Journal of Computer Security 24 (2016) 379–403
DOI 10.3233/JCS-160548
IOS Press

Minimum cost rule enforcement for cooperative database access

Meixing Le [a], Krishna Kant [b,*], Malek Athamnah [b] and Sushil Jajodia [c]
[a] *Security Technology Business Unit, Cisco Corporation, San Jose, CA, USA*
E-mail: meile@cisco.com
[b] *Computer & Information Science Department, Temple University, Philadelphia, PA, USA*
E-mails: kkant@temple.edu, mathamna@temple.edu
[c] *Center for Secure Information Systems, George Mason University, Fairfax, VA, USA*
E-mail: jajodia@gmu.edu

Abstract. In this paper, we consider restricted data sharing between a set of parties that wish to provide some set of online services requiring such data sharing. Each party is 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. We examine 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. However, the paper is not focused on specific cost functions, but instead of efficient methods for enforcing rules in the face of access restrictions and inter-party data transfer needs. We propose an efficient heuristic algorithm for this minimal enforcement since the exact problem is NP-hard. In some cases, it is not possible to enforce the rules with the regular parties only. In such cases, we need help of trusted third parties (TPs). If all parties trust a single TP, such a party can enforce all unenforced rules, but it is desirable to use the TP minimally. We also consider the extended case where multiple TPs are required since not every regular party can trust a single TP.

Keywords: Cooperative data access, rule enforcement, consistent query planning, third parties

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

*Corresponding author. E-mail: kkant@temple.edu.

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 paper and will be addressed in future work. The purpose of this paper 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 paper 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 paper. 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 do not 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 will not 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 [13], 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 [14]. 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 [15], we studied the special case of a single TP trusted by all parties. In this paper, 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 [16] 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 paper we seek a single combined enforcement and query planning algorithm instead of the two different ones studied in [14] and [16]. 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 paper also provides more insights into the single TP case discussed in [15] and extend the analysis to multiple third parties.

The rest of the paper is organized as follows. Following the related work in Section 2, the problem is defined formally in Section 3. This section also illustrates why our problem is more complex than classical query planning and studies its complexity. Section 4 then describes a heuristic algorithm for combined enforcement and query planning and discusses its performance. Section 5 considers the issues in involving single and multiple third parties for rule enforcement and query planning. Section 6 then concludes the discussion.

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. [8] 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 [9] 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 [5,17] 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 [3,6,12], 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 [10] 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 [18]. 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 [19], and [21] 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 [11] 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 paper.

In addition, there are services such as Sovereign joins [2] to provide third party join services; we can think this as one possible TP model in our scenario. There is also some research [1,7,20] about how to secure the data for out-sourced database services.

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 1. We will illustrate various concepts using a simple running example, and thus we start with the example first.

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 1 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

Table 1
Authorization rules for the running example

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

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 1), and (b) rules that represent composition of more basic rules to satisfy the consistency property. For example, in Table 1, specifying basic rules (1) and (2) obligates us to add the derived rule (3) that gives P_W access to $E \bowtie_{\text{pid}} W$.

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 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]
(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 SDB_1 and SDB_2, 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 paper, 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.

3.2.1. 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 SDB,[2] 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

[1] It 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 paper.

[2] Since 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.

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 paper.

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 relational schemas $JR_t = [R_1, R_2, \ldots, R_n]$ with join predicates $JA_t = [(A_{l1}, A_{r1}), (A_{l2}, A_{r2}), \ldots, (A_{l,n-1}, A_{r,n-1})]$ respectively, where (A_{li}, A_{ri}) are the join attributes from the ith 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 paper; 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 3.2. This is unlikely to be the case in general, 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 paper; 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] According to this definition, $n = 1$ corresponds to the case of no join.

(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 paper; therefore, issues of cheating or holding back information are not considered here.

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 $E \bowtie_{\text{oid}} S$ 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 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*.

[4]The 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.

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

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 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 paper.

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 paper. 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 1. 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 $J_{\text{pl}} = E \bowtie_{\text{pid}} W$, and $A_{\text{pl}} = \{oid, pid, sid, location\}$.

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.

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 $R \bowtie S$.

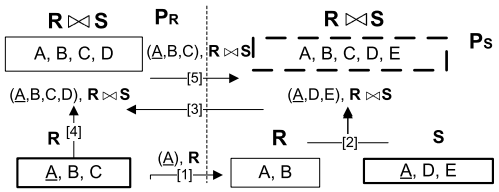

Fig. 1. Illustration of query planning.

