Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues

Anastasios Kementsietsidis Marcelo Arenas Renée J. Miller
Department of Computer Science
University of Toronto
{tasos,marenas,miller}@cs.toronto.edu

ABSTRACT

We consider the problem of mapping data in peer-to-peer data-sharing systems. Such systems often rely on the use of mapping tables listing pairs of corresponding values to search for data residing in different peers. In this paper, we address semantic and algorithmic issues related to the use of mapping tables. We begin by arguing why mapping tables are appropriate for data mapping in a peer-to-peer environment. We discuss alternative semantics for these tables and we present a language that allows the user to specify mapping tables under different semantics. Then, we show that by treating mapping tables as constraints (called mapping constraints) on the exchange of information between peers it is possible to reason about them. We motivate why reasoning capabilities are needed to manage mapping tables and show the importance of inferring new mapping tables from existing ones. We study the complexity of this problem and we propose an efficient algorithm for its solution. Finally, we present an implementation along with experimental results that show that mapping tables may be managed efficiently in practice.

1. INTRODUCTION

Traditionally, data integration and exchange between heterogeneous data sources is provided mainly through the use of views, i.e., queries that map and restructure data between the heterogeneous schemas [13, 20]. Since queries depend on the underlying schemas, to correctly restructure and map data, the sources must be willing to share their schemas and cooperate in establishing and managing the queries. In our work, we consider peer-to-peer settings in which such close cooperation is either not desirable (perhaps for privacy reasons) or not feasible (perhaps due to resource limitations or the dynamic nature of the data structures) [11, 17].

SIGMOD 2003, June 9-12, 2003, San Diego, CA.

To find data when there is no agreement on the logical design of data (or on how different logical designs correspond), we must focus on data values and how values correspond. If we can map values, particularly identifying values (names or keys), we can still request and exchange specific data of interest. Indeed, this idea provides the basis for data exchange in current peer-to-peer systems. In file-sharing systems, like Gnutella [2] and Napster [7], querying is performed by using simple value searches on file names [19]. So queries are of the form: "Retrieve all files named X (or containing the phrase X)". This simple form of search has proven effective for so many applications, that there has been a flurry of research on making such searches more efficient and scalable [24; 26, and others]. This scheme works in domains where there is consensus on what the names should be. So for music files, where there is a standard, commonly accepted name for each song or album, data can be shared because each peer uses the same (or similar) values to name files.

However in other domains, where there is no accepted naming standard, different peers may necessarily have had to develop their own naming conventions. Standards often emerge after many sources have set up their own naming conventions. There may be many applications that depend on the use of the internal conventions. So, migration to conform to external standards is time-consuming and expensive [12]. To search data in such environments, people have made use of mapping tables that store the correspondence between values. At their simplest, these tables are binary tables containing pairs of corresponding identifiers from two different sources. With such tables, we can still use our simple value searches but for a peer to find a file called X it first consults a (shared or local) mapping table to find the name(s) of X in each acquainted peer. In general, we may need to map values containing multiple attributes. For example, geographic locations may be indicated by some form of federal postal code in one peer and by pairs of area codes and town names in a second. However, we can still use these mapping tables to exchange data related to specific values. The query "retrieve all information related to postal code X" in peer one becomes "retrieve all information related to the area code, town pair (Y, Z)" in peer two. Or the translation may be to a set of values if the map-

ping is not one-to-one or many-to-one. Note that data exchange in more structured data management settings (as opposed to file sharing settings) is often achieved using self-describing data models. Since there is no *a priori* agreement on the structure of the result, the data, once found, is exchanged with its descriptive schema. Nonetheless, simple value queries, where there is agreement on how values correspond, can be quite effective for facilitating search and exchange between data management systems.

Contributions: Mapping tables represent *expert knowledge* and are typically created by domain specialists. Indeed, currently the creation of mapping tables is a time-consuming and manual process performed by a set of expert curators. While widely used, especially in the biological domain [15], we are aware of no data management tools currently designed to facilitate the creation, maintenance and management of these tables. In this work, we discuss alternative semantics for these tables and we present a language that allows the specification of mapping tables under different semantics. We illustrate how automated tools can help managing mapping tables between multiple sources by inferring new mappings (that is, new entries in a mapping table). Specifically, we show that by treating mapping tables as constraints (called mapping constraints) on the exchange of information between peers it is possible to reason about them and check their consistency. Note that these constraints are not on the peers themselves (or their content), only on the way in which search values are translated. We study the complexity of the inference and consistency problems and we propose an algorithm for solving these problems. Finally, we present an implementation of our algorithm along with experimental results which show that, by using our results, mapping tables may be managed efficiently in practice.

The work presented in this paper is part of the Hyperion project [4]. The main goals of the project are: the definition of a peer-to-peer data management architecture; the study of viable data integration, exchange, and mapping mechanisms; the development of algorithms for the efficient search, retrieval and exchange of data among peers. Mapping tables provide the foundation for exchanging information between peers.

Outline of the paper: In Section 2, we illustrate our techniques with a specific example. Section 3 formally introduces the notion of mapping tables. Then, in Section 4, we present the benefits of considering mapping tables as constraints and we introduce a language for specifying how tables may be combined. Section 5 analyzes the complexity of both the inference and the consistency problems. Section 6 presents an efficient algorithm for solving these problems while Section 7 presents its implementation along with experimental results. Section 8 describes related work, while Section 9 offers a summary of the conclusions.

2. MOTIVATING EXAMPLE

Consider an example drawn from the domain of biological databases. Genomic data can be found in a large number of authoritative sources ranging from well-known public sources, to ones specific to individual research labs. The examples in this paper will be drawn from public sources including GDB [1] (a gene database), SwissProt [8] (a protein database), and MIM [6] (a database about genes and genetic disorders related with these genes). Integration of these sources to provide uniform access for scientists, although extremely desirable, seems unattainable due to a myriad of political, financial and technical reasons [15]. Among the technical reasons is the inherent heterogeneity of the sources which range from relational databases to formatted files or spreadsheets. In addition, the schemas and formats of the sources evolve rapidly in response to new biological techniques and requirements.

To achieve some degree of integration, biologists commonly use what we have called *mapping tables*. For example, mapping tables can be used to relate gene data in one source to the related protein data in another source (where the gene is said to *encode for* the protein). Note that the mapping table is not necessarily a function, there may be many proteins related to a gene. Even a mapping table relating gene identifiers may be many-to-many. This occurs often in biological sources where there may be aliases for the same identifier. As identifiers are updated, old identifiers may need to be kept. For example, they may refer to the content of static (non-updateable) sources such as journal articles which may contain antiquated names and identifiers for entities.

In what follows, we discuss some of the main characteristics and uses of mapping tables. First, we show that mapping tables can be used to associate values both within and across domains. Second, we show that mapping tables are an appropriate tool to use in peer-to-peer systems since they respect the autonomy of the peers. Finally, we motivate the need for alternative semantics in mapping tables and we present some examples that motivate why reasoning capabilities are desirable in an environment where mapping tables are used.

