5.2, e_i 's denote different enterprises, and c_j 's denote different cloud or service providers. Enterprise e_1 delegates email filtering to a third party c_1 , a Managed Security Services (MSS) provider. In this case, any incoming email to e_1 is delivered through cloud c_1 . Note that the outbound mail from e_1 is configured to be sent directly as shown by arrows to e_2 and c_3 . (Not showing all arrows to avoid clutter.) In contrast, e_2 and e_4 configure their email service locally for direct sends/receives to/from other entities. Finally, e_3 uses a cloud SaaS service for email; i.e., both sends and receives happen through the provider c_3 . This means that no email server talks with e_3 directly and is instead forced to go through c_3 . In all cases, the enterprises use a public DNS service hosted by the service provider c_2 .

In order to diagnose email problems in such an environment, each party will need to setup a number of probes relating to various problems in its sphere of control. Some common email issues include the following: (a) Spammers could use email servers to relay emails through them, usually by pretending to be the higher distance MX servers for the domain that may be engaged under heavy load or failure of the primary. (b) When an email DNS server has an "A" record for the sender but not the reverse record, the recipient email server may force all emails into the junk/spam folder or reject them outright. The same happens if the sender erroneously gets on a blacklist (not uncommon). (c) Email could be disrupted from other misconfigurations such as

those in the ISP network or those in the email appliances (EA) discussed above.

5.3 Diagnosis in Multiparty Systems

Diagnosis of problems in a multi party environment is particularly difficult because the infrastructure owned or accessible by each party is differently configured/managed and other parties have no visibility into or understanding of a party's infrastructure, configuration, or compatibility issues across parties. The multiparty diagnosis problem can be eased by each party providing a set of "probes" with well defined interfaces and cross-party access rights to the probes so that a multiparty test can be constructed out of such probes. Note that several parties may provide the same probing capability; this is quite normal since the available probing capability often depends on the location and vantage point of the party. In fact, a large organization often has multiple physical locations, each with different access capabilities; for example, a branch office may not have the same visibility and probing capability as the main office. For the purposes of this chapter, we shall consider such multiple locations as different "parties" with the access rules across our parties reflecting any organizations boundary considerations. In particular, if an organization has access to some probing capabilities, they will likely be available from multiple locations of the organization (which we are considering as different "parties" in our modeling).

The key to such multi-organizational and multi-location testing mechanism is a clear definition of semantics and format of the inputs and outputs of each probe such that they can be meaningfully connected across parties. This involves several issues that we discuss in the following, namely, how to build multiparty tests from party-specific probes, how to define cross-party access rights, and various questions that would interesting to study regarding the testing.

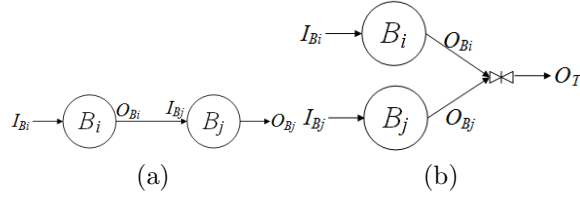


Figure 5.3: Combination of probes in (a) series, and (b) parallel.

5.3.1 Generating Tests from Probes

Consider parties P_i , and P_j , each of which define the probe B_i and B_j respectively. Let (I_{B_i}, O_{B_i}) and (I_{B_j}, O_{B_j}) denote the input/output of probes B_i and B_j respectively. Now consider a *requesting party* P_R that constructs a multiparty test (MPT) T out of these probes and runs it, eventually receiving the results.

The two obvious ways of combining the probes are: (a) **Series**, i.e., run B_i and use O_{B_i} to define I_{B_j} . Then run B_j and use O_{B_j} to define the test output O_T for P_R , and (b) **Parallel**, i.e., run B_i and B_j concurrently using I_{B_i} , O_{B_i} and use a some composition O_{B_i}, O_{B_j} to generate the test output O_T for P_R . This is depicted in Fig. 5.3. In the series case, I_{B_j} needs to be compatible with O_{B_i} , which means that (a) either both probes B_i and B_j are simple, or both are structured, and (b) it is possible to derive I_{B_j} from O_{B_i} .

For a parallel test, the crucial part is the “join” node, which represents some type of composition of the outputs (O_{B_i} and O_{B_j}) received from the preceding probes. In this case, we requires both outputs to be either simple or both structured. In case of structured outputs, this composition could be any of the standard relational operations such as equi-join, union, intersection, etc. In case of simply outputs, there is no explicit composition, both inputs are provided to the next probe.

In general, the test graph could be an arbitrary acyclic graph with a single source and single destination. However, such graphs become tricky with respect to I/O compatibility; furthermore, finding efficient ways of conducting tests with arbitrary structure becomes quite difficult. Therefore, in this chapter, we assume that the test

graphs are recursively composed of series or parallel subcomponents.

One other point in defining the tests is whether to specify the needed probes uniquely (e.g., via a unique id of the probe) or more generally. The constituent probes of a test are specified uniquely, the test construction problem is trivial (since there is no choice). A somewhat more general method is to specify probe in terms of its input, functionality and output (simply called “functionality” for short), rather than a unique id. This allows consideration of probes that have identical or superset of the requested functionality owned by various parties, and we will consider this as discussed later. It is possible to make the specification even more abstract, but we do not consider it here.

5.3.2 Defining Cross-Party Access Rights

The key attribute of the multiparty environment is the restrictions on cross party accesses. The access control can be specified at two levels: (a) access to the data, i.e., access to the output of a probe (or set of probes), and (b) access to the entire probe, meaning ability to give input to the probe, command its execution, and collect the output.¹ It is important to make data vs. probe access since it is possible to provide access to the output of a probe result to a party, without that party having the ability to actually run the probe. Also note that in case of a parallel arrangement of probes, the input to the next probe is derived from the composition of the outputs, only the access to this composition is crucial.

Several models are possible for connecting multiple probes with regard to access control, ranging from very strict to promiscuous. In a strict model, a party must have direct access to all the probes used in the test and also to their outputs. This is illustrated in Fig. 5.4(a) for a simple test with 3 probes (B1, B2, B3) in series, but similar situation applies in general. In this figure, we have an implicit assumption

¹Bundling probe and data access here is just for clarity in discussion; the detailed specification in section 5.4.1 actually defines the data and probe access functionalities separately.

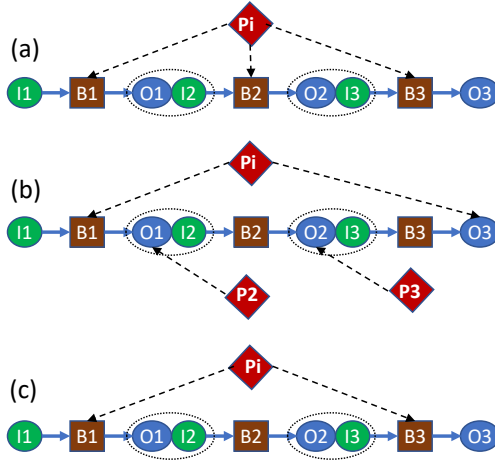


Figure 5.4: Illustration of Access Possibilities

that probes B_1, B_2, B_3 are owned by parties P_1, P_2 , and P_3 respectively, and the test is to be invoked by some other party P_i . the dotted ovals show compatibility between inputs and outputs, and this is always required. More important, the figure shows that the requesting party P_i must have access to all 3 probes and their inputs and outputs.