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 [12] 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.

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 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.

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 [12] 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 $U = \{A_1, A_2, \ldots, A_n\}$, and a distinct attribute A_0 that we will use as key for join. We define a relation R with the schema $\{\underline{A_0}, A_1, A_2, \ldots, A_n\}$, S has the schema $\{\underline{A_0}, A_x\}$ where A_x is not in U. We also have $m > 1$ sets $\{S_1, S_2, \ldots, S_m\}$, 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, $i = 1, \ldots, n$, 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.

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

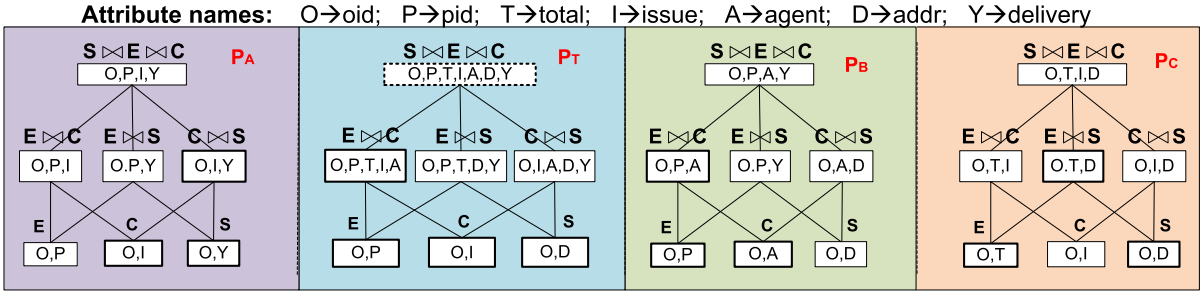

Fig. 2. Illustration of enforcement complexity.

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.

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.

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 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.

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 do not 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. Figure 3 shows part of the built graph for our running example. In particular, party P_W can compute $E \bowtie_{\text{pid}} W$ (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 $E \bowtie_{\text{pid}} W$ directly. On the other hand, P_S has access to both S and E (rules r_{13} and r_{14}), and so can get $E \bowtie_{\text{oid}} S$ (authorized by rule r_{15}) directly. S can further join $E \bowtie S$ and $E \bowtie W$ to obtain $E \bowtie_{\text{oid}} S \bowtie_{\text{pid}} W$ which is authorized by rule r_{17}.

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

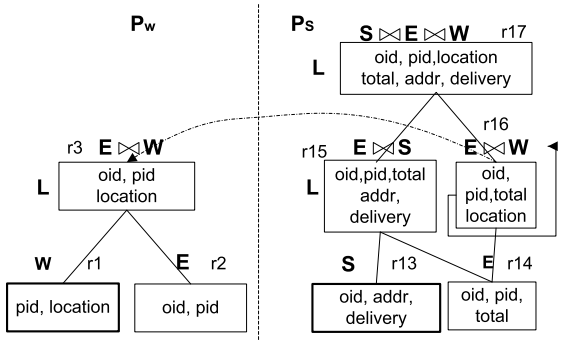

Fig. 3. Relevance graph for local enforcement.

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 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 \ 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 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.

