A Semantic Approach to Discovering Schema Mapping Expressions

Yuan An
University of Toronto
yuana@cs.toronto.edu

Alex Borgida
Rutgers University
borgida@cs.rutgers.edu

Renée J. Miller
John Mylopoulos
University of Toronto
{miller,jm}@cs.toronto.edu

Abstract

In many applications it is important to find a meaningful relationship between the schemas of a source and target database. This relationship is expressed in terms of declarative logical expressions called schema mappings. The more successful previous solutions have relied on inputs such as simple element correspondences between schemas in addition to local schema constraints such as keys and referential integrity. In this paper, we investigate the use of an alternate source of information about schemas, namely the presumed presence of semantics for each table, expressed in terms of a conceptual model (CM) associated with it. Our approach first compiles each CM into a graph and represents each table's semantics as a subtree in it. We then develop algorithms for discovering subgraphs that are plausible connections between those concepts/nodes in the CM graph that have attributes participating in element correspondences. A conceptual mapping candidate is now a pair of source and target subgraphs which are semantically similar. At the end, these are converted to expressions at the database level. We offer experimental results demonstrating that, for test cases of non-trivial mapping expressions involving schemas from a number of domains, the "semantic" approach outperforms the traditional technique in terms of recall and especially precision.

1 Introduction

In many applications, including data exchange, it is necessary to discover a meaningful relationship between a source and a target schema. Such a relationship is expressed by a collection of mapping expressions in a declarative language. The discovery problem is inherently difficult to automate, so interactive and semi-automatic tools are assumed to be the solution. Such a tool may employ a two-phase paradigm. First, specify some *simple correspondences* between schema elements; there are many tools that support such "matching" currently [16]. Then, derive plausible declarative mappings for users to select from. Systems that behave like this include TranSem [14], Clio [13, 15], HePToX [4] and MQG [11].

In this paper, we focus on the second problem, that of *deriving plausible declarative mapping expressions starting from element correspondences.* The element correspondences we consider will be quite simple: pairs of column names in the source and target relational schema, presumably signifying that data from the source column will contribute to data to appear in the target column. For example, in Figure 1, v_1 is a correspondence between column pname of table person and column aname of hasBookSoldAt.

Current solutions such as Clio [15] and MQG [11] rely on integrity constraints (especially referential integrity constraints) to assemble "logically connected elements", which then give rise to mappings between the tables.

However, as shown by the motivating examples below, this technique sometimes does not produce directly the most natural semantic connections, nor the most likely one, when there are several. In this paper, we investigate a complementary approach, which assumes an additional source of information, namely the *semantics* of the database schemas. To capture the semantics of a database schema, we use a conceptual model of the domain (abbreviated as CM), and a formal description of how it relates to the database schema.[1] We observe that obtaining the semantics of a schema is not necessarily a difficult task. For example, many database schemas are developed from a conceptual model, such as an Extended Entity-Relationship diagram. Consequently, keeping the EER schema and the mapping between the EER schema and the relational schema needs limited effort.[2] In addition, we have recently developed a tool [1, 2, 3] to recover the semantics of a legacy database schema in terms of an existing CM that covers roughly the same domain of discourse as the database. This could be quite useful, given the proliferation of ontologies and conceptual models motivated by visions of the Semantic Web.

It is important to note that we **do not assume** that the CMs for the source and target are identical, or are connected at the semantic level, as in many data integration proposals. Instead, we rely on the element correspondences between

[1] Such semantic specifications are found in [5, 2, 3], for example.

[2] In [2, 3], we have shown how to do this formally for standard designs.

the table columns, which have proven to be so useful for others.

We next clarify the input/output of the problem studied, including the notation for relational model and CM used in this paper, through several motivating examples.

Example 1.1: Consider the source relational schema given in the upper part of Figure 1. It contains five tables: person(pname), writes(pname, bid), book(bid), soldAt(bid, sid), and bookstore(sid). In a relational schema, the underlined column name(s), such as pname, indicates the primary key of each table. A dashed arrow represents a *Referential Integrity Constraint (RIC)*, e.g., a foreign key referencing a key. For example, the dashed arrow r_1 pointing from column pname of the table writes(pname,bid) to column pname of the table person(pname), written textually as writes.pname$\subseteq$ person.pname, indicates that the values in the former column are a subset of the latter.