A considerably looser model would allow the requester to only possess end point access rights, as illustrated in Fig. 5.4(b). Here P_i only needs the ability to run probe B_1 and access the output O_3 . However, to move the access along, we need P_2 to have access to the output O_1 and P_3 should have access to O_2 . With this, the entire chain is completed, which means that the test can be run and result obtained by P_i .

Finally, Fig. 5.4(c) shows the loosest model. Here P_i needs the ability to run B_1 and collect result O_3 (as in case (b)); however, no other intermediate access restrictions are required to complete the test.

Many other possibilities also exist, such as granting access rights based on the results of the tests; however, these 3 cases illustrate the tradeoff between flexibility of testing and the protection/accessibility of the probes. Case (a) is easy to handle in determining the feasibility and minimality of the test; however, the restrictions it places are perhaps identical to access restrictions during normal functioning. We

believe that when problems arise, a looser access control model is more appropriate in order to grant the necessary accesses quickly. Case (c) is the easiest in this regard; however, it allows for many possibilities, which means that the question of minimal test construction becomes more interesting, and is in fact NP-hard, as discussed later.

5.3.3 Testing Issues

Now consider a requesting party P_R that has the rights to its desired inputs and outputs, and it wants to design a suitable MPT to obtain the output. Then, we can pose the following problems:

- Feasibility: Can we select the probes, probe graph, and probe I/O mappings for an MPT such that the output produced by the MPT is a superset of the output requested by P_R ?
- Additional Rights: If the problem is infeasible, what is the minimum additional access rights that P_R needs in order to construct a feasible MPT?
- Effectiveness/Cost: If there are multiple ways to define an MPT that satisfies P_R 's needs, how do we define a suitable measure of “effectiveness” or “Cost” that can be optimized?

The cost/effectiveness of the test can be defined in multiple ways. One simple definition, that we shall use in this chapter, is the involvement of minimum number of parties. However, one may instead want to minimize the number of probes, or the overall cost/overhead of running the test.

These problems can be considered as extended versions of the issues addressed in our earlier work Le et al. (2013a); Le, Kant, Athamnah, & Jajodia (2016); Athamnah & Kant (2016b). In particular, we have shown in Le et al. (2013a) that the rule enforceability problem (similar to the feasibility problem above) is NP-hard, and highly efficient and effective minimum cost enforcement algorithms are developed in Le,

Kant, Athamnah, & Jajodia (2016), along with further extensions and generalizations in Athamnah & Kant (2016b). Furthermore, we have considered the question of rule change and minimal extension of the rules in Le et al. (2012b). These approaches are applicable here as well, except for a significant additional complication that (a) the test is given as a graph, rather than a single rule to be enforced, (b) one needs to consider the input/output compatibility and corresponding access rights as well.

Because of these complications, we will not all these problems in this chapter. In particular, we do not consider the problem of adding minimal additional access rights to make the testing feasible. We also do not consider different variations in access rights requirements for running tests (discussed in section 5.3.2). Instead we only consider the most promiscuous case, where there may be many ways of satisfying the test.

5.4 Problem description and Solution Method

In this section we describe the problem of minimal cost test construction and show that the problem is quite complex and easily shown to be NP-Hard. We then describe a heuristic solution.

5.4.1 Preliminaries and notations

Assume that there is a set of parties $\mathbb{P} = \{P_1, P_2, \dots, P_p\}$. Also assume that there are n probes $\mathbb{F} = \{F_1, F_2, \dots, F_n\}$, each one can be provided/accessed by multiple parties. Each party P_i provides a set of probes $B_i \subseteq \mathbb{F}$. Each probe F_i can take a set of inputs I_i and produces an output O_i . For example, a probe can be a DNS lookup that takes an input type *domain names* and produces an output that is a numerical *IP address* corresponding to a computer service or device.

As each probe is accessed by multiple parties, we will have multiple, possibly overlapping probes across parties. Also, the parties can give controlled access to

specific data from their provided probes to one another. The specific access of each probe is governed by an explicit set of rules. The rules are of two different types: *run access* and *data access*.

The run access \mathbb{R} is described by 2-tuple, $\mathbb{R}(P_i, F_j)$, means party P_i is allowed to run probe F_j using an arbitrary valid input. The data access \mathbb{D} is also described by a 2-tuple, $\mathbb{D}(P_i, O_j)$, which indicates that party P_i is allowed to access the data O_j (i.e., the output of probe F_j). The *success* function is the result of the combination of the run access and data access, which means that a party can successfully execute a probe. Thus a success function \mathbb{S} can be described by 3-tuple, $\mathbb{S}(P_i, I_j, O_j)$, which means that the party P_i can successfully execute probe F_j with input I_j and obtain output O_j . This is denoted as $P_i \rightarrow F_j$.

We also have a controller C who can communicate with all parties. The controller has multiple roles. The controller figures out which probes from other parties can serve particular party's test, and what are the required input types and values for the probes. It also provides the best routing plan for specific test through all available probes. The controller can be thought of as a party that can be trusted to perform these functions correctly and reliably.

We also assume that there are some input/output compatibility relationships across the probes. That is, a probe F_j 's input is compatible with F_i 's output (denoted as $F_i \rightarrow F_j$), if F_j 's input can be derived from F_i 's output, i.e. $I_j \subseteq O_i$.

If a party P_R wants to run a test, it first sends a request to the controller C with all the required information. C then provides the plan for the test to P_R after considering the access rights across different parties along with a one-time certificate (OTC) for accessing the probes across other parties. The OTC comes with a timeout and can be used by P_R only for carrying out that specific test. In this chapter we assume that given a test from P_R , the controller C finds out the test plan that involves minimum number of parties needed to be involved for conducting the test, which we define as

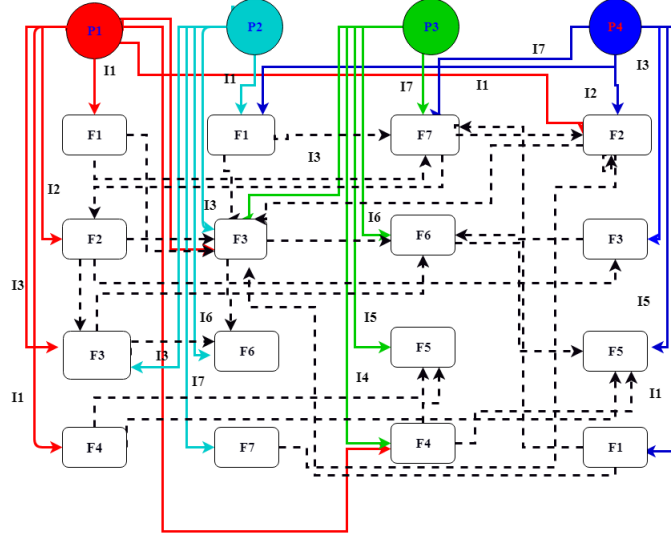


Figure 5.5: Controller graph

minimum party testing (MPT) problem.