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 ∈ 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 \ [Att(r_1) + Att(r_2)];
16:      if A_m(r) ≠ 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) \ Att(r_new);
27:         if A_m(r_t) ≠ 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) ≠ 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
```

The key quantity in the running time of the algorithm is N_{ql}, 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 $O(N_{\text{ql}} * N_{\text{qt}}^2)$.

Theorem 2. *A query plan generated by Algorithm 1 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 r_t and its local relevant rule is always consistent. A semi-join between a rule r_t 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 1 is consistent with the access rule set R. □

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 that are not considered by the algorithm are such that their missing attributes 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 the 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), or deleting/adding some rules. 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 $E \bowtie_{\text{oid}} S$
R3-b: P_W given {*oid*, *pid*, *sid*, *total*, *factory*, *location*, *addr*, *delivery*} over JP $S \bowtie_{\text{oid}} E \bowtie_{\text{pid}} W \bowtie_{\text{sid}} P$
R20-a: P_C given {*oid*, *location*, *pid*, *total*, *addr*} over JP $S \bowtie_{\text{oid}} E \bowtie_{\text{pid}} W$

Table 2 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 1. 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.

Table 2
Rule enforcement and planning results

Case	Rule	Changes in rules	Enforcement plan	Cost	Optimal 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

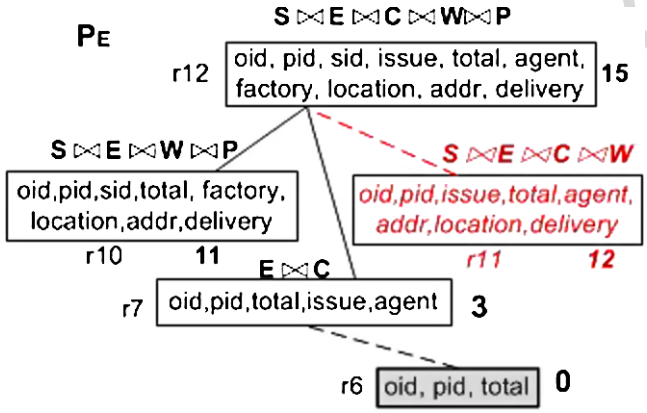

Fig. 4. New vs. old algorithm for local enforcement.

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 1, but for all 190 rule enforcements across these variants. In all cases, the algorithm runs in less than 0.1 seconds.

We note that our current (new) algorithm is more thorough than the algorithm in [16]. That algorithm only considers rules at level $n-1$ (instead of levels $n-1$ and $n-2$). Also, once it has already found a total enforcement (i.e., enforcement with all required attributes), it does not look further to reduce cost at level n. To illustrate the impact of these differences, consider the enforcement of rule r_{12} (Case 2 of Table 2). One difference is shown in Fig. 4. The old algorithm would choose r_{11} (shown in red italics) and join it with r_{10} since both of those lie at level $n-1$. Instead, the new algorithm joins r_{10} (at level $n-1$) with r_7 (at level $n-2$) and thereby gets the lower (and optimal) cost of 15. For the same scenario, the second difference is shown in Fig. 5. Here the focus is on how we enforce r_{10} itself. At first, both algorithms will enforce r_{10} by using r_8 and r_9 (shown in red italics) with a cost of 18. The old algorithm will stop at this point since all attributes of rule r_{10} are already covered. Instead, the new algorithm will still 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.

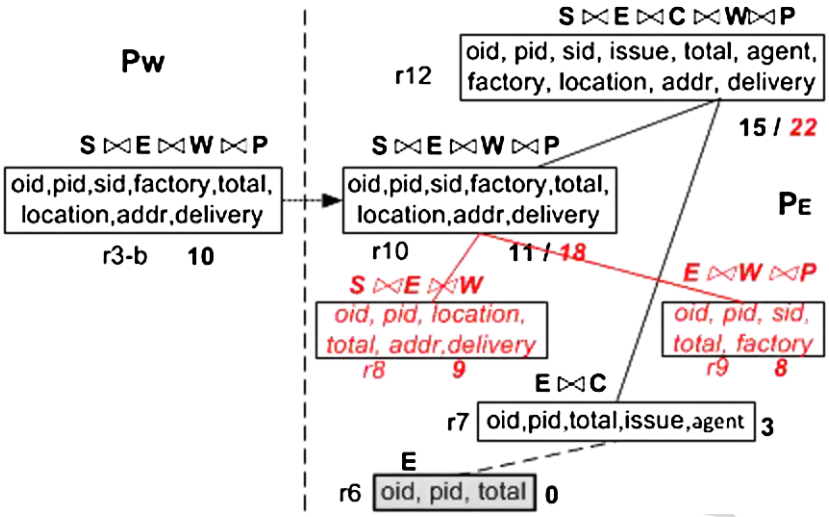

Fig. 5. New vs. old algorithm for remote enforcement.

5. Query planning with a third parties