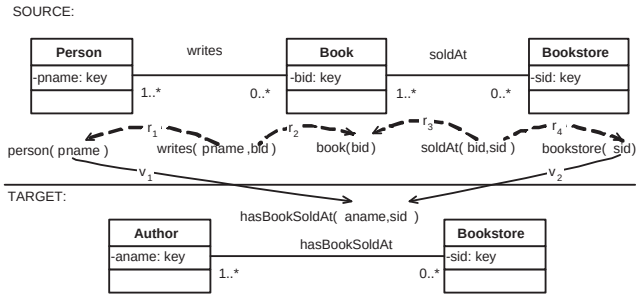

Figure 1. Relational Schemas, CMs, and Correspondences

The conceptual semantics of a database is normally specified using a conceptual modeling language, CML. In this paper, the CML captures common features of data models like EER and UML. Specifically, CML provides for *entity sets ("classes"), relationships between classes ("properties")*, and *attributes*, as well as *cardinality constraints* imposed on the participation in relationships. We use UML's notation for cardinality constraints on binary relations and their inverses, where $min..max$ specifies lower and upper bounds on the number of range objects related to a single domain object. Thus, $0..*$ means *"no constraint"*, $1..\_$ means *"total participation"*, while $\_..1$ indicates that each domain instance *functionally determines* the other participating instance in R. For instance, in Figure 1, a person writes 0 or more books, while a book is written by 1 or more persons. To encode constraints for identifying objects, we need a special **key** annotation to indicate (collections of) attributes that act as identifiers of entities. As illustrated in the next example, CML also supports subclass hierarchies.

Continuing with our example, a *target schema* is given in the lower part of Figure 1. The target schema contains, among others, a table hasBookSoldAt(aname,sid). The table is associated with the CM shown below it.

Now let us turn to the mapping task. To initiate the process, inter-schema correspondences need to be specified. We use the simplest form of correspondences discovered by most matchers, relating pairs of column names in the source and target. Figure 1 shows two correspondences using solid lines with arrows: v_1, connecting person.pname in the source to hasBookSoldAt.aname in the target, and v_2, connecting bookstore.sid in the source to hasBookSoldAt.sid in the target. Textually, a correspondence is written as person.pname$\leftrightsquigarrow$hasBookSoldAt.aname.

Current Solution The current solutions, which we call the *RIC-based techniques*, take as input the source schema, the target schema, database constraints (including keys, foreign keys, and more generally RICs), and the correspondences. In our examples, we will use an approach proposed in Clio [15], that is perhaps the most general of the solutions and generates GLAV mappings in the form of source-to-target tuple-generating dependencies [7]. Specifically, to generate a mapping expression, Clio uses an extension of the relational chase algorithm to first assemble logically connected elements into so-called *logical relations*. In this example, RICs r_1 and r_2 are applied to table writes(pname,bid) to produce the logical relation (expression):

S_1: person(pname) $\bowtie$ writes(pname, bid) $\bowtie$ book(bid).

Likewise, we can chase the table soldAt(bid,sid) using r_3 and r_4 to produce:

S_2: book(bid) $\bowtie$ soldAt(bid,sid) $\bowtie$ bookstore(sid).

In the target, a logical relation is:

T_1: hasBookSoldAt(aname,sid).

To interpret the correspondences, Clio looks at each pair of source and target logical relations, and checks which are *covered* by the pair. For example, the pair $\langle S_1, T_1\rangle$ covers v_1, while the pair $\langle S_2, T_1\rangle$ covers v_2. So the mappings are actually written as $\langle S_1, T_1, v_1\rangle$ and $\langle S_2, T_1, v_2\rangle$. The complete algorithm will then generate (among others) the following two candidate mapping expressions:

M_1: $\forall pname, bid.$(person($pname$)$\wedge$writes($pname, bid$) $\wedge$book(bid)$\rightarrow\exists x$hasBookSoldAt($pname, x$)).

M_2: $\forall bid, sid.$(book(bid)$\wedge$soldAt(bid, sid)$\wedge$bookstore(sid) $\rightarrow\exists y$hasBookSoldAt(y, sid)).