Associations within and Across Domains: Notice that by using mapping tables we are able to associate seemingly unconnected databases, something that has been called *mediation across multiple worlds* [21]. In a typical integration scenario, we are often dealing with *one world*, for example, a set of sources all containing information about genes. However, there are situations, where sources from *disjoint worlds* can be associated since the corresponding worlds are semantically close to each other. As an example, the GDB database has a mapping table in which it stores associations between its gene identifiers and protein identifiers from SwissProt. Using the mapping table, users of the GDB database are able to retrieve, for each gene, related protein(s) from SwissProt. An example of such a table is shown in Figure 1. While we have simplified (and shortened!) the data in the figure for exposition, the experiments we report in Section 7 use the actual GDB to SwissProt mapping table containing 8,780 entries.

Peer Autonomy: Autonomy is of utmost importance in any peer-to-peer system and in many types of

GDB_id	SwissProt_id
GDB:120231	P21359
GDB:120231	O00662
GDB:120231	Q9UMK3
GDB:120232	P35240
GDB:120233	P01138

Figure 1: Mapping Table 1

	Open-world	Closed-world
present X-value	Any Y-value	indicated Y-values
missing X-value	Any Y-value	No Y-value

Table 1: Alternative open/closed world semantics

networked applications. Mapping tables respect the autonomy of the sources that they associate. They are minimally invasive in that they permit searching across peers, but do not restrict the operation of peers in anyway beyond the agreement on values expressed in the tables. To see this, notice that the mapping table shown in Figure 1 does not express how genes and proteins are related in general, nor how they should be represented or stored in their respective sources. Rather, it only encodes the fact that a domain expert has determined that certain genes are related to certain proteins.

Semantics: Domain specialists have varying levels of expertise. Mapping tables should record not only the associations suggested by the domain specialists but also their confidence on these associations. In what follows, we discuss alternative semantics for mapping tables and show how these can be used to express partial or complete knowledge for a domain of interest.

A mapping table consists of two disjoint sets of attributes X and Y (we use a double line in figures to distinguish between the two). A tuple (x, y) in the mapping table is called a mapping and it indicates that the value x is associated with y. We say that an X-value appearing in a mapping table follows the *open-world* semantics if it can be associated with any Y-value. Under this semantics, the table encodes only partial information about X-values that appear in it. Alternatively, we say that an X-value appearing in the mapping table follows the *closed-world* semantics if it can only be associated with the indicated Y-values. Under this semantics, the table encodes complete information about the X-values that appear in it. Similarly, for an X-value not present in a mapping table, we say it follows the *open-world* semantics if it can be associated with any Y-value, while it follows the *closed-world* semantics if it cannot be associated with any Y-value. Tables 1 summarizes the above discussion.

Thus, we are led to a space of four alternative semantics for mapping tables. Specifically, under the *open-open-world* (OO-world) semantics both the X-values present in the table and those missing follow the corresponding open-world semantics. This semantics essentially allows the association of any X-value with any Y-value and is thus of no practical interest. Under the *open-closed-world* (OC-world) semantics the X-values present in the table follow the open-world semantics, while those missing follow the corresponding closed-world semantics. Given such a semantics, a mapping table essentially specifies the set of X-values that can be mapped to any Y-value, while the Y-values indicated on the table are not taken into consideration. As such, the semantics is of little practical use to a specialist aiming to record particular mappings between specific X and Y-values. Of more practical use, a mapping table under the *closed-open-world* (CO-world) semantics is capable of representing partial knowledge. This proves useful in situations where the domain specialist is an expert only on a subset of the domain. As such, she is able to record her expertise by constraining the Y-values with which specific X-values can be associated. However, since she is agnostic regarding the remaining X-values, she allows them to be associated with any Y-value. The *closed-closed-world* (CC-world) semantics is useful for representing complete knowledge for a domain. Under this semantics, a specialist specifies the complete set of correct mappings. Since the latter two semantics are of most interest, we focus our attention only on these two.

Automated discovery of mappings: Given a semantics for mapping tables, we would like to reason about them. To achieve this, we treat mapping tables as constraints on the exchange of information. The simplest rule for combining mapping tables is to take their conjunction, i.e., to look for all the associations that satisfy all constraints. Consider the mapping tables shown in Figure 2. Suppose domain specialists have specified a CO-world semantics for all three tables. The first table indicates that pairs of genes and proteins can *together* be associated with a genetic disorder. Table 2(b) associates genes with proteins while Table 2(c) associates genes directly with genetic disorders. Users may use Table 2(c) directly in a query to find all genetic disorders associated with a specific gene. However, a user may also wish to make use of the expertise of the domain specialists who created Tables 2(a) and 2(b) to discover if there are additional disorders that may be associated with this gene. Under a CO-world semantics, the mapping (GDB:120231, 193520) can be derived from these three tables since we can find a *witness* tuple that involves all the attributes of these tables and has GDB:120231 as GDB_id and 193520 as MIM_id. This tuple is t = (GDB:120231, O00662, 193520). Notice that t satisfies Table 2(c) since GDB:120231 is not mentioned there. On the other hand, the mapping (GDB:120231, 162200) is not valid with respect to these mapping tables, since there is no witness tuple for these values (no value of SwissProt_id satisfies the conditions mentioned above). Alternatively, the specialists could have used a CC-world semantics. If one or more of the mapping tables in Figure 2 have a CC-world semantics, the set of mappings between GDB and MIM changes. In this paper, we present solutions for inferring new mappings under both semantics.

GDB_id	SwissProt_id	MIM_id
GDB:120231	P21359	162200
GDB:120231	O00662	193520
GDB:120232	P35240	101000

Mapping table 2(a)

GDB_id	SwissProt_id
GDB:120231	O00662

Mapping table 2(b)

GDB_id	MIM_id
GDB:120233	162030

Mapping table 2(c)

Figure 2: An initial set of mapping tables

3. MAPPING TABLES

In what follows, we offer a formal definition of mapping tables. We use the relational model to present our ideas (since mapping tables fit conveniently into the relational model). However, our solutions do not require that any of the peers use this model. Indeed, our solutions are designed to work with peers that are information retrieval systems, DBMS, or file-sharing systems.

We use the letters A, B, C, D to denote individual attributes. For an attribute A, $dom(A)$ is the domain of A. The domains we consider are the typical domains found in most relational databases, such as, integers, strings, real numbers, booleans etc. The letters U, X, Y are used to denote sets of attributes. Letter R is used to denote a relation schema. We use the notation $R[U]$ to explicitly show the attributes of a relation schema. Letter r is used to denote a relation instance. The letter t is used to represent tuples. We use $t[X]$ to denote the values of tuple t in the attributes of X. Let $X = \{A_1, A_2, \ldots, A_k\}$ and let $dom(A_i)$ $(i \in [1, k])$ denote the domain of attribute A_i. Then, $dom(X) = dom(A_1) \times dom(A_2) \times \ldots \times dom(A_k)$. Finally, we use standard relation algebra operators such as projection (π_X) and selection $(\sigma_{X=x})$.