5.4.2 Problem Definition and Complexity of MPT

Given the access rights across the parties, and the input/output compatibility across the probes, the controller C generates a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of directed edges. Here V consists of the set of parties and probes, i.e. $V = (\mathbb{P}, \mathbb{F})$. E consists of the access rights of the parties, and the input/output compatibility across the probes, i.e. $E = (P_i \rightarrow F_j, F_j \rightarrow F_k) \forall P_i \in \mathbb{P}$ and $F_j, F_k \in \mathbb{F}$. An example of the graph G is depicted in Fig. 5.5. In Fig. 5.5, the graph G consists of four parties P_1, P_2, P_3 , and P_4 , each one has access to all 4 probes (shown in solid colored lines). For example, P_1 has access to $F_1 - F_4$. Furthermore, a probe can be accessed by multiple parties. For example, F_3 is accessed by P_1, P_2 and P_4 . The input/output compatibility across the probes are also shown by dashed lines. For example, $F_1 \rightarrow F_3, F_3 \rightarrow F_6$ and so on.

A test can also be represented by a graph $\mathcal{T} = (\mathcal{T}_v, \mathcal{T}_e)$, where \mathcal{T}_v and \mathcal{T}_e represent a set of vertices and edges of \mathcal{T} . In a test graph \mathcal{T}_v represents the set of probes and \mathcal{T}_e represents the input/output compatibility across the probes. An example of a test

graph is illustrated in Fig. 5.6, where $\{a, b, c, d, e\}$ is the set of probes and the edges in between them represent the input/output compatibility across the probes.

Given a test graph P_R , the controller first checks whether \mathcal{T} is a legitimate test of not. The condition for checking a legitimate test is described in section 5.4.3. Once a test \mathcal{T} is found to be legitimate, C then finds the test plan for conducting the test by solving the MPT problem. Below we first prove that the MPT problem is NP-hard. For this proof we assume a special case of the MPT problem where $\mathcal{T} \subseteq G$.

Theorem 5.1. *The MPT problem for given test is NP-hard.*

Proof. For this proof we reduce the well-known set cover problem to MPT. The set cover problem can be described as follows. Assume an universe \mathbb{U} that consists of a set of n elements, i.e. $\mathbb{U} = \{A_1, A_2, \dots, A_n\}$. Also assume that \mathcal{S} is a collection of sets, i.e. $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ where S_i is a set of elements from \mathbb{U} . Given the input pair $(\mathbb{U}, \mathcal{S})$ the minimum set cover problem is to find out the minimum subfamily $\mathbb{C} \subseteq \mathcal{S}$ whose union is \mathbb{U} .

Assume a legitimate test graph $\mathcal{T} = (\mathcal{T}_v, \mathcal{T}_e)$. Also assume that $\mathbb{F}' = \{\mathcal{T}_v \cap B_1, \mathcal{T}_v \cap B_2, \dots, \mathcal{T}_v \cap B_p\}$ represents the derived set of probes from original probes \mathbb{F} that only includes \mathcal{T} 's probes corresponding to the parties $\mathbb{P} = \{P_1, P_2, \dots, P_p\}$. Now the task is to find out the minimum $\mathbb{C}' \subseteq \mathbb{F}'$ whose union is \mathcal{T} . This is identical to find the MPT for conducting a given test. To map the set cover problem to MPT we construct a universe \mathbb{U} which consists of the vertices of \mathcal{T} . We also construct \mathcal{S} which is identical to \mathbb{F}' . Now the problem becomes how to find minimum number of sets in \mathcal{S} that cover all probes in \mathbb{U} . In such case, if the MPT can be found in polynomial time, the set covering problem also has a polynomial solution. Thus, the MPT problem is NP-hard. \square

Fig. 5.6 shows an example test graph with $\mathcal{T} \subseteq G$. Assume that P_1, P_2, P_3 and P_4 have accesses to the probes $\{a, b, c\}$, $\{b, e\}$, $\{c, e\}$ and $\{d, e\}$. Thus C finds the

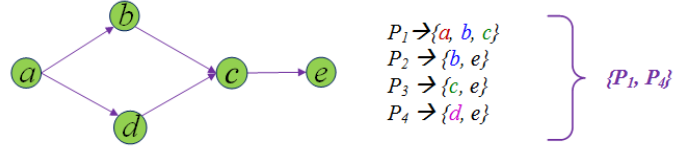


Figure 5.6: Illustration of the MPT problem.

Algorithm 7 Greedy set cover (\mathbb{U}, \mathbb{S})

- 1: $\mathbb{C} = \phi$
 - 2: **while** $\mathbb{U} \neq \phi$ **do**
 - 3: Pick $S_i \in \mathbb{S}$ that maximizes $|\mathbb{S} \cap \mathbb{U}|$;
 - 4: $\mathbb{C} = \mathbb{C} \cup S_i$;
 - 5: $\mathbb{U} = \mathbb{U} - S_i$;
 - 6: **return** \mathbb{C} ;
-

minimum number of sets (i.e. parties), union of which constructs the test graph \mathcal{T} , by solving the minimum set cover problem. For Fig. 5.6 we can observe that by involving P_1 and P_4 , the controller can conduct \mathcal{T} . For solving the minimum set cover problem, we use a greedy heuristic shown in Algorithm 7. The heuristic selects the party containing the maximum number of uncovered probes at each step, until all the probes in the test graph are covered.

5.4.3 Proposed solution

In the previous subsection we assume that a test \mathcal{T} is a legitimate test if $\mathcal{T} \subseteq G$. Considering the test graph of Fig. 5.6, b has input/output compatibility with a in G , which we denote as $(a \rightarrow b) \in G$. In this section we relax this condition to generalize the test model by using the *transitive* relations, i.e.

$$(x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge \dots \wedge (x_{n-1} \rightarrow x_n) \implies (x_1 \rightarrow x_n)$$

which we define as *transitive input/output compatibility (TIOC)*. To check whether \mathcal{T} is a legitimate test or not, C checks two conditions. First it checks whether the requesting party P_R has access to the initial and terminal probes of \mathcal{T} , so that it can

pass the initial input and access the final output of the test. (In Fig. 5.6, these are a and e respectively.) In addition, we also need to ensure that for each $(x \rightarrow y) \in \mathcal{T}$, there is a TIOC in between x and y in G , i.e. there exists a *path* from x to y in G . For example in Fig. 5.6 to ensure the test graph to be legitimate, there needs to be a path in between a to b , a to d , b to c , d to c and c to e in G .

If the test is legitimate, we need to find the minimum number of parties that the controller C needs to involve for conducting the test. In view of the NP-hardness result, we develop a greedy algorithm explained in Algorithm 8. For finding the minimum number of parties needed for conducting the test, the controller C first computes the K shortest paths in between the initial and terminal nodes. We use Yen's algorithm *K-Shortest Path* (n.d.) to find out the K shortest paths (line 7). For each path $\Gamma_i \in \Gamma$, C records the edges of the test graph whose TIOC is satisfied by the Γ_i . It then generates all possible sets of Γ_i 's whose union covers all edges of the test graph.