Since, in this example, the tables person(pname) and bookstore(bid) are also logical relations, then the following are also candidate mappings:

M_3: $\forall pname$(person($pname$)$\rightarrow\exists x$hasBookSoldAt($pname, x$)).

M_4: $\forall sid$(bookstore(sid)$\rightarrow\exists y$hasBookSoldAt(y, sid)).

Thereafter, all candidate mappings are presented to the user for further examination and debugging.

Note that the mappings M_1 through M_4 do not produce complete tuples in the target relations. Thus, when mappings are realized as queries (as in data exchange), Skolem

functions are generally used to represent existentially quantified variables [15]. In some cases, Skolem functions (and more complex mapping expressions like nested mappings) can be used to represent how data generated by different mappings should be merged [8]. However, no mapping generation algorithm that we are aware of would automatically generate a mapping that pairs authors with bookstores that stock their books, an interpretation we motivate below.

Alternate Solution We believe that the following is a more natural mapping expression in this case and should be generated as a candidate:

M_5: $\forall pname, bid, sid.$(person($pname$)$\wedge$writes($pname, bid$)
$\wedge$Book(bid)$\wedge$soldAt(bid, sid)$\wedge$bookstore(sid)
$\rightarrow$ hasBookSoldAt($pname, sid$)).

The mapping pairs in the source a person and a bookstore when the person writes a book and the book is sold at the bookstore. Looking into the semantics of the schemas, we observe that there is indeed a semantic connection between the classes Person and Bookstore, namely the composition of writes and soldAt.

Furthermore, note that the many-to-many cardinality constraint that can be inferred for the composed connection is compatible with that of the target relationship hasBookSoldAt. Contrast this to the hypothetical case when the upper bound of hasBookSoldAt would have been 1, indicating that each author is associated with at most one bookstore: we contend that such pairings are semantically incompatible, and do not lead to reasonable mapping expressions.

Note that the RIC-based techniques avoid generating lossy joins (like writes$\bowtie$soldAt), because these would provide an overabundance of logical relations, making the technique much less useful in practice. So any semantic solution must strictly limit, though not rule out, the use of such compositions. □

Example 1.2: CML supports the modeling of classes connected by ISA relationships, as well as *disjointness* and *completeness* constraints concerning the subclasses.

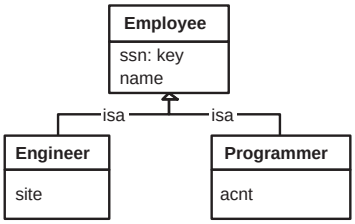

Figure 2. Using Rich Semantics in CM

Consider a CM, illustrated in Figure 2, with class Employee and two subclasses Engineer and Programmer, which are not disjoint, and cover the superclass. The bottom classes and their respective ISA relationships represent the semantics of the tables programmer(ssn,name,acnt) and engineer(ssn,name,site), forming the source schema. Suppose that the target database has schema employee(eid,name,site,acnt), and its CM is identical to Figure 2. These two databases represent alternative ways of encoding ISA hierarchies in relational tables, except for the fact that they use different identifiers, ssn and eid, as keys. Given correspondences that pair all columns with identical names (so ssn and eid do **not** correspond), the RIC-based techniques will suggest mappings $\langle$programmer, employee$\rangle$ and $\langle$engineer, employee$\rangle$, which will not merge the information about the engineer programmers. We would prefer instead a mapping that makes this connection and computes the outer-joins. This will be made possible by the presence of the superclass in the CM, which is absent in the database schema. □

Example 1.3: In addition to cardinality considerations, the CM may contain additional information useful in eliminating or prioritizing possible mappings. For example, consider a case resembling Example 1.2, where information about Departments and Faculty are encoded using different internal keys in the source and target db. If the source had two functional relationships, chairOf and deanOf, between Department and Faculty, while the target only had one, call it foo, then even considering cardinality constraints one cannot distinguish the two mapping candidates: $\langle$chairOf, foo$\rangle$ and $\langle$deanOf, foo$\rangle$. On the other hand, if the semantics indicates that chairOf and foo are **partOf** relationships (marked by filled-in diamond in UML), but deanOf is not, then the second mapping is less likely and can be eliminated or downgraded. □

The rest of the paper presents our principled approach to the problem of schema mapping using table semantics. In summary, the proposed algorithm will be seen to obtain better recall than RIC-based techniques by using ISA relationships not visible as RICs, and looking, *if necessary*, for "minimally lossy joins"; and better precision, by (i) eliminating certain candidate logical relations, like ones that cannot be consistently satisfied because of constraints such as disjointness, and (ii) eliminating mappings that pair relationships that have suspiciously different semantics. As Clio, our approach will generate GLAV mappings in the form of source-to-target tuple-generating dependencies.