The values appearing in the mappings (and mapping tables) presented thus far are only constants. However, to represent the different semantics of mapping tables (CO-world or CC-world semantics) it is necessary to introduce variables. Specifically, let V be a set of variables where $V \cap dom(A) = \emptyset$, for every attribute A. We define a mapping to be a tuple which may contain constants or variables (such tuples are usually called *free tuples* in the literature [10]). More formally:

DEFINITION 1. *Given a set of attributes U, t is a mapping over U if for each $A \in U$, $t[A]$ is either a constant in $dom(A)$, a variable in V or an expression of the form $v - S$, where $v \in V$ and S is a finite subset of $dom(A)$.*

To describe a set of mappings, we use the term "mapping table" instead of the term "mapping relation". This is consistent with the literature where the term "table" has been used to denote relations containing variables [10]. Moreover, we impose the restriction that each variable appears in at most one mapping. This restriction is consistent with our intuition that two different mappings in a mapping table are completely independent. The following definition formalizes the above.

DEFINITION 2. *Let X and Y be nonempty disjoint sets of attributes. A mapping table m from X to Y is a finite set of mappings over $X \cup Y$ such that each variable appears in at most one mapping.*

Variables offer a compact and convenient way of representing common associations between values. An example of such an association is the identity.

EXAMPLE 3. *Consider a biological database at the University of Toronto and assume that it uses the same identifiers as the GDB database. We can represent this mapping table as a list of mappings of the form (id, id), where id is an identifier in the GDB database. Alternatively, we can construct a more succinct, data independent, mapping table containing the single mapping, (v, v), where v is a variable.*

Without variables, users must manually specify in their queries whether an identity mapping should be used. In mapping SwissProt data to GDB, the answer is likely no, while as the above example shows, other searches should make use of the identity. Variables also permit a simple representation for alternative mapping table semantics.

EXAMPLE 4. *Consider the mapping tables in Figure 3. The mapping table on the top of the figure uses the CO-world semantics. Thus, any gene, apart from the ones explicitly mentioned, can be associated with any protein. Now, consider the table on the bottom of the same figure which uses CC-world semantics. The first two mappings of this table are identical with the ones appearing in the top table. The last mapping states that all GDB_ids, except GDB:120231 and GDB:120232, may be mapped to any protein. Notice that the top table with CO-world semantics expresses the same associations as the bottom table with the CC-world semantics.*

We conclude this section by introducing the notion of valuation which will prove useful in the following paragraphs.

DEFINITION 5. *A valuation ρ over a mapping table m is a function that maps each constant value in m to itself and each variable v of m to a value in the intersection of the domains of the attributes where v appears. Furthermore, if v appears in an expression of the form $v - S$, then $\rho(v) \notin S$.*

4. MAPPINGS AS CONSTRAINTS

In this section, we view mapping tables as constraints (called *mapping constraints*) on the exchange of information between the sources. In doing so, we show that

GDB_id	SwissProt_id
GDB:120231	P21359
GDB:120232	P35240

GDB_id	SwissProt_id
GDB:120231	P21359
GDB:120232	P35240
v - {GDB:120231, GDB:120232}	v'

Figure 3: CO-world vs. CC-world semantics

we are able to reason about mapping constraints, that is, given a set of mapping constraints, we are able to infer new mapping constraints and determine if a set of constraints is inconsistent. In the following paragraphs we consider first how a single mapping table can be treated as a constraint. Then, we consider how sets of mapping constraints can be combined.

4.1 Mapping Constraints

Consider relations r and r' with schemas $R[U]$ and $R'[U']$, respectively, and also consider a mapping table m from X to Y, where $X \subseteq U$ and $Y \subseteq U'$. Let r'' be the Cartesian product of relations r and r' where every tuple t of r is related to every tuple t' of r'. Given a mapping table m from X to Y, we can use m as a condition to *filter* the above Cartesian product. Specifically, let t'' be a tuple of the Cartesian product. Tuple t'' must be removed from relation r'' if there is no valuation ρ over m such that $t''[X] \in \pi_X(\rho(m))$ and $t''[Y] \in \pi_Y(\sigma_{X=t''[X]}(\rho(m)))$. The intuition behind our definition is as follows. Consider the mapping table m from X to Y, a valuation ρ over m and a value $x \in \pi_X(\rho(m))$. Then, the value x is associated with a certain set of values in the domain of Y, namely, with the set $\pi_Y(\sigma_{X=x}(\rho(m)))$. As a result, a tuple $t \in r$ such that $t[X] = x$ can be mapped, with respect to mapping table m and valuation ρ, only to tuples $t' \in r'$ for which $t'[Y] \in \pi_Y(\sigma_{X=x}(\rho(m)))$. The above condition guarantees exactly this.

Example 6. *In Figure 4, the first two relations correspond to relations in the GDB and SwissProt databases, respectively. The third relation is a mapping table between GDB and SwissProt which uses the CC-world semantics. If we take the Cartesian product of the first two relations and use the mapping table as a condition to filter this product, we get the relation at the bottom of the same figure. For example, the first tuple in this relation is valid since there is a valuation ρ such that $\rho(v)$ = GDB:120231 and $\rho(v')$ = P21359.*

By treating a mapping table as a constraint, we are able to identify (in)valid mappings between objects that reside in different sources. That is, given a set of mapped objects we are in a position to tell whether they *satisfy* a mapping table m. We now formalize the above and introduce a new type of constraint, called a *mapping constraint*. Then, we introduce the notion of satisfiability for these new constraints and we discuss a number of issues that arise.

Let m be a mapping table from X to Y. We define $Y_m(x)$, where $x \in dom(X)$, as follows:

$$Y_m(x) = \{y \mid \exists t \in m \text{ and there exists valuation } \rho \text{ over } m \text{ such that } \rho(t[X]) = x \text{ and } \rho(t[Y]) = y\}.$$

Notice that $Y_m(x)$ contains all the values in $dom(Y)$ with which value $x \in dom(X)$ can be mapped, under the mapping table m.

Definition 7. *Let $U = X \cup Y$. An expression of the form $X \xrightarrow{m} Y$ is a mapping constraint over U. A U-tuple t satisfies $X \xrightarrow{m} Y$, denoted as $t \vDash X \xrightarrow{m} Y$, if $t[Y] \in Y_m(t[X])$. Furthermore, a relation r satisfies $X \xrightarrow{m} Y$ if every $t \in r$ satisfies $X \xrightarrow{m} Y$.*

As an example, in Figure 4, the relation at the bottom of the figure satisfies the constraint GDB_id $\xrightarrow{m}$ SwissProt_id, where m is the mapping table shown in 4(c). We use the letter μ to denote a mapping constraint, and Σ to denote a set of mapping constraints.

Notice that the previous definition assumes a CC-world semantics. We choose this semantics since we can translate (as shown in Example 4) any constraint μ under the CO-world semantics into a constraint μ' under the CC-world semantics such that r satisfies μ (under the CO-world) if and only if r satisfies μ' (under the CC-world). From now on, we assume that (by default) every mapping constraint is under the CC-world semantics. Every time that we mention a mapping constraint μ under the CO-world semantics, we assume that we are referring to the mapping constraint μ' mentioned above.