As an example, in Fig. 5.6, the union of two paths $\Gamma_1 = a \rightarrow b \rightarrow c \rightarrow e$ and $\Gamma_2 = a \rightarrow d \rightarrow c \rightarrow e$ will cover the test graph \mathcal{T} . Another path $\Gamma_3 = a \rightarrow b \rightarrow d \rightarrow c \rightarrow e$ will also satisfy the TIOC conditions, thus this can also be a valid set to cover \mathcal{T} . In reality finding all possible sets of Γ_i 's for covering the test graph's edges can be exponential; therefore, we remove the paths from Γ once they are chosen by the set cover algorithm (line 21). We repeat the process until all Γ is empty or there is there is no possible combination of Γ_i 's that covers the \mathcal{T} (line 8-22).

For all these possible set of Γ_i 's whose union covers \mathcal{T} , we find the minimum number of parties to run these set of probes by solving the minimum set cover problem (line 20). We then choose the set of Γ_i 's that can be conducted by involving the minimum number of parties (line 24).

Algorithm 8 Finding the minimum number of parties $\text{MPT}(\mathcal{T}, G)$

```
1:  $P_R$  : Requesting party in  $\mathcal{T}$ ;  
2:  $\mathcal{S}$  : Initial probe in  $\mathcal{T}$ ;  
3:  $\mathcal{D}$  : Terminal probe in  $\mathcal{T}$ ;  
4: if  $P_R$  has access to  $\mathcal{S}$  and  $\mathcal{D}$  then  
5:   Min_Party = null;  
6:    $\Gamma = \text{Yen\_Algorithm}(\mathcal{S}, \mathcal{D}, G, K)$ ;  $\triangleright$  Generate  $K$  shortest paths  
7:   while  $\Gamma \neq \text{null}$  &&  $\text{Set\_Cover}(\mathcal{T}, \text{Paths}) \neq \text{null}$  do  
8:     Selected_Paths =  $\text{Set\_Cover}(\mathcal{T}, \Gamma)$ ;  $\triangleright$  Cover TIOC conditions  
9:     Party_subsets = NULL;  
10:    for each party  $P_i$  in  $G$  do  
11:      Party_probes = NULL  
12:      for each probe  $F_j$  that is accessed by  $P_i$  do  
13:        if  $F_j$  in Selected_Path then  
14:          Party_probes = Party_probes  $\cup F_j$ ;  
15:          Party_subsets = Party_subsets  $\cup$  Party_probes;  
16:          Min_Party = Min_Party  $\cup$  Set_Cover( $\mathcal{T}$ , Party_subsets)  
17:           $\Gamma = \Gamma - \text{Selected\_Paths}$ ;  
18: return the set of parties having minimum cardinality in Min_Party;
```

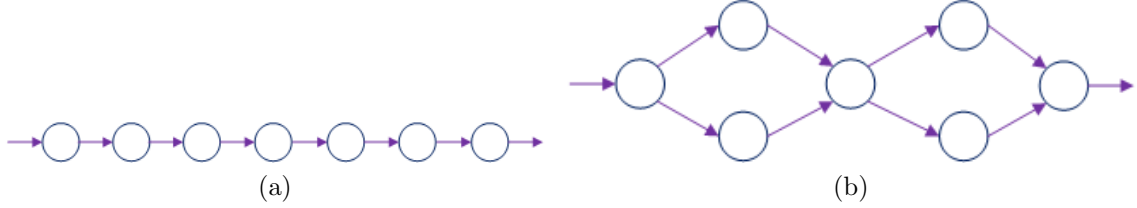
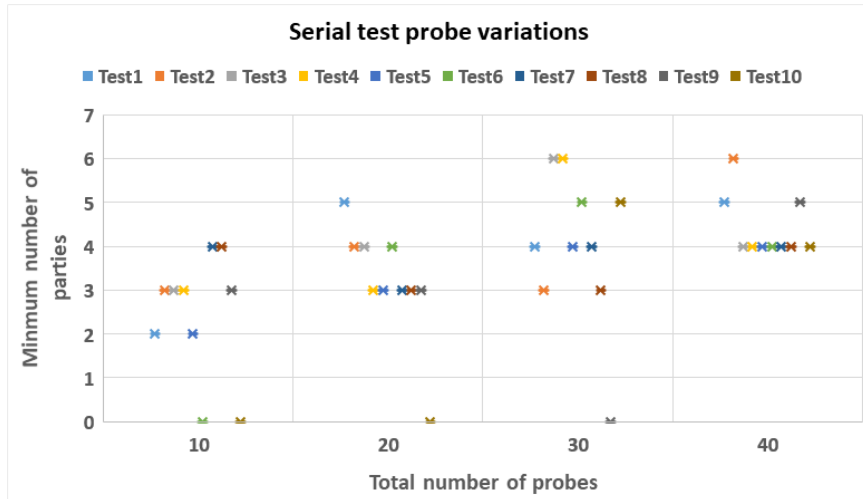


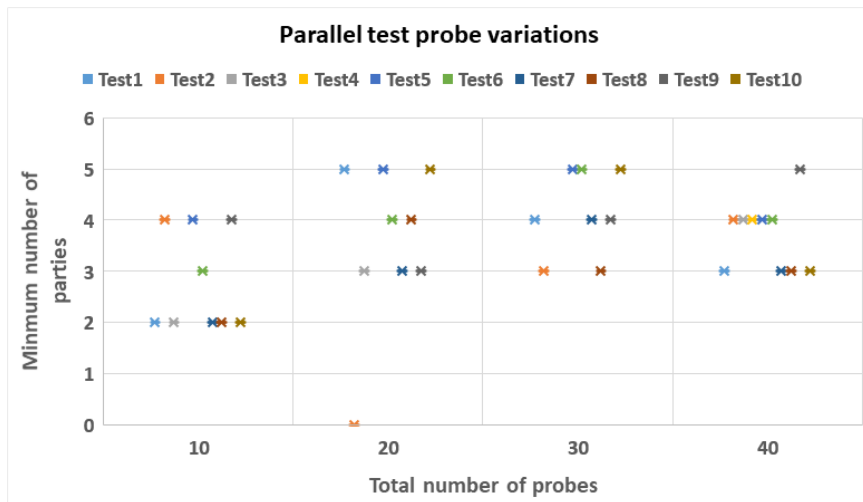
Figure 5.7: Illustration of the (a) serial and (b) parallel test graphs.

5.5 Performance evaluation

For the experimental evaluation, we generate the graph G and the test graph \mathcal{T} randomly. For constructing G , the access rights for parties towards the probes are generated uniformly randomly with a probability of 0.7. The input/output compatibility among the probes are also generated with a probability of 0.7. The test graphs consist of 7 distinct probes (as shown in Fig. 5.7) that are generated uniformly randomly among the number of available probes. Unless otherwise stated, K (the maximum number of paths) is assumed to be 50 for the experiments.