It was discussed earlier that some rules may be unenforceable. As discussed in Section 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 [14] 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 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 paper.

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 $r_1, r_2, \ldots, r_k$ for some $k \leqslant K$ denote the rules sent to TP for a $k - 1$ way join. Let J_i denote the join path for rule r_i, $i = 1, \ldots, k$. (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 $\mathrm{Dup}(S_i)$, and this includes the key of S_i and possibly others. Therefore, $\pi(r_i)$ can be defined as follows:

$$\pi(S_i) = \left| S_i \cap \left(U \setminus \bigcup_{j=1}^{i-1} S_j \right) \right| + \left| \mathrm{Dup}(S_i) \cap \left(\bigcup_{j=1}^{i-1} S_j \right) \right|. \tag{1}$$

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

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 $\frac{w(S_i)*\pi(S_i)}{|S_j \cap (U \setminus \bigcup_{j=1}^{i-1} S_j)|}$, where $\pi(S_i)$ is defined in Eq. (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. Figure 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

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: $\text{Cost}(C) \leftarrow 0$
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 $\alpha = \frac{w(S_i)*\pi(S_i)}{|S_i \bigcap (U \setminus \bigcup_{j=1}^{i-1} S_j)|}$
9: $R \leftarrow R \setminus r_i$
10: $U \leftarrow U \setminus S_i$
11: $\text{cost}(C) += \pi(S_i) * w(S_i)$
12: $C \leftarrow C \bigcup r_i$
13: Return C and $\text{Cost}(C)$

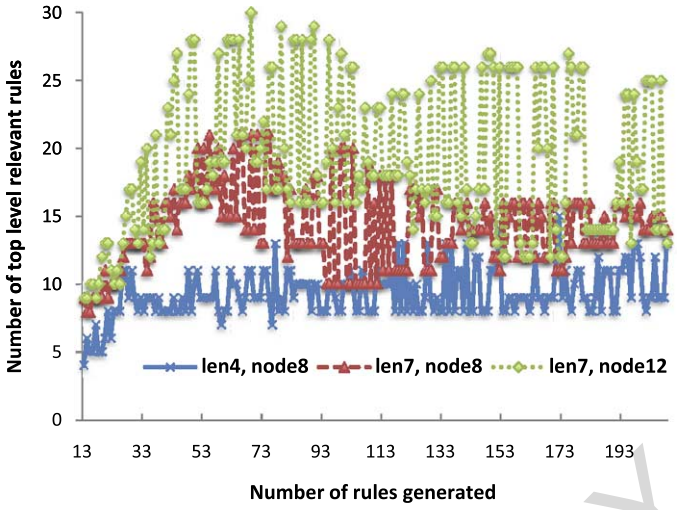

Fig. 6. Number of candidate rules.

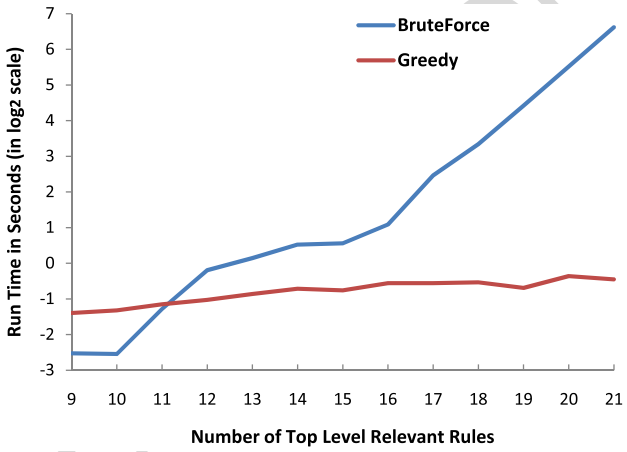

Fig. 7. Time comparison between two algorithms.

required. Figure 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. Figure 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. 8, the legends of "BruteForce4" and "Greedy4" indicate the target 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.

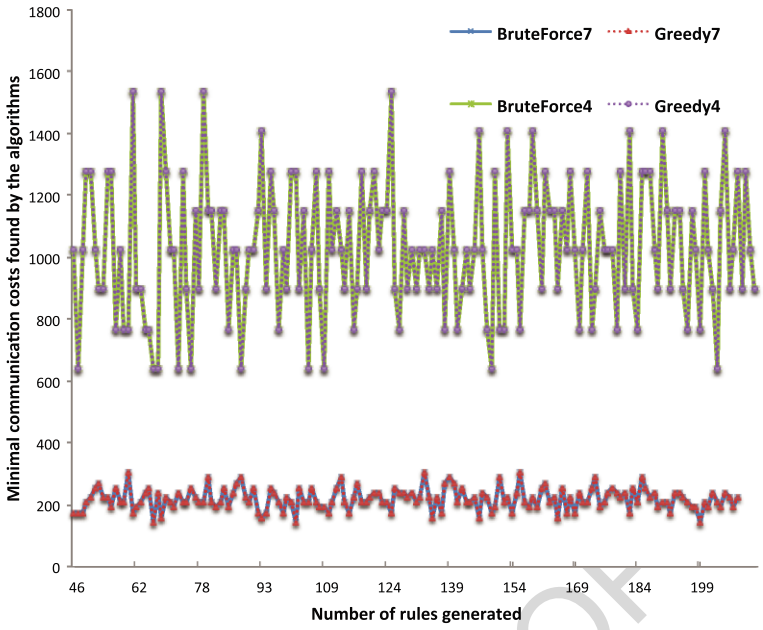

Fig. 8. Minimal communication costs found by two algorithms.

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, $i = 1, \ldots, K$. 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 4.2, we will use a combined algorithm for both.

In Section 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) 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 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 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 4.2. If the TP cost is zero, our algorithm can handle it as well.

Table 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 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

Table 3
Authorization rules for running example

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

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 [4]. All of these cases are easily handled by our algorithm by defining appropriate rules for TPs.

6. Conclusions and future work

In this paper 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 paper 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 paper 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 paper 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.

Acknowledgment

This research was supported by NSF award CNS-1527346.

References

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas and Y. Xu, Two can keep A secret: A distributed architecture for secure database services, in: *CIDR*, 2005, pp. 186–199.