4.2 Mapping Constraint Formulas

In this section, we introduce a language which allows us to form expressions that combine mapping constraints by using conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$). Before we formally introduce the language, we motivate the use of such expressions.

Example 8. *Assume that mapping constraints μ_1 and μ_2 shown in Figure 5 were constructed by two different curators. How should these mapping constraints be combined? Clearly, this is a decision that only the user can make. For instance, if the user trusts both curators, then she will probably take the union of the mapping constraints (represented by $\mu_1 \vee \mu_2$). Alternatively, if the user only wishes to use mappings that have been validated by both curators, then she would use the intersection of mapping constraints (represented by $\mu_1 \wedge \mu_2$).*

In what follows we introduce a language for *mapping constraint formulas* (MCF) that can be used to express situations such as the ones mentioned in the example. The grammar of the language is defined as follows:

$$\text{MCF} := (\text{MCF} \wedge \text{MCF}) \mid (\text{MCF} \vee \text{MCF}) \mid \neg\text{MCF} \mid \mu$$

where μ is the only terminal symbol representing a mapping constraint defined over some mapping table m. We have already defined what it means for a tuple to satisfy a mapping constraint μ. We now offer a definition of what it means for a tuple to satisfy a mapping constraint formula defined over a set of mapping constraints.

GDB_id	Gene Name
GDB:120231	NF1
GDB:120232	NF2
GDB:120233	NGFB

4(a) Relation in GDB

SwissProt_id	Protein Name
P21359	NF1
P35240	MERL

4(b) Relation in SwissProt

GDB_id	SwissProt_id
GDB:120232	P35240
$v - \{$ GDB:120232 $\}$	$v' - \{$ P35240 $\}$

4(c) Mapping table from GDB to SwissProt

GDB_id	Gene Name	SwissProt_id	Protein Name
GDB:120231	NF1	P21359	NF1
GDB:120232	NF2	P35240	MERL
GDB:120233	NGFB	P21359	NF1

Tuples that can be mapped from GDB to SwissProt

Figure 4: Using mapping tables as constraints

GDB_id	SwissProt_id
GDB:120231	P21359
GDB:120231	Q9UMK3

(a) Mapping constraint μ_1

GDB_id	SwissProt_id
GDB:120231	Q14930
GDB:120231	Q9UMK3

(b) Mapping constraint μ_2

Figure 5: Constraints from GDB to SwissProt

DEFINITION 9. *Consider mapping constraint formula ϕ over a set of attributes U and a U-tuple t.*

- *If $\phi = \mu$, then $t \vDash \phi$ iff $t \vDash \mu$.*
- *If $\phi = \neg\phi_1$, then $t \vDash \phi$ iff it is not true that $t \vDash \phi_1$.*
- *If $\phi = \phi_1 \wedge \phi_2$, then $t \vDash \phi$ iff $t \vDash \phi_1$ and $t \vDash \phi_2$.*
- *If $\phi = \phi_1 \vee \phi_2$, then $t \vDash \phi$ iff $t \vDash \phi_1$ or $t \vDash \phi_2$.*

Apart from combining mapping constraints, mapping constraint formulas augment our expressiveness, as the following example illustrates.

EXAMPLE 10. *The identity between pairs of attributes A, B and C, D can be specified by means of the mapping constraint $\mu : AB \xrightarrow{m} CD$ with mapping table m containing only the mapping (u, v, u, v), where u and v are variables. Assume that a user wants to specify a mapping constraint where the values in A, B are identical to the values in C, D except for the set of tuples $\{(a_i, b_i) \mid i \in [1, n]\} \subseteq dom(A) \times dom(B)$. How should mapping constraint μ be modified to express this?*

By definition, through mapping constraints, we are only able to exclude certain values, of some column, from participating in any mapping. Mapping constraint formulas allow us to do the same thing for whole tuples. Going back to our example, for every $i \in [1, n]$, let $\mu_i : AB \xrightarrow{m_i} CD$ be a mapping constraint with mapping table m_i containing only the mapping (a_i, b_i, a_i, b_i). Then, the following mapping constraint formula expresses the desired constraint: $\mu \wedge \neg\mu_1 \wedge \cdots \wedge \neg\mu_n$.

A final note, regarding the introduction of negation. In the next section, we show that it allows us to have a uniform approach to solve the two problems considered in this paper: consistency and inference.

5. CONSISTENCY AND INFERENCE

Given that considerable effort is put into creating mapping tables we have found that curators and users alike often need the following capabilities.

Infer new mapping tables: A common task is to find the set of all mapping tables that are valid over a specific set of attributes U. To do this, we must combine the knowledge from the mapping tables available in a network of peers. As we show experimentally in Section 7, a set of mapping tables, viewed as constraints, can often be used to infer additional mappings – mappings that are not explicitly represented in any peer.

Determine consistency of mapping constraints: Curators edit, copy, or merge mapping tables that come from a variety of sources and it can be a cumbersome task to ensure that the mapping constraints of one table do not invalidate those expressed by another. As an example, note that the conjunction of mapping constraints shown in Figure 2 is inconsistent under the CC-world semantics. Even for this small example, inconsistency is not apparent and close inspection of the mappings is necessary to discover this. We want to provide automated techniques to help curators determine whether a set of mapping constraints is consistent.

Inferred mappings and knowledge of inconsistencies may be directly useful to a curator. In addition, both of these mechanisms play an important role in helping a curator understand and correctly specify the semantics of a set of mapping constraints. We do not expect a curator to write down a set of complex constraint formulas unaided. Rather, we expect that automated inference and consistency checks will help a curator understand whether a default semantics (perhaps the disjunction of a set of constraints under the CC-world semantics) is appropriate for a specific set of domains.

5.1 Problem Definition

Given a mapping constraint formula (MCF) ϕ over a set of attributes U, we say that ϕ is *consistent* if there exists a nonempty relation r of U that satisfies ϕ. Furthermore, given a set of MCFs $\Sigma \cup \{\phi\}$ over U, we say that Σ *implies* ϕ, denoted by $\Sigma \models \phi$, if for every relation r of U, if $r \models \Sigma$ then $r \models \phi$. The *consistency problem* is the problem of determining whether a given MCF is consistent, and the *inference problem* is the problem of verifying whether a set of MCFs implies another MCF.

The consistency and inference problems for MCFs are equivalent. To check whether a mapping constraint formula ϕ is consistent we verify whether it is not true that ϕ implies a mapping table with no mappings. To check whether a set of mapping constraint formulas Σ implies ϕ we verify whether $\neg\phi \wedge \bigwedge_{\varphi \in \Sigma} \varphi$ is not consistent. Thus, to analyze the complexity of these problems, we can focus in the consistency problem.

5.2 Complexity of Consistency Problem

The following result shows that the consistency problem for mapping constraint formulas cannot be solved efficiently.