The remaining content is organized as follows: Section 2 details the representation of table semantics. Section 3 describes the algorithm for mapping generation. Section 4 evaluates, using a set of experiments, the proposed approach in comparison to the techniques that use only constraints in the logical schema. Section 5 discusses related work. Finally, Section 6 presents conclusions and points to possible future directions.

2 Representing the Semantics of Schemas

We shall represent a given CM using a labeled directed graph, called a *CM graph*. We assume that attributes in CMs are simple and single-valued (composite and multi-valued attributes can be transformed into classes). We construct the CM graph from a CM described in the CML introduced in Example 1.1 as follows: We create a class node labeled with C for each class C, and a *directed edge* labeled with p from the class node C_1 to the class node C_2 for each binary relationship set p linking C_1 to C_2; for each such p, there is also an edge in the opposite direction for its *inverse*, labeled with p^-.

Note that we will deal with n-ary relationships, relationships with attributes, and so-called higher-order relationships (which relate relationships themselves) in Section 3.3 by reifying them. We shall eventually also reify many-to-many binary relationships since the algorithm will treat these the same way.

For each attribute f of a class C, create a separate attribute node, whose label is f, and also add an edge labeled with f from C to the attribute node. For each subclass C_1 of a class C_2, create an edge labeled with isa connecting C_1 to C_2 with cardinality $1..1$, and $0..1$ on the inverse. For the sake of succinctness, we use UML notation to represent a CM graph, placing the attributes inside the class rectangle nodes, and representing relationships and their inverses by a single undirected edge. The presence of such an (undirected) edge, labeled p, between classes C and D will be written in text as C ---p--- D. It will be important for our approach to distinguish *functional edges* — ones with upper bound cardinality of 1, and their composition: *functional paths*. If the relationship p is functional from C to D, we write C ---p->-- D.

In this paper, the semantics of a table is represented by a subtree in a CM graph. We call such a subtree a *semantic tree (or s-tree)*, where columns of the table associate uniquely with attribute nodes of the s-tree. This representation of table semantics was presented in [3], and corresponds to a subclass of conjunctive formulas (which can be derived from the s-tree). The encoding uses unary predicates for classes, binary predicates for attributes, and binary predicates for binary relationships. For relational schemas, we use a LAV-like logical formula of the form $T(X) \rightarrow \exists Y.\Phi(X,Y)$ to represent the semantics of table T, where T has columns X (which become arguments to its predicate), and Φ is a conjunctive formula over predicates representing an s-tree. The encoding introduces a new variable for every node in the tree, and proceeds recursively (see [3]). For example, the source table writes(pname,bid) in Figure 1, whose semantics is represented by the s-tree consisting of nodes Person and Book connected by edge writes, as well as the attributes pname and bid of Person and Book, respectively, has logical semantics

$$\mathcal{T}\text{:writes}(pname, bid) \rightarrow \mathcal{O}\text{:Person}(x), \mathcal{O}\text{:Book}(y), \mathcal{O}\text{:writes}(x,y), \mathcal{O}\text{:pname}(x, pname), \mathcal{O}\text{:bid}(y, bid).$$

where we use prefixes $\mathcal{T}$ and $\mathcal{O}$ to distinguish terms in the relational schema and the CM, and we omit the existential quantifiers on the right.

In order to handle multiple relationships between entities, as well as "recursive" relationships, while continuing to use trees, we duplicate concept nodes, and all the relationships in which they participate (see [3]). So, for example, the semantics of table pers(pid,name,age,spousePid) is represented by a graph with two nodes, Person and Person$_{copy1}$, connected by edge hasSpouse. And an additional column, pers.bestFriendPid, would require an additional node, Person$_{copy2}$, connected to Person by edge hasBestFriend. Note that this approach allows us to handle correctly *cyclic* RICs since the table semantics has to specify the number of times the loop has to be unfolded.