(a)

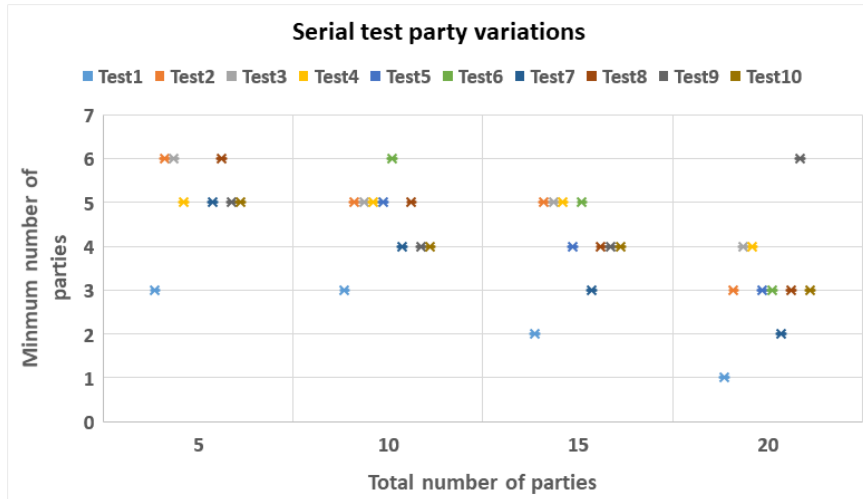


(b)

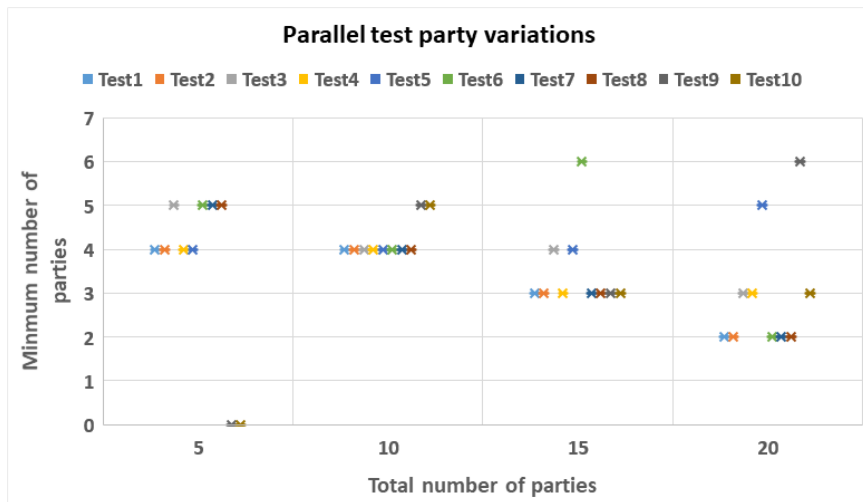
Figure 5.8: Comparison of the minimum number of parties with different number of probes in G for (a) serial and (b) parallel test graphs.

5.5.1 Effect of total number of probes

Fig. 5.8 shows the variation of the minimum number of parties with different number of probes. In Fig. 5.8 we assume 10 parties each having access to 5 probes. We randomly generate 10 different test cases and find out the minimum number of parties in each case. From Fig. 5.8 we can observe that the minimum number of parties almost doubles when the number of probes increases from 10 to 40. This is because as the total number of probes increases, each probe is accessed by fewer



(a)



(b)

Figure 5.9: Comparison of the minimum number of parties with different number of parties in G for (a) serial and (b) parallel test graphs.

parties. This results in an increasing number of parties required to conduct the tests.

Also notice that the number of parties for conducting the tests are similar for both serial and parallel cases.

5.5.2 Effect of total number of parties

Fig. 5.9 shows the variation in the number of parties required to conduct the tests with different number of parties. The total number of probes is assumed to be 15. Each party has access to 5 probes. From Fig. 5.9 we can observe that the number of

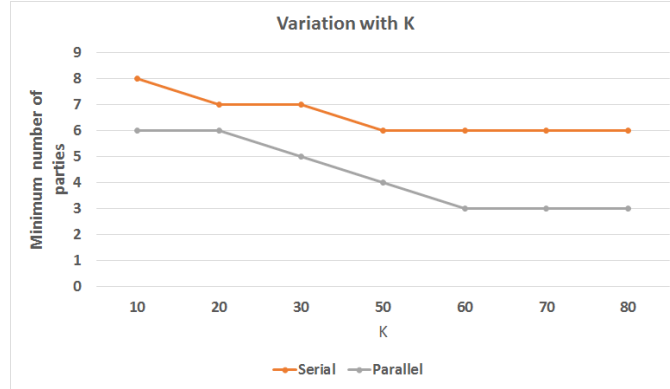


Figure 5.10: Variation of number of parties required for conducting the tests with different K .

parties for conducting the tests decreases approximately from 4 to 2 on the average when the total number of parties increase from 5 to 20. This is because as the total number of parties increases, MPT generates shorter paths between the initial and terminal probes, which results in fewer required parties for conducting the tests. The number of parties for conducting the tests does not change significantly for serial and parallel test cases.

5.5.3 Effect of K

Fig. 5.10 shows the minimum number of parties involved in conducting two arbitrary serial and parallel test cases with increasing K . For Fig. 5.10 the total number of parties are assumed to be 10, each one having 5 probes. The total number of probes is assumed to be 30. From Fig. 5.10 we can observe that the required number of parties decreases with increasing K and then saturates beyond a certain point. This is because the MPT scheme generates more paths from the initial probe to the terminal probe as K increases, which provides more options for finding the minimum number of required parties for conducting the tests. Notice that in Fig. 5.10 the specific test cases for the serial and parallel are chosen arbitrarily, and thus the relative gap between the serial and parallel test cases is not important.

5.6 Related Work

Change management is an active and well-studied subject in the literature and can be helpful in tracking the impact of misconfigurations and allowing for a fix. The general idea is to record the system behavior as a function of various configuration parameters as the system evolves. These records can be exploited to assess what combinations of parameters lead to “good” or “bad” behavior. Evolgen Kaluza (n.d.) provides many tools based on such an approach that use machine learning techniques to determine normal and abnormal patterns. The STRIDER tool reported in Y.-M. Wang et al. (2004) applies a black box approach that assumes an underlying “statistical golden state” and uses a Hidden Markov Model based estimation procedure to detect and fix problems. Many other tools also exist such as Glean Kiciman & Wang (2004), PeerPressure H. J. Wang et al. (2004), etc. However, there are no tools that tackle the multiparty environment.

Although such automated approaches can be useful in identifying and even fixing problems, they generally apply to specific types of errors and also suffer from substantial false alarms. In particular, such approaches are limited by the comprehensiveness of the collected data and the nature of dependencies. For example, if the misbehavior happens for certain combination of parameters which cannot be deduced from the available data, the approach is not useful. In short, there are still numerous instances where human involvement is essential to identify what tests might be most useful to get to the bottom of the problem. This chapter concerns building an infrastructure that can automate constructing and selecting tests.