THEOREM 11. *The consistency problem for mapping constraint formulas is NP-complete.*

The size of the input of this problem is the size of the formula to be checked for consistency. This size depends on the number of mapping constraints in the formula and the number of attributes and mappings in each mapping constraint. Thus, the consistency problem is NP-complete in the size of these three parameters.

A natural question at this point is what kind of restrictions can be imposed on the consistency problem to reduce its complexity. A natural restriction is to consider only conjunctions of mapping constraints. Given a set of mapping constraints $\Sigma = \{\mu_1, \ldots, \mu_n\}$, we say that Σ is consistent if $\mu_1 \wedge \cdots \wedge \mu_n$ is consistent. The following theorem shows that the complexity does not change for conjunctions of mapping constraints.

THEOREM 12. *The consistency problem for conjunctions of mapping constraints is NP-complete.*

To deal with this high complexity we provide a solution that is based on the notion of path. A path θ is a list $P_1, P_2, \ldots, P_n$ of peers such that peer P_i stores mapping tables between its data items and the data items of peer P_{i+1}, where i ranges from 1 to $n-1$. The idea behind our proposal is to check the consistency for conjunctions of mapping constraints associated to paths. The details of this algorithm are presented in the next section. In the remainder of this section, we describe what are the parameters that determine the complexity of the consistency problem over paths, and we present results that show how these parameters influence its complexity. We use these results to understand under what assumptions the problem can be solved efficiently.

A set of mapping constraints Σ over a set of attributes U forms a path if there exists a collection $U_1, \ldots, U_n$ of nonempty pairwise disjoint subsets of U such that for every $X \xrightarrow{m} Y \in \Sigma$, there exists $i \in [1, n-1]$ such that $X \subseteq U_i$ and $Y \subseteq U_{i+1}$. The complexity of the consistency problem for conjunctions of mapping constraints forming paths depends on the number of mapping constraints in each peer, the number of mappings in each mapping constraint, the length of the paths and the arity of the mapping constraints. Mapping constraints can contain thousands of mappings and, therefore, it does not seem reasonable to impose restrictions on this number. Thus, we investigate only the assumptions that we have to impose on the other parameters in order to obtain an efficient algorithm to solve the consistency problem.

The following theorem shows two conditions that must be taken into account in order to construct an efficient algorithm for the consistency problem. First, if the number of mapping constraints per peer is not fixed, then the consistency problem cannot be solved efficiently. Second, if the length of the path and the arity of the mapping constraints are not fixed, then the consistency problem cannot be solved efficiently.

THEOREM 13. *For each of the following conditions the consistency problem for conjunctions of mapping constraints forming paths is NP-complete.*

- *The length of the paths and the arity of the mapping constraints are fixed.*
- *The number of mapping constraints per peer is fixed.*

Given the above, we make two assumptions to solve the consistency problem. First, we assume that the number of mapping constraints per peer is small. Second, we assume that the length of the paths is also small. Paths of fixed length arise often in practice. For example, it is known that in Gnutella the paths of interest have a maximum size of 7. Under these assumptions, in the next section we present an efficient algorithm for checking consistency and doing inference of mapping constraints forming paths.

6. THE ALGORITHM

Consider a path $\theta = P_1, P_2, ..., P_n$ of peers, and let U_i be the set of attributes in peer P_i, $1 \leq i \leq n$. Let Σ denote the set of mapping constraints over path θ. Two more notions are necessary for our purposes. The first notion is that of an extension of a mapping constraint. Specifically, given a mapping constraint $\mu : X \xrightarrow{m} Y$, we define the extension of μ, denoted as $ext(\mu)$, to be:

$$ext(\mu) = \{\rho(t) \mid t \in m \text{ and } \rho \text{ is a valuation over } m\}.$$

Furthermore, we say that μ is a *cover* of a set of mapping constraints Σ over U if

1. Σ is consistent if and only if there exists $t \in ext(\mu)$
2. For every mapping constraint $\mu' : X \xrightarrow{m'} Y$, $\Sigma \models \mu'$ if and only if $ext(\mu) \subseteq ext(\mu')$.

The algorithm presented in the following paragraphs accepts as input a path θ, a set Σ of mapping constraints over path θ, and two sets of attributes $X \subseteq U_1$, $Y \subseteq U_n$

in peers P_1 and P_n, respectively. Then, it computes a mapping constraint $\mu : X \xrightarrow{m} Y$ that is a cover of the set Σ of constraints. As such, the algorithm can be used to solve both the inference and the consistency problems.

For the inference problem, given Σ and a mapping constraint $\mu' : X \xrightarrow{m'} Y$ we want to check whether $\Sigma \models \mu'$. To solve this problem, it is sufficient to run the proposed algorithm and check whether $ext(\mu) \subseteq ext(\mu')$. The check, due to Condition 2 above, provides an answer to the inference problem.

For the consistency problem, we run our algorithm as before with the exception of sets X and Y which, in this case, are all the attributes in peers P_1 and P_n respectively, i.e., $X = U_1$ and $Y = U_n$. At the end of the algorithm, we can check whether Condition 1 above is satisfied. If this is the case, set Σ is consistent.

6.1 Design Decisions

Most algorithms for checking consistency or doing inferencing of constraints are centralized in that they assume that all constraints are locally available. However, in a peer-to-peer environment each peer stores locally only the constraints that involve itself and its immediate acquaintances. Still, for small networks with a small number of mappings per constraint it may be reasonable to send all the constraints to a single peer and perform all the necessary computation there. In a more realistic scenario, though, there will be tens or even hundreds of peers. Although we do not expect each peer to have a large number of mapping constraints, we do expect that each constraint may have a large number of mappings. In some situations, the size of constraints may be proportional to the size of stored relations. In the GDB peer, for example, there are approximately 6.5 million objects stored in the peer while there are 3.5 million links to external sources. This latter number is the cumulative number of mappings in the mapping constraints of the GDB peer. Given the above, we propose an algorithm that takes advantage of the distributed nature of the peer-to-peer architecture and distributes its computation among the peers in a given path.

The algorithm runs on top of a prototype peer-to-peer data management system in which each peer manages a collection of data. Each peer autonomously chooses a logical design and physical organization for the data. Peers communicate using our own implementation of a Gnutella-like protocol, customized to our specific needs. The main algorithm was developed with two main goals in mind. First, it must distribute the computation and, thus, take advantage of the computational resources of each peer. Second, it should deliver results in a streaming fashion. Streaming has proven valuable in many deployed peer-to-peer systems where results are delivered as soon as they become *available*.

To present our algorithm, we use a simple running example that involves a path $\theta = P_1, P_2, P_3, P_4$ of four peers. The mapping constraints Σ in this path are shown in Figure 6. Peer P_1 has attributes A_i ($i \in [1, 6]$), P_2 has attributes B_i ($i \in [1, 6]$), P_3 has attributes C_i ($i \in [1, 4]$) and P_4 has attributes D_i ($i \in [3, 4]$). Our aim is to compute the cover μ between the attributes of

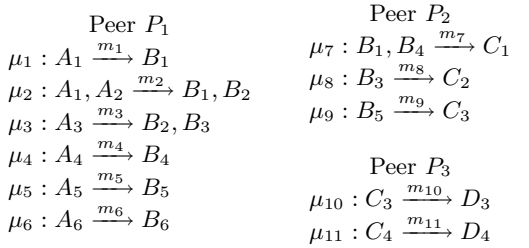

Figure 6: A path $\theta = P_1, P_2, P_3, P_4$ of 4 peers.

peer P_1 and those of peer P_4.