[2] R. Agrawal, D. Asonov, M. Kantarcioglu and Y. Li, Sovereign joins, in: *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006*, Atlanta, GA, USA, 3–8 April 2006, IEEE Computer Society, 2006, p. 26.
[3] P.A. Bernstein, N. Goodman, E. Wong, C.L. Reeve and J.B. Rothnie Jr., Query processing in a system for distributed databases (SDD-1), *ACM Transactions on Database Systems* **6**(4) (1981), 602–625.
[4] D.F.C. Brewer and M.J. Nash, The Chinese wall security policy, in: *IEEE Symposium on Security and Privacy*, 1989, pp. 206–214.
[5] A. Calì and D. Martinenghi, Querying data under access limitations, in: *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008*, Cancún, México, 7–12 April 2008, IEEE, 2008, pp. 50–59.
[6] S. Chaudhuri, An overview of query optimization in relational systems, in: *Proceedings of the 7th ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems*, 1998, pp. 34–43.
[7] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati, Keep a few: Outsourcing data while maintaining confidentiality, in: *ESORICS*, Lecture Notes in Computer Science, Vol. 5789, 2009, pp. 440–455.
[8] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati, Controlled information sharing in collaborative distributed query processing, in: *ICDCS 2008*, Beijing, China, June 2008, pp. 2–7.
[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati, Authorization enforcement in distributed query evaluation, *Journal of Computer Security* **19**(4) (2011), 751–794.
[10] J. Goldstein and P. Larson, Optimizing queries using materialized views: A practical, scalable solution, in: *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001, pp. 331–342.
[11] A.Y. Halevy, Answering queries using views: A survey, *VLDB Journal* **10**(4) (2001), 270–294.
[12] D. Kossmann, The state of the art in distributed query processing, *ACM Computing Surveys* **32**(4) (2000), 422–469.
[13] M. Le, K. Kant and S. Jajodia, Access rule consistency in cooperative data access environment, in: *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012)*, October 2012, pp. 11–20.
[14] M. Le, K. Kant and S. Jajodia, Consistency and enforcement of access rules in cooperative data sharing environment, in: *Computers and Security*, November 2013, pp. 10–12.
[15] M. Le, K. Kant and S. Jajodia, Rule enforcement with third parties in secure cooperative data access, in: *27th Data and Applications Security and Privacy, DBSec 2013*, July 2013, pp. 3–6.
[16] M. Le, K. Kant and S. Jajodia, Consistent query plan generation in secure cooperative data access, in: *28th Data and Applications Security and Privacy, DBSec 2014*, July 2014, pp. 10–12.
[17] C. Li, Computing complete answers to queries in the presence of limited access patterns, *VLDB Journal* **12**(3) (2003), 211–227.
[18] R. Pottinger and A.Y. Halevy, MiniCon: A scalable algorithm for answering queries using views, *VLDB Journal* **10**(2,3) (2001), 182–198.
[19] S. Rizvi, A. Mendelzon, S. Sudarshan and P. Roy, Extending query rewriting techniques for fine-grained access control, in: *Proceedings of the ACM International Conference on Management of Data (SIGMOD'04)*, June 2004, pp. 3–6.
[20] R. Sion, Query execution assurance for outsourced databases, in: *VLDB*, ACM, 2005, pp. 601–612.
[21] Z. Zhang and A. Mendelzon, Authorization views and conditional query containment, in: *ICDT*, Vol. 3363, Springer, 2005, pp. 259–273.