We remind the reader that there are well-known methodologies for designing logical database schemas from a CM, such as EER diagrams. We call such methodologies *er2rel designs* (e.g., [12]). Essentially, the er2rel design maps each class/entity to a (class/entity) table, and each relationship to a (relationship) table, with foreign keys to the participating entities. In addition, it also permits merging table T_1 into table T_2 when the key of T_1 has a foreign key to the key of T_2; such a merge reduces the need for joins, at the cost of possibly introducing null values in some columns of T_1. We can now assert that s-trees allow the encoding of the semantics of all tables obtained by er2rel design, and there are ways of dealing with more complex formulas Φ, but this is beyond the scope of this paper, since it requires using non-trees, which complicate matters considerably.

The previous study [3] also associates two additional notions with the semantics of a table T: (1) An *anchor*, which is the central object in the s-tree from which T is derived, if an er2rel design was used. For example, if T(c,d) was derived from a functional relationship C ---p->-- D, then C is the anchor of table T. (2) A rule expressing how classes involved in the s-tree of T are identified by columns of T. In the preceding example, class C is identified by the column c of T, while class D is identified by the column d. (More details about these can be found in [3].)

3 Generating Mapping Candidates

3.1 Description of the Problem

The input to our problem consists of (i) a source relational schema S and a target relational schema T; (ii) a CM ($\mathcal{G}_S$ and $\mathcal{G}_T$ respectively) associated with each relational schema (S and T resp.) via table semantic mappings; (iii) a set of correspondences $\mathcal{L}$ linking a set $\mathcal{L}(S)$ of columns in S to a set $\mathcal{L}(T)$ of columns in T. Assuming that $\mathcal{L}$ specifies pairwise "similar" table columns, we seek to find a pair

of expressions $\langle E_1, E_2 \rangle$ which are "semantically similar" in terms of modeling the subject matter.

As shown earlier, the table semantics relate each table in the schema to an s-tree in the respective CM graph, associating with each table column a class node in the graph through the bijective associations between columns and attribute nodes. Consequently, the set $\mathcal{L}(S)$ of columns gives rise to a set $\mathcal{C}_S$ of marked class nodes in the graph $\mathcal{G}_S$. Likewise, the set $\mathcal{L}(T)$ gives rise to a set $\mathcal{C}_T$ of marked class nodes in the graph $\mathcal{G}_T$. We call the s-trees associated with tables that have columns participating in $\mathcal{L}$ *pre-selected s-trees*. Our approach will consist of two major steps: (1) finding a subgraph D_1 connecting concept nodes in $\mathcal{C}_S$, and a subgraph D_2 connecting concept nodes in $\mathcal{C}_T$ such that D_1 and D_2 are "semantically similar" — we call these *conceptual subgraphs (CSG)*; (2) translating D_1 and D_2, including the relevant attribute nodes, into algebraic expressions E_1 and E_2, and returning the triple $\langle E_1, E_2, \mathcal{L}_M \rangle$ as a mapping candidate, where $\mathcal{L}_M \subseteq \mathcal{L}$ is the set of correspondences covered by the pair $\langle E_1, E_2 \rangle$.

In the rest of this section, we first present the CSGs for different situations and illustrate the algorithms for discovering them using examples. Then, we describe a process of translating a CSG into an algebraic expression by using the table semantics in terms of LAV expressions.

3.2 Basic Conceptual Model

We first consider basic constructs: classes and functional binary relationships, including ISA. We delay the treatment of all other kinds of relationships to the next subsection.

There are many ways to connect the marked nodes in $\mathcal{C}_S$ to create CSGs; similarly for $\mathcal{C}_T$. We propose to systematically explore the information encoded in the correspondences and the table semantics to discover a pair of similar CSGs. First, note that a node $v \in \mathcal{C}_S$ corresponds to a node $u \in \mathcal{C}_T$ when v and u have attributes that are associated with corresponding columns via the table semantics. Second, we take into consideration the following: (i) For a pair of nodes (v_1, v_2) in $\mathcal{C}_S$ and a pair of nodes (u_1, u_2) in $\mathcal{C}_T$, with v_1 corresponding to u_1 and v_2 corresponding to u_2, if there is to be a connection between v_1 and v_2 then it should be "semantically similar" or at least "compatible" to the connection between u_1 and u_2. The compatibility is decided by either the cardinality constraints of the connections, or the semantic type of the connections, e.g., **is-a** and **partOf**.[3] (ii) Since columns appearing in the same table are assumed to represent particularly relevant semantic connections between the concepts carrying the respective attributes, there is a preference that the CSGs use edges from the pre-selected s-trees. (iii) To the extent that there are choices available, we want the CSG to represent "intuitively meaningful concepts/queries". (iv) All things being equal, we want the CSG to be compact – as per Occam's principle.