Consider the following algorithm for computing the cover. First, peer P_1 sends to P_2 all the mapping constraints between these two peers. Peer P_2 uses these constraints, along with its own constraints, to create a cover between peers P_1 and P_3. Then, peer P_2 forwards the cover to peer P_3. Peer P_3 repeats the process and creates a cover between peers P_1 and P_4. Peer P_3 sends the computed cover back to peer P_1 and the computation concludes.

The above algorithm, although it does distribute computation, suffers from two shortcomings. First, it performs unnecessary computation. Notice in Figure 6 that peer P_1 need not send to peer P_2 the actual mappings corresponding to constraint μ_6. The reason for this is that the part of the computation of the cover that involves attribute A_6 can be done locally in peer P_1. Furthermore, this computation can be done independently of the computation that involves the remaining constraints. The second shortcoming of the algorithm is that it does not work in a streaming fashion since peer P_1 has to wait for the whole computation to finish in order to retrieve the cover between itself and peer P_4. In the following paragraphs, we present an algorithm which addresses these issues.

6.2 Partitions

A key concept in our algorithm is that of *partitions*. Consider peers P and P' and let $\Sigma_{P,P'}$ be the set of constraints between these two peers. A partition Π is a subset of $\Sigma_{P,P'}$ and it is constructed as follows. Consider a graph $G_{P,P'} = (V, E)$, where V contains one vertex for each constraint in $\Sigma_{P,P'}$ and there is an edge between two constraints if their attributes overlap. Partition Π contains all the constraints of $\Sigma_{P,P'}$ whose corresponding vertices belong to the same connected component of $G_{P,P'}$. Applying the above procedure in the constraints between peer P_1 and peer P_2, shown in Figure 6, we get four partitions. The first partition Π_1 consists of the first three constraints, while each of the remaining three constraints constitutes a partition by itself. Figure 7 shows the partitions for the first three peers.

The benefit gained from partitioning the constraints in each peer is two-fold. First, while computing the cover, we are able to consider the constraints of each partition in isolation. This reduces the computational cost, with the exception where there is only one partition in $\Sigma_{P,P'}$, since we consider fewer constraints at

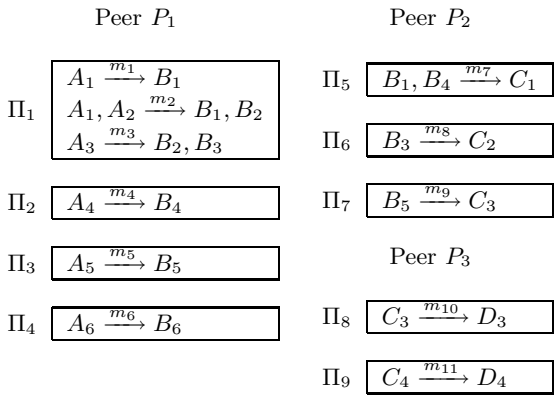

Figure 7: Peer P_1, P_2 and P_3 partitions.

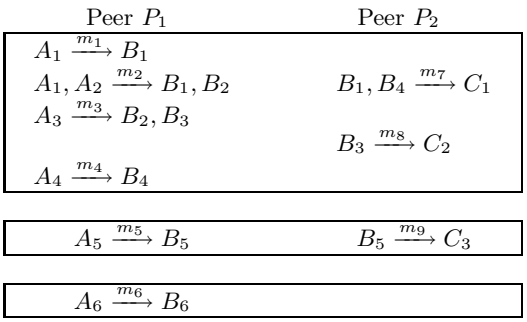

Figure 8: Inferred partitions over P_1 and P_2.

a time. Second, we can work on different partitions in parallel. This reduces the computation time and it also facilitates the delivery of results in a streaming fashion.

6.3 Description of the Algorithm

The algorithm has two phases, namely, the *information gathering phase* and the *computation phase*. The objective of the former phase is to collect enough information from the peers so as to reduce the computation in the latter phase. At the same time, the information gathered is used to reduce network traffic and to help determine how much of the computation can be executed in parallel. In what follows, we examine each of the two phases in more detail. To explain the algorithm, we use the example of Figure 6.

6.3.1 The Information Gathering Phase

The phase begins in peer P_1 with the computation of partitions. Then, peer P_1 sends to peer P_2, for each of his partitions, the set of attributes in the partition. No mappings are sent at this phase. Peer P_2 computes its own partitions and, using the information for the partitions of peer P_1, it computes a new set of *inferred* partitions. The inferred partitions possibly involve constraints of both peers and they are computed as follows. We construct a partition graph $G_\Pi = (V_\Pi, E_\Pi)$, where V_Π contains one vertex for each partition in the union of partitions of P_1 and P_2. There is an edge between two partitions if their attributes overlap. Each connected component of graph G_Π is an inferred partition. Figure 8 shows the inferred partitions for the first two peers.

We use inferred partitions to discover interdependencies, or lack thereof, between partitions. Then, in the computation phase, we perform parallel computation and streaming of results along different inferred partitions. As an example, if the information gathering phase terminates here, then computation of the cover between peers P_1 and P_3 can be performed independently and in parallel along the three inferred partitions.

The information gathering phase continues with peer P_2 sending to peer P_3 the sets of attributes corresponding to its inferred partitions. Peer P_2 sends only the inferred partitions involving some its own constraints, i.e., only the top two in Figure 8. Peer P_3 computes its own partitions, and using the information regarding the propagated inferred partitions from peer P_2, it computes a new set of inferred partitions. This concludes the information gathering phase.

6.3.2 The Computation Phase

For each of the inferred partitions in the previous phase, the set of constraints in the partition belongs to a set of peers that form a sub-path θ' of path θ. Let $\Sigma_{\theta'}$ denote the set of constraints in an inferred partition over path θ'. During this phase, we consider each inferred partition and we compute its cover. Specifically, given the path θ' and the set $\Sigma_{\theta'}$ of constraints, we compute a cover of $\Sigma_{\theta'}$ over the attributes of the peers that are the endpoints of θ'.

In more detail, assume an inferred partition over the whole path, i.e., $\theta' = \theta$. The computation of the cover starts, in general, at the last peer of the path. In the special case where $\theta' = \theta$, the computation starts at the penultimate peer. Thus, in our example we start at peer P_3. Peer P_3, using the local constraints of the current inferred partition, executes a local algorithm that computes a cover between peers P_3 and P_4. The cover only involves the attributes of the two peers that appear in the inferred partition. The mappings belonging to the cover are streamed to peer P_2. Using the information from the inferred partitions, P_2 determines with which of its own partitions the incoming stream of mappings should be associated. Then, it uses this information to generate a cover between itself and peer P_4. The mappings from this cover are, in turn, streamed to peer P_1. In the final step, peer P_1 uses the incoming stream of mappings to generate a cover between its own attributes and those of peer P_4.