Another diagnosis approach is to define processes based on best practices such as ITIL (Information Technology Infrastructure Library), which are then automatically managed by incident management systems such as IBM’s Tivoli Service Request Manager I. Corp. (n.d.) and BMC Remedy Service Suite B. Corp. (n.d.). The known problems can be diagnosed automatically but others are left to administrators. Yet

another method is to define a set of configurations rules which can be checked automatically. EnCore uses this approach to detect software misconfigurations J. Zhang et al. (2014). This approach is simple but applies to only very specific cases.

Reference Pasquale et al. (2009) proposes a REST based infrastructure to provide restricted exposure of configuration across parties. This work can be considered somewhat similar to what we are proposing. The work of Pasquale et al. (2009) has discussed a configuration management infrastructure named REST in the context of a multi-domain, enterprise Web services. They have proposed a decentralized configuration change management architecture, which takes into account the distribution of configuration information and facilitates the execution of management processes across organizational boundaries while maintaining each organization's autonomy. Social media activities such as photo sharing experiences multi-party conflict scenarios too. Mechanisms for resolving such issues have been studied in Such & Criado (2016) and could be useful for considering complex constraints.

5.7 Conclusion

Misconfigurations in large scale cyber systems are known to be responsible for an overwhelming percentage of failures, poor service, and exploitation by hackers for cyber-attacks. In this chapter, we explored a comprehensive, multi-party infrastructure to assist in the diagnosis by considering the access rights across different parties. We proposed an efficient way of finding out the minimum number of parties involved in conducting different multi-party test cases via valid connection of different probes (along with suitable access control) to allow for a flexible multiparty probing mechanism. We are currently building a comprehensive testing infrastructure for multi-party testing using Common Open Research Emulator (CORE) Ahrenholz et al. (2008); Ahrenholz (2010) package. This package can support multiple, geographically separated data centers with intervening connection layers (e.g., DMZ, WAN edge,

WAN, etc) and will be useful in gaining further insights into the real misconfiguration problems. We will also examine access control models other than the promiscuous model used here. Finally, we will examine minimization objectives other than the number of parties and understand how sensitive the test construction is to various types of objectives.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we consider the scenarios that require different parties and enterprises to cooperate with each other to perform computations and meet business requirements. Each of these enterprises owns and manages its data independently using a private cloud, and these parties need to selectively share some information with one another. We consider the authorization model where authorization rules are used to constrain the access privileges based on the results of join operations over relational data. We can interpret the authorization rules in two ways. With implicit authorization, we presented an efficient algorithm to decide whether a given query can be authorized using the join properties among the given rules. Under the explicit semantic, access conflicts may arise among the rules made according to business requirements. Therefore, we proposed a mechanism to make the set of cooperative authorization rules consistent. In addition, we also presented algorithms to maintain the rule consistency in the case of granting and revocation of access privileges. To prevent undesired computation results, negative rules are introduced. We proposed an algorithm to check whether the given authorization rules will violate the negative rules. We proposed an algorithm that uses a constructive method to check the rules in a bottom-up manner based on the number of the relations involved, and the mechanism finds all the information that can be enforced among existing collaborating parties. There are some rules cannot be enforced due to the conflicts with the negative rules, and we give a mechanism to weaken the rules so as to make them

enforceable with no conflict. Since these problems are NP-hard, and we proposed greedy algorithms to generate solutions. With extensive simulation evaluations, we concluded that our algorithms were efficient and the solutions were close to the optimal ones. we also considered different data models to apply our authorization model. Therefore, we studied the problem of executing queries on multiparty repositories where the queries need to retrieve tuples based on similarity function defined via graphs. Such queries are extremely useful in providing additional information to the user based on the graph relationships that may represent similarities between products, relationships between resources, etc. We presented an algorithm to find the similarity using tuples scanning mechanism. The algorithm is further enhanced by exploiting dominating attributes and partitioning when possible. Finally, we apply our authorization model on a real problem, therefore we consider a misconfiguration in large scale cyber systems and we proposed a framework to control the diagnosis process for system failures cases. We proposed an efficient way of finding out the minimum number of parties involved in conducting different multi-party test cases via valid connection of different probes (along with suitable access control) to allow for a flexible multiparty probing mechanism.

6.2 Future work

In the future, it is possible to look into the more dynamic situation where dynamic changes to the model queries and/or safety properties, and hence the rules. Normally, changes are infrequent and driven by changing business needs, but in some applications, rules may be tied to the phase of operation. We have explored efficient incremental algorithms to handle certain change scenarios, but a more comprehensive treatment (e.g., change in type of access, rule-interdependencies, set of parties, non-relational databases, etc.) remain to be examined. We assume the collaborating parties first make the rules via negotiations, and then check whether a query is authorized

and the safe ways to answer the query. It is possible to consider reversing the process. That is, we may want to figure out the complete set of queries that should be answered to meet business requirements, and after that we design authorization rules for cooperative parties so that only these wanted queries can be answered. However, due to the local computation, we may authorize extra information when granting privileges for this set of queries. Thereby, the problem becomes to figure out the best way of making rules so that minimal amount of extra information will be released together with the rules. We assume the rules are not changing very frequently. In some scenarios, the permissions as well as the data may change in a fast pace. An authorization rule given to a party may only apply to a short period of time. After certain period of time, the authorization no longer exists. Since data is also changing frequently, it will become useless after some time. In such environments, the authorization rule can be granted dynamically based on the demands. For instance, if there is a query need to be authorized and answered, then for each step to process this query request, we can grant permissions to authorize the operation on the fly. Once such a step is executed, the authorizations are revoked. This is similar to the workflow scenario. By granting privileges for a short time period, the extra information that is obtainable via local computation can be limited. The challenging problem becomes finding a way to schedule the queries as well as the time points to grant the authorizations so that minimal amount of extra information is released. Our current model does not assume any malicious insiders and all the parties are expected to strictly follow the given authorization rules. Nevertheless, this may not be the case in practice. A party may not behave honestly during the collaboration. For instance, a party may be authorized to obtain some information from a data owner, then it may leak this information to some unauthorized parties. To give another example, a party receives data from the data owner and it needs to send the data to another party according to the generated query plan, the party may change some of data it should transfer or

it can simply choose not to send all the required data. Thus, it is required to have a mechanism that can verify the integrity of the received data. One possibility is to use the existing mechanisms such as hash values, merkle trees and signatures to ensure the data integrity. Considering the properties in the collaboration environment, it may be possible to check the data integrity through collaboration. In cooperative data access, there may exist more than one legitimate data transmission path beginning from the data owner to the authorized party. Therefore, parties can exchange the information they have. By doing that, if the number of misbehaving parties is limited, it is very possible to detect them. Furthermore, existing mechanisms such as reputation systems and trust management can be considered to ensure the data integrity in the cooperative data sharing environment. Last but not least, we want to implement and evaluate all the proposed algorithms and mechanisms in real world scenarios.