In relational database, there appears to be consensus that observation (iii) favors the joins in the query to be lossless. Previous research on graphical querying of ER diagrams [18] indicates that *functional trees* in such diagrams correspond to lossless joins. Formally, a functional tree $\mathcal{F}$ containing a set of nodes $\{v_1, v_2, ..., v_n\}$ is a tree with a root u such that all paths from u are functional. (Such a tree is formally a Steiner tree: a spanning tree allowed to pass through additional nodes in order to reach marked nodes.) The preference for functional trees is motivated by the fact that functional properties in the CM determine functional dependencies, and hence the application of the er2rel design to a functional tree gives rise to a set of relational tables whose join is lossless. Combining this with observation (iv), we are led to seek *minimal functional trees* containing, as a subset, the marked nodes in $\mathcal{C}_S$ ($\mathcal{C}_T$). Interestingly, Wald and Sorenson [17], while considering the problem of querying ER diagrams, also suggested using minimal-cost Steiner trees, but in this case passing, if necessary, through non-functional edges, whose individual cost is greater than the sum of all the functional edges.

Note also that meaningful queries should not be equivalent to $false$, so we will eliminate CSGs that include an ISA edge from a class node C to its parent and then an ISA$^-$ edge to a node D corresponding to a disjoint subclass from C.

We now begin to present the algorithm, which starts by finding a CSG on one side and constructs a "semantically similar" CSG on the other side. For ease of presentation, we assume that we always start from the target side, and then try to find a similar CSG in the source. There are two subcases:

- Case A: The target CSG D_2 is known, e.g., it is the s-tree associated with a single table.
- Case B: The target CSG is to be constructed itself.

For the sake of illustration, the rest of the discussion is in terms of examples. But it should be noted that the essential steps of the formal algorithm are presented throughout the examples.

Case A. We use the following example to illustrate the construction of a similar CSG in the source when the target CSG D_2 is given.

Example 3.1: Consider an example involving a source schema with tables control(proj,dept) and manage(dept,mgr), and a target schema with a table proj(pnum, dept, emp). Suppose the correspondences given are v_1:control.proj⇝proj.pnum, v_2:control.dept⇝proj.dept, and v_3:manage.mgr⇝proj.emp. Figure 3 provides the semantics of target table proj as the graph, rooted at Proj, while the semantics of the source tables are subgraphs

[3] Such principles are well known in the ontology and CM integration literature.

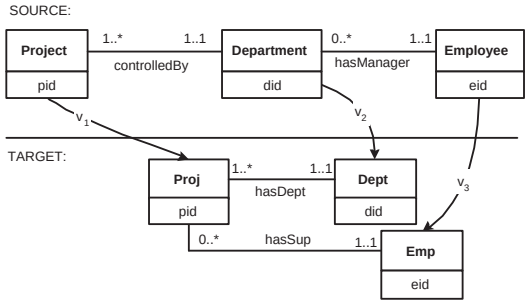

Figure 3. Input to Example 3.1

`Project` `---controlledBy->--` `Department` and `Department` `---hasManager->--` `Employee`

In Figure 3, the correspondences are lifted to correspondences between the associated class nodes.

Notice that the target CSG is an *anchored s-tree*, where the anchor is Proj, and the path from the anchor to every other node is functional. This leads us to believe that a "similar" CSG in the source should be a functional tree with a root corresponding to the anchor.

Case A.1 Suppose for the anchor Proj in the target we find a corresponding node in the source, in this case Project. Then, we try to connect it to every other node that has a correspondence to the target CM, (Department and Employee in this case) using minimal cost functional paths. Since observation (ii) above directs us to follow edges in pre-selected trees as much as possible, the edges in pre-selected trees do not contribute to the cost of functional paths. At last, we choose the functional tree(s) satisfying the following conditions as the CSGs in the source: (1) having the minimal cost and (2) containing the most number of edges in the pre-selected trees. Each of such functional trees is rooted at the node corresponding to the anchor of the target tree. In this example, the tree `Project` `---controlledBy->--` `Department` `---hasManager->--` `Employee` is the CSG in the source that matches the target CSG.

Case A.2 If a user only specified correspondences v_2 and v_3 (v_1 is missing), then we can no longer find a corresponding root in the source; we are nonetheless seeking an CSG in the source that is a functional tree. In this case, we look for all functional trees in the source that contain the nodes in $\mathcal{C}_S$. Such trees should be as small as possible, hence, minimal functional trees.

In this example, we would return the same anchored tree as above. Note that even if there were another class Intern in the source graph, and a functional relationship `Intern` `---works_on->--` `Project`, the functional tree rooted at Intern would not be returned because it is not minimal: the functional tree rooted at Project already contains the necessary nodes.

Note that it is this technique that finds the appropriate answer for Example 1.2.

Suppose that the nodes in C_S are not covered by a single (minimal) functional tree in the source CM graph. Then, in Case A.1, we connect as many nodes as possible using a single tree rooted at the node corresponding to the anchor and leave the rest unconnected. Consequently, the correspondences will be split among the tree and the remaining unconnected nodes. In Case A.2, we find the collection of trees covering different subsets of the nodes, and return each paired with the target CSG. □

Case B. Next, we consider the case where there are several pre-selected s-trees in the target and we want to connect them. Once again, we use the idea of minimal-cost functional trees to connect the marked nodes which belong to these pre-selected trees. Consequently, we construct a set of minimal functional trees in the target. Similarly, we can construct a set of minimal functional trees in the source. From these two sets we form pairs of CSGs by reverting to Case A, i.e., following heuristics such as matching the roots of tree pairs and seeking compatible connections.

3.3 Reified Relationships

In order to represent n-ary relationships ($n > 2$) in a CM like UML, one reifies them, introducing a special class connected to the participants using so-called "roles". For example, to represent that stores sell products to persons, we introduce class Sell, with functional properties/roles seller, buyer, sold pointing to classes Store, Person and Product respectively. (See Figure 4.) Such reified relationship nodes will be indicated in our text by tagging their name with $\diamondsuit$, although formally this can be encoded in the CM by making such classes be subclasses of a special top-level class ReifiedRelationship. Note that classes for reified relationships may also be used to attach descriptive attributes for relationships (e.g., dateOfPurchase). In fact, we need to use this modeling approach for binary relationships that have attributes. For ease of algorithm design, we have also chosen to represent many-to-many binary relationships, such as "person likes food", in reified form.

In terms of the formulas for table semantics, reified relationships are used in the standard way. For example, if we had table sells(sid,prodid,pid, date) whose semantics is represented by Figure 4, then the formula is specified as follows:

$\mathcal{T}$:sells$(sid, prodid, pid, date) \rightarrow \mathcal{O}$:Store$(x)$,
$\mathcal{O}$:Product(y), $\mathcal{O}$:Person(z), $\mathcal{O}$:Sell(s),
$\mathcal{O}$:seller(s, x), $\mathcal{O}$:buyer(s, z), $\mathcal{O}$:sold(s, y),
$\mathcal{O}$:sid(x, sid), $\mathcal{O}$:prodid$(y, prodid)$, $\mathcal{O}$:pid(z, pid),
$\mathcal{O}$:dateOfPurchase$(s, date)$.

Note that cardinality constraints $0/1..1$ on inverse roles can be used to indicate those cases where an object can participate at most once in a relation-

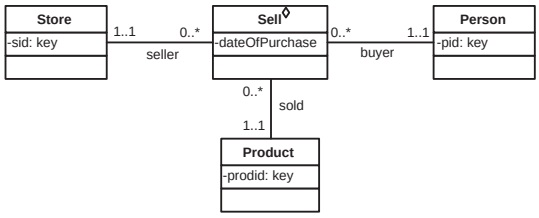

Figure 4. Reified Relationship Diagram

ship. Thus functional paths, such as `Project ---has_manager-->- Employee` can still be recognized in reified form as `Project -<--what⁻-->- Management◇ ---who-->- Employee`.

When a reified relationship node appears in a CM graph, we make several adjustments in the mapping algorithm.

First, a path of length two passing through a reified relationship node should be counted as a path of length 1, because a reified relationship could have been eliminated, leaving a single edge.

Second, the semantic category of a target tree rooted at a reified relationship induces *preferences* for similarly rooted (minimal) functional trees in the source. This includes the anchor being many-to-many, many-to-one or one-to-one (distinguished by the cardinality restrictions on the role inverses, as in the Management example above), the number of roles (exact arity), or subclass relationship to top-level ontology concept (e.g., partOf◇).

Third, note that non-functional relationships between entities in a CM can also be derived as the composition of edges on non-functional paths. For example, traversing the path `Person ---shopsAt--- Store ---location-->- City` yields a many-to-many relationship between persons and cities where the stores are located. Thus, in seeking matches in the source for a target (reified) many-to-many binary relationships between A and B, one must also consider the possibility that they appear as paths from A' to B' in the source that are not functional in either direction, where A' and B' are nodes in the source corresponding to A and B in the target, respectively. Note that using a single reified relationship as an anchor and extending this graph by functional paths from the roles, corresponds to lossless joins with the table representing the root; therefore such CSGs are preferred. More generally, we look for CSGs that minimize the number of lossy joins, by minimizing the number of *direction reversal changes* along each path.

Example 3.2 [Example 1.1 revisited]: The solution to the problem in Example 1.1 is then obtained as follows. The target s-tree in Figure 1 is a many-to-many relationship, which our algorithm represents as a reified relationship with anchor hasBookSoldAt◇. To find a matching CSG connecting the node Person in the source (corresponding to Author in the target) and the node Bookstore in the source (corresponding to Bookstore in the target), we look for paths connecting them that are not functional in either direction. Note that going from one role filler to another of a reified many-many binary relationship produces exactly such a path. In this case, no such single reified node can be found in the source. So we look for longer paths, obtaining the path from Person to Bookstore through writes◇, Book, and soldAt◇.

3.4 Obtaining Relational Expressions

The final mapping expression provides a pair of algebraic expressions using the tables in the input relational schemas only. Therefore, we need to translate the discovered CSGs in the CM graphs into algebraic expressions over the database schemas. Consider a CM as a collection of primitive relations/predicates for its concepts, attributes and properties. The semantics of a relational table associated with the CM is then a LAV expression over these predicates. Translating a discovered subgraph in the CM graph into expressions over tables associated with the CM graph becomes a query rewriting problem.

The first step of the translation is to express the CSG as a query using CM predicates. The encoding algorithm proposed in [3] can be used for this purpose. The following example illustrates this.

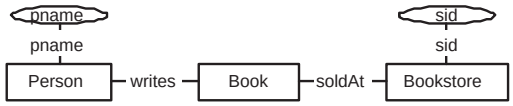

Figure 5. A Discovered Tree over a CM Graph

Example 3.3: Figure 5 is a fully specified CSG in the source CM of Example 1.1, with attribute nodes shown. (For simplicity of presentation, we revert to unreified binary relationships.) Taking Person as the root of the tree, the encoding algorithm recursively constructs a logic formula using unary predicates for the class nodes and binary predicates for the edges. An attribute node is encoded as a fresh variable in the formula and appears in the answer tuple. Assigning a name ans to the query, we obtain

q: ans(v_1,v_2) :- $\mathcal{O}$:Person(x_1), $\mathcal{O}$:pname(x_1, v_1),
$\mathcal{O}$:writes(x_1, x_2), $\mathcal{O}$:Book(x_2), $\mathcal{O}$:soldAt(x_2, x_3),
$\mathcal{O}$:Bookstore(x_3), $\mathcal{O}$:sid(x_3, v_2). □

Given the table semantics in terms of logical formulas, we rewrite the query q above to a new query q' which only mentions the tables in the relational schema by taking advantage of the object identifier information in the table semantics (see [6] for more sophisticated rewriting algorithm involving recursive query plans). The new query

q' will be faithful to the original query q, i.e., maximally-contained (see [9]) in q, and will mention tables that have columns linked by the correspondences.

Example 3.4: For the sake of completeness, we