Notice that there can be more than one inferred partitions over the whole path θ. Each such partition produces a cover over non-overlapping subsets of attributes of peers P_1 and P_4. To compute the cover μ between all the attributes of the two peers, we first take the Cartesian product μ' of the computed covers. To finish the computation of μ we need to take into account the inferred partitions that involve attributes of peer P_1, but are not over the whole path θ (e.g. the partition in Figure 8 involving attribute A_6). Specifically, in the fi-

$GDB \xrightarrow{m_1} MIM$
$GDB \xrightarrow{m_2} SwissProt$
$Hugo \xrightarrow{m_3} GDB$
$Hugo \xrightarrow{m_4} Locus$
$Hugo \xrightarrow{m_5} SwissProt$
$Hugo \xrightarrow{m_6} MIM$
$Locus \xrightarrow{m_7} GDB$
$Locus \xrightarrow{m_8} Unigene$
$Locus \xrightarrow{m_9} MIM$
$Unigene \xrightarrow{m_{10}} SwissProt$
$SwissProt \xrightarrow{m_{11}} MIM$

Figure 9: Biological mapping tables

Path	Length	Computed Mappings	New Mappings	Time (in secs)
1	5	6163	927	16.00
2	4	6193	11	15.00
3	3	9334	543	22.00
4	3	8704	10	22.00
5	3	6525	64	10.00
6	5	3276	397	26.00
7	4	8813	24	23.00

Figure 10: Inferred mappings

nal step of the computation of μ we take the Cartesian product of μ' with the values of attribute A_6.

7. EXPERIMENTAL RESULTS

To evaluate our algorithm, we undertook two studies. The first was designed to understand whether (and to what extent) our solutions provide added value for communities that already use and exchange mapping tables extensively. For this study, we used real mapping tables from several publicly available biological databases. The second study was designed to evaluate whether the characteristics of our algorithm are appropriate and effective in a peer-to-peer environment. We are not aware of any other work designed to manage mapping tables, so we were unable to do a comparison study. However, we do present some of the performance characteristics of our algorithms on both the biological data and on a B2B example.

Our implementation uses geographically distributed machines with one peer per machine. Each peer consists of two modules. The first module interacts with the peers storage manager to retrieve mappings (from disk) and performs the computation of the cover. Also, the module is responsible for the creation of inferred partitions and the validation of incoming mapping streams. The second module implements the peer-to-peer networking protocol. In terms of memory requirements, we allow each peer to decide how much *cache* to use. Peers that use a small cache are able to store only a limited number of mappings during the computation of covers. Such peers generally produce more network traffic since their available cache may fill up quickly and thus they have to stream mappings more often. On the other hand, peers with a larger cache generate less traffic and do more computation between two consecutive network transmissions.

Biology Domain For this study we used actual mapping tables retrieved from six biological databases, GDB, MIM, and SwissProt (which have been discussed in our examples), along with Hugo, Locus, and Unigene [3, 5, 9]. The table sizes range from seven thousand to twenty-eight thousand mappings with an average of thirteen thousand mappings. These mapping tables are relatively simple tables (they are all binary and comprise a single partition). However, this setting is a very common one where mapping tables represent links between identifiers used in different data sources.

Throughout the paper, we have stressed the importance of inferring new mappings especially when two peers first become acquainted. This experiment shows the benefit of inferring mappings even between peers that are *already* acquainted. Figure 9 lists the mapping tables that we found between the 6 biological databases. Consider two such peers, the Hugo database and the MIM database. There exists a mapping table with eight thousand mappings between the identifiers of these two databases. We assumed two sources to be acquainted if one contained a mapping table with attributes from the other. Under this assumption, our peer-to-peer network contained seven different paths, of different sizes, between Hugo and MIM. For example, one such path is θ = *Hugo, GDB, SwissProt, MIM*. In our experiments, we visited these paths in turn, in the order shown in the first column of Figure 10, and computed the cover between the endpoint peers for each of these paths. In the same figure, we show both the number of mappings we computed between Hugo and MIM in each path, and the number of new mappings that we computed which are not in the initial Hugo to MIM mapping table (and are not computed by the previously visited paths). Notice that the length of the path is not correlated with the number of computed mappings. Rather, the number of computed mappings is more a reflection of the knowledge embedded in the different tables used. Overall, after visiting all seven paths we compute approximately two thousand new mappings which is a 25% increase with respect to our initial set of mappings. The whole computation along each path takes on average 19 seconds. Since we stream results along each path, the reported computation time here, and in the following paragraphs, is the time it takes to receive the last computed mapping. The first mapping always arrives with no perceptible delay.

To understand how well our algorithm performs, we consider the scalability of the algorithm for different path sizes and different mapping table sizes. To understand the effect of path size in isolation, we use three paths of different lengths that all produced about the same number of computed mappings. Figure 11 shows that the running time of the algorithm scales gracefully for each of the path lengths, as we change the average number of mappings in each of the mapping tables along the path. In all the above experiments, we keep constant the size of the cache in each peer. Specifically, we allow each peer to store, on average, 64 mappings. Our experiments with different cache sizes indicate that

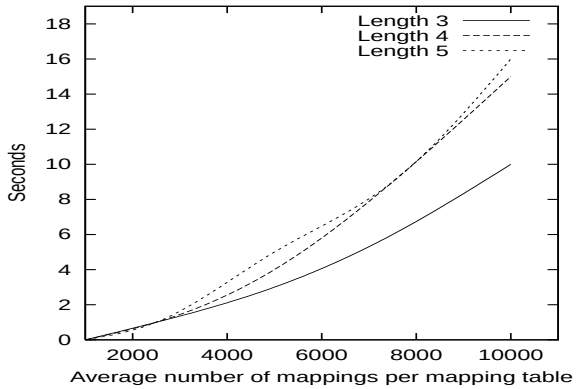

Figure 11: Scalability in path and table size

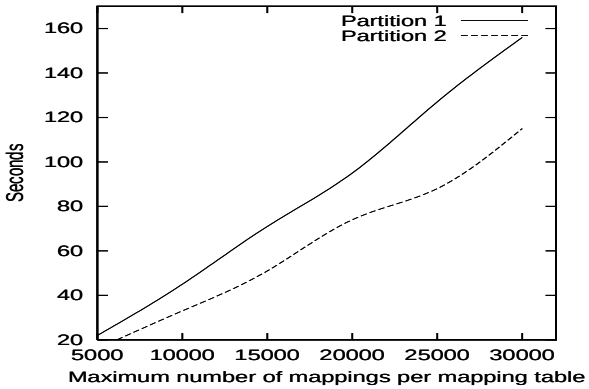

Figure 12: Per partition execution time

for small paths, increasing the cache size has no significant effect on the running time. For larger paths, increasing the cache has initially a noticeable positive effect on the running time. However, as the cache size continues to increase, the running time starts to increase also and there is a noticeable delay in the arrival of the first streamed mapping(s). The reason for this is that, as we increase the cache size, we allow each peer to do more computation, on bigger data sizes, but this results in less streaming. This, in turn, creates a delay in the arrival of the first streamed mappings. In the extreme case where the cache size is big enough to fit all the mappings that a peer can process during a computation, no streaming occurs and each peer forwards its mappings to the next peer only after it finishes its part of the computation. In the meantime, all other peers are idle. Clearly, this approach is not efficient. In conclusion, our experiments show that cache sizes from 64 to 128 mappings result the best running times for these data characteristics.