BIBLIOGRAPHY

- Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., ... Xu, Y. (2005). Two can keep A secret: A distributed architecture for secure database services. In *Cidr* (pp. 186–199). Retrieved from <http://www.cidrdb.org/cidr2005/papers/P16.pdf>
- Agrawal, R., Asonov, D., Kantarcioglu, M., & Li, Y. (2006). Sovereign joins. In *Proceedings of the 22nd international conference on data engineering, ICDE 2006, 3-8 april 2006, atlanta, GA, USA* (p. 26). IEEE Computer Society. Retrieved from <http://doi.ieeecomputersociety.org/10.1109/ICDE.2006.144>
- Ahrenholz, J. (2010). Comparison of core network emulation platforms. In *Military communications conference, 2010-milcom 2010* (pp. 166–171).
- Ahrenholz, J., Danilov, C., Henderson, T. R., & Kim, J. H. (2008). Core: A real-time network emulator. In *Military communications conference, 2008. milcom 2008. ieee* (pp. 1–7).
- Álvarez, S., Brisaboa, N. R., Ladra, S., & Pedreira, Ó. (2010). A compact representation of graph databases. In *Proceedings of the eighth workshop on mining and learning with graphs* (pp. 18–25).
- Andoni, A., & Indyk, P. (2008, Jan). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*.
- Athamnah, M., & Kant, K. (2016a). Generalized inter-cloud structured data sharing. In *Cloud computing technology and science (cloudcom), 2016 ieee international conference on* (pp. 198–205).
- Athamnah, M., & Kant, K. (2016b, Dec). Multiparty database sharing with generalized access rules. In *Proc. of cloudcom, luxemburg* (p. 198-205).
- Athamnah, M., & Kant, K. (2017). Graph property augmented queries in collaborative databases. In *31st data and applications security and privacy, dbsec 2017*. (submitted)
- Athamnah, M., Pal, A., & Kant, K. (2018). A framework for misconfiguration diagnosis in interconnected multiparty systems..

- Barroso, L. A., Clidas, J., & Hölzle, U. (2013). The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3), 1–154.
- Ben Arbia, D., Alam, M. M., Kadri, A., Ben Hamida, E., & Attia, R. (2017). Enhanced iot-based end-to-end emergency and disaster relief system. *Journal of Sensor and Actuator Networks*, 6(3). Retrieved from <http://www.mdpi.com/2224-2708/6/3/19> doi: 10.3390/jsan6030019
- Bernstein, P. A., Goodman, N., Wong, E., Reeve, C. L., & Rothnie, Jr., J. B. (1981, December). Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4), 602–625.
- Brewer, D. F. C., & Nash, M. J. (1989). The chinese wall security policy. In *Ieee symposium on security and privacy* (pp. 206–214).
- Butler, K., Farley, T. R., McDaniel, P., & Rexford, J. (2010, Jan). A survey of bgp security issues and solutions. *Proceedings of the IEEE*, 98(1), 100-122. doi: 10.1109/JPROC.2009.2034031
- Byun, J.-Y., Nasridinov, A., & Park, Y.-H. (2014). Internet of things for smart crime detection. *Contemporary Engineering Sciences*, 7(15), 749–754.
- Caesar, M., & Rexford, J. (2005, Nov). Bgp routing policies in isp networks. *IEEE Network*, 19(6), 5-11. doi: 10.1109/MNET.2005.1541715
- Calì, A., & Martinenghi, D. (2008). Querying data under access limitations. In *Proceedings of the 24th international conference on data engineering, ICDE 2008, april 7-12, 2008, cancún, México* (pp. 50–59). IEEE.
- Cappelli, W. (2015, Oct). Causal analysis makes availability and performance data actionable. *Gartner Report G00273922*.
- Chaudhuri, S. (1998). An overview of query optimization in relational systems. In *Proceedings of the 7th acm sigact-sigmod-sigart symposium on principles of database systems* (pp. 34–43).
- Ciriani, V., di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., & Samarati, P. (2009). Keep a few: Outsourcing data while maintaining confidentiality. In *Esorics* (Vol. 5789, pp. 440–455). Retrieved from <http://dx.doi.org/10.1007/978-3-642-04444-1>
- Connolly, F. (2014, Mar). Production operations – the last mile of a devops strategy. *LMC Report*.
- Corp., B. (n.d.). *Bmc remedy9 service suite*. Retrieved from www.bmc.com/it-solutions/remedy-itsm.html

- Corp., I. (n.d.). *Ibm tivoli service request manager*. Retrieved from www-03.ibm.com/software/products/en/servicerequestmanager/
- Damiani, E., di Vimercati, S. D. C., Paraboschi, S., & Samarati, P. (2004). An open digest-based technique for spam detection. *ISCA PDCS, 2004*, 559–564.
- De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., & Samarati, P. (2008, June). Controlled information sharing in collaborative distributed query processing. In *Icdcs 2008* (pp. 2–7). Beijing, China.
- di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., & Samarati, P. (2011). Authorization enforcement in distributed query evaluation. *Journal of Computer Security, 19*(4), 751–794. Retrieved from <http://dx.doi.org/10.3233/JCS-2010-0413>
- Elliot, S. (2014). Devops and the cost of downtime: Fortune 1000 best practice metrics quantified. *International Data Corporation (IDC)*.
- Goldstein, J., & Larson, P. (2001). Optimizing queries using materialized views: a practical, scalable solution. In *Proceedings of the 2001 acm sigmod international conference on management of data* (pp. 331–342).
- Gope, P., & Hwang, T. (2016, March). Bsn-care: A secure iot-based modern health-care system using body sensor network. *IEEE Sensors Journal, 16*(5), 1368–1376. doi: 10.1109/JSEN.2015.2502401
- Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems, 29*(7), 1645 - 1660. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0167739X13000241> doi: <https://doi.org/10.1016/j.future.2013.01.010>
- Halevy, A. Y. (2001). Answering queries using views: A survey. *VLDB Journal, 10*(4), 270–294.
- Hwang, K., & Li, D. (2010). Trusted cloud computing with secure resources and data coloring. *IEEE Internet Computing, 14*(5), 14–22.
- Jajodia, S., Kant, K., Samarati, P., Singhal, A., Swarup, V., & Wang, C. (2014). *Secure cloud computing*. Springer.
- Kaluza, B. (n.d.). *Top 5 it ops challenges and how machine learning can help*. Evolven Corp.
- Kant, K., & Pal, A. (2017, Jan). Internet of perishable logistics. *IEEE Internet Computing, 21*(1), 22–31. doi: 10.1109/MIC.2017.19
- Kiciman, E., & Wang, Y.-M. (2004). Discovering correctness constraints for self-management of system configuration. In *Autonomic computing, 2004. proceedings. international conference on* (pp. 28–35).

- Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3, 91–97.
- Kossmann, D. (2000a). The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4), 422–469. Retrieved from <http://doi.acm.org/10.1145/371578.371598>
- Kossmann, D. (2000b). The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4), 422–469.
- K-shortest path.* (n.d.). Retrieved from <http://www.mathworks.com/matlabcentral/fileexchange/32513-k-shortest-path-yen-s-algorithm>
- Kulis, B., et al. (2013). Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4), 287–364.
- Le, M., Kant, K., Athamnah, M., & Jajodia, S. (2016). Minimum cost rule enforcement for cooperative database access. *Journal of Computer Security*, 24(3), 379-403.
- Le, M., Kant, K., Athamnah, M., & Jajodia, S. (2016, June). Minimum cost rule enforcement for cooperative database access. In *Journal of computer security*.
- Le, M., Kant, K., & Jajodia, S. (2012a, oct.). Access rule consistency in cooperative data access environment. In *8th international conference on collaborative computing: Networking, applications and worksharing (collaboratecom2012)* (p. 11-20).
- Le, M., Kant, K., & Jajodia, S. (2012b, oct.). Access rule consistency in cooperative data access environment. In *8th international conference on collaborative computing: Networking, applications and worksharing (collaboratecom2012)* (p. 11-20).
- Le, M., Kant, K., & Jajodia, S. (2013a, Nov.). Consistency and enforcement of access rules in cooperative data sharing environment. In *Computers and security* (p. 10-12).
- Le, M., Kant, K., & Jajodia, S. (2013b, July). Rule enforcement with third parties in secure cooperative data access. In *27th data and applications security and privacy, dbsec 2013* (p. 3-6).
- Le, M., Kant, K., & Jajodia, S. (2014, July). Consistent query plan generation in secure cooperative data access. In *28th data and applications security and privacy, dbsec 2014* (p. 10-12).
- Li, C. (2003). Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3), 211–227.

- Linden, G., Smith, B., & York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1), 76–80.
- Linden, G. D., Jacobi, J. A., & Benson, E. A. (2001, July 24). *Collaborative recommendations using item-to-item similarity mappings*. Google Patents. (US Patent 6,266,649)
- Maneth, S., & Peternek, F. (2015). A survey on methods and systems for graph compression. *arXiv preprint arXiv:1504.00616*.
- Mano, L. Y., Faial, B. S., Nakamura, L. H., Gomes, P. H., Libralon, G. L., Meneguete, R. I., ... Ueyama, J. (2016). Exploiting iot technologies for enhancing health smart homes through patient identification and emotion recognition. *Computer Communications*, 89(Supplement C), 178 - 190. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0140366416300688> (Internet of Things Research challenges and Solutions) doi: <https://doi.org/10.1016/j.comcom.2016.03.010>
- Mei, T., Rui, Y., Li, S., & Tian, Q. (2014). Multimedia search reranking: A literature survey. *ACM Computing Surveys (CSUR)*, 46(3), 38.
- Metcalf, D., Milliard, S. T. J., Gomez, M., & Schwartz, M. (2016, Sept). Wearables and the internet of things for health: Wearable, interconnected devices promise more efficient and comprehensive health care. *IEEE Pulse*, 7(5), 35-39. doi: 10.1109/MPUL.2016.2592260
- Microsoft. (n.d.). *Adventureworks sample databases*. Retrieved 03.19.2017, from [http://technet.microsoft.com/en-us/library/ms124501\(v=sql.100\).aspx](http://technet.microsoft.com/en-us/library/ms124501(v=sql.100).aspx)
- Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340), 2.
- Muja, M., & Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11), 2227–2240.
- Oliver, J., Cheng, C., & Chen, Y. (2013). Tlsh—a locality sensitive hash. In *Cybercrime and trustworthy computing workshop (ctc), 2013 fourth* (pp. 7–13).
- Oppenheimer, D., Ganapathi, A., & Patterson, D. A. (2003). Why do internet services fail, and what can be done about it? In *Usenix symposium on internet technologies and systems* (Vol. 67).
- Pasquale, L., Laredo, J., Ludwig, H., Bhattacharya, K., & Wassermann, B. (2009). Distributed cross-domain configuration management. *Service-Oriented Computing*, 622–636.
- Pottinger, R., & Halevy, A. Y. (2001). Minicon: A scalable algorithm for answering queries using views. *VLDB J*, 10(2-3), 182–198.

- Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 acm conference on computer supported cooperative work* (pp. 175–186).
- Rizvi, S., Mendelzon, A., Sudarshan, S., & Roy, P. (2004, June). Extending query rewriting techniques for fine-grained access control. In *Proceedings of the acm intl conf on management of data(sigmod'04)* (p. 3-6).
- Roussev, V. (2010). Data fingerprinting with similarity digests. In *Ifip international conference on digital forensics* (pp. 207–226).
- Sagiroglu, S., Terzi, R., Canbay, Y., & Colak, I. (2016, Nov). Big data issues in smart grid systems. In *2016 ieee international conference on renewable energy research and applications (icrera)* (p. 1007-1012). doi: 10.1109/ICRERA.2016.7884486
- Shahin, M. (2015). Architecting for devops and continuous deployment. In *Proceedings of the aswec 2015 24th australasian software engineering conference* (pp. 147–148).
- Sikos, L. F. (2015). Linked open data. In *Mastering structured data on the semantic web: From html5 microdata to linked open data* (pp. 59–77). Berkeley, CA: Apress. Retrieved from https://doi.org/10.1007/978-1-4842-1049-9_3 doi: 10.1007/978-1-4842-1049-9_3
- Sion, R. (2005). Query execution assurance for outsourced databases. In *Vldb* (pp. 601–612). ACM. Retrieved from <http://www.vldb2005.org/program/paper/thu/p601-sion.pdf>
- Slaney, M., & Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal Processing Magazine*, 25(2), 128–131.
- Such, J. M., & Criado, N. (2016). Resolving multi-party privacy conflicts in social media. *IEEE Transactions on Knowledge and Data Engineering*, 28(7), 1851–1863.
- Wang, H. J., Platt, J. C., Chen, Y., Zhang, R., & Wang, Y.-M. (2004). Automatic misconfiguration troubleshooting with peerpressure. In *Osd* (Vol. 4, pp. 245–257).
- Wang, Y.-M., Verbowski, C., Dunagan, J., Chen, Y., Wang, H. J., Yuan, C., & Zhang, Z. (2004). Strider: A black-box, state-based approach to change and configuration management and support. *Science of Computer Programming*, 53(2), 143–164.
- Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L. N., & Pasupathy, S. (2011). An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the twenty-third acm symposium on operating systems principles* (pp. 159–172). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/2043556.2043572> doi: 10.1145/2043556.2043572

- Young, N. E. (2008). Greedy set-cover algorithms (part 7 of encyclopedia of algorithms). *Springer Encyclopedia of Algorithms*. doi: 10.1007/978-0-387-30162-4_175
- Zhang, B., & Al-Shaer, E. (2011). On synthesizing distributed firewall configurations considering risk, usability and cost constraints. In *Proceedings of the 7th international conference on network and services management* (pp. 28–36). Laxenburg, Austria, Austria: International Federation for Information Processing. Retrieved from <http://dl.acm.org/citation.cfm?id=2147671.2147677>
- Zhang, J., Renganarayana, L., Zhang, X., Ge, N., Bala, V., Xu, T., & Zhou, Y. (2014). Encore: Exploiting system environment and correlation information for misconfiguration detection. *ACM SIGPLAN Notices*, 49(4), 687–700.
- Zhang, Z., & Mendelzon, A. (2005). Authorization views and conditional query containment. In *Icdt* (Vol. 3363, p. 259-273). Springer.
- Zhao, P. (2017). Similarity search in large-scale graph databases. In *Handbook of big data technologies* (pp. 507–529). Springer.