B2B Domain The second domain is a business-to-business (B2B) setting. We used this study to evaluate the effectiveness of our algorithms on more complicated (non-binary) mappings involving combinations of attributes. We considered a setting in which multiple businesses, or organizations within a business, are exchanging client data. Each organization may store data in different formats and the data itself may be dirty or inconsistent (within and across peers). Data cleansing, particularly duplicate elimination [25], can be used to detect data values that correspond. The results of these techniques can be represented as mapping tables between sets of domains. Additionally, we may have mapping tables representing a fixed set of known relationships (perhaps $Age \rightarrow AgeGroup$) across mismatched domains [16]. We consider a simple scenario with three organizations whose mapping constraints are shown in Figure 13. Note that peer P_1 has two partitions while peer P_2 has three. Also, several of the mapping tables involve variables including the first constraint m_1 which includes the mapping $(v_1, v_2) \rightarrow (v_1, v_2)$ indicating the identity mapping, together with mappings containing common misspellings and nicknames.

For our experiments, we used synthetically generated mappings of constants along with some manually created variable mappings. Figure 12 shows the running times of the algorithm along each of the two partitions of this example as the number of mappings is increased. This experiment shows that despite the presence of variables and the use of a richer semantics for tables, we are still able to compute new mappings efficiently. As with our biological example, the streaming provided by our algorithm delivers the first mappings almost instantaneously and the total execution time scales approximately linearly with the number of computed mappings.

Peer P_1	Peer P_2
$FName, LName \xrightarrow{m_1} FN, LN$	$FN \xrightarrow{m_5} Gender$
$AreaCode, Street \xrightarrow{m_2} Zip$	$Zip, City \xrightarrow{m_6} State$
$Street \xrightarrow{m_3} Zip$	$Age \xrightarrow{m_7} AgeGroup$
$AreaCode \xrightarrow{m_4} City$	

Figure 13: Mapping tables for a B2B domain.

8. RELATED WORK

There has been a great deal of recent work on data management in peer-to-peer systems on issues ranging from data placement [17], schema mediation [18], reliable data archiving [14], to data modeling [11]. Bernstein et al. [11] introduce the Local Relational Model (LRM) as a data model which is specifically designed for peer-to-peer applications. The objective of the model is to support semantic interoperability between relational databases in the absence of a global schema. The proposed model makes use of *domain relations* which are equivalent to the notion of mapping tables. However, unlike our approach, no provision is taken to manage domain relations.

Lenzerini [20] describes a general framework for modeling data integration applications. In particular, this framework can be used to represent peer-to-peer applications. The main difference between this approach and ours is that the former focuses on expressing constraints on the information contained in the peers (primarily using views) [22, 23], while we impose constraints only on the exchange of information between peers. As such,

our approach respects peer autonomy since it does not restrict the operation of peers in anyway beyond the agreement on values expressed in the tables.

Our experience with mapping tables shows that these can be used in support of value searches. The performance of value searches has received a great deal of attention [27, 24, 26]. Recently, researchers have proposed architectures for more advanced query processing in peer-to-peer systems (most notably Harren et al [19] and Gribble et al [17]). The proposal of Harren et al is similar in spirit to our own in that it is data model independent, relying only on peers to expose identifiers and optionally a set of descriptive attributes [19].

9. CONCLUSIONS

In this paper, we have considered the problem of managing collections of mapping tables. While such tables are used extensively in data-sharing systems, we are not aware of any other work that considers how these collections can be maintained and managed. Our results provide a first step in this direction.

In detail, we have discussed alternative semantics for mapping tables and we presented a language that allows the specification of mapping tables under different semantics. We have showed that by treating mapping tables as constraints on the exchange of information between peers we are able to reason about them and check their consistency. Given the high complexity of both the inference and the consistency problems, we have proposed an algorithm that, under realistic assumptions, solves the above problems efficiently. Our algorithm is fully implemented and in this work we have also presented experimental results which show its effectiveness.

In future work, we plan to study the behavior of a peer-to-peer system in which each peer continuously discovers alternative paths between itself and its acquaintances and it uses our algorithm to update its mapping tables. Furthermore, we intend to investigate the use of mapping tables in support of query answering.

10. REFERENCES

[1] GDB. http://www.gdb.org/.

[2] Gnutella. http://www.gnutelliums.com/.

[3] Hugo. http://www.gene.ucl.ac.uk/hugo/.

[4] Hyperion. http://www.cs.toronto.edu/db/hyperion/.

[5] Locus. http://www.ncbi.nlm.nih.gov/LocusLink/.

[6] MIM. http://www.ncbi.nlm.nih.gov/omim/.

[7] Napster. http://www.napster.com/.

[8] SWISS-PROT. http://www.ebi.ac.uk/swissprot/.

[9] Unigene. http://www.ncbi.nlm.nih.gov/UniGene/.

[10] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[11] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *WebDB*, 2002.

[12] M. L. Brodie and M. Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces, and the Incremental Approach*. Morgan Kaufmann, 1995.

[13] C.-C. K. Chang and H. Garcia-Molina. Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Sources. In *SIGMOD*, pages 335–346, 1999.

[14] B. Cooper and H. Garcia-Molina. Peer-to-peer data trading to preserve information. *TOIS*, 20(2):133–170, 2002.

[15] S. Davidson, G. C. Overton, and P. Buneman. Challenges in Integrating Biological Data Sources. *Journal of Computational Biology*, 2(4):557–572, 1995.

[16] L. G. DeMichiel. Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. *KDE*, 1(4):485–493, 1989.

[17] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What Can Databases Do for Peer-to-Peer? In *WebDB*, 2001.

[18] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. To appear in ICDE 2003.

[19] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-Based Peer-to-Peer Networks. In *IPTPS*, volume 2429 of *LNCS*, pages 242–250, 2002.

[20] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[21] B. Ludäscher, A. Gupta, and M. E. Martone. Model-Based Mediation with Domain Maps. In *ICDE*, pages 81–90, 2001.

[22] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy. Representing and Reasoning about Mappings between Domain Models. In *AAAI*, pages 80–86, 2002.

[23] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, Aug. 2002.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *SIGCOMM Conference*, pages 161–172, 2001.

[25] S. Sarawagi, editor. *IEEE Data Engineering Bulletin: Special Issue on Data Cleaning*, volume 23, Dec. 2000.

[26] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *SIGCOMM*, pages 149–160, 2001.

[27] B. Yang and H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. In *VLDB*, pages 561–570, 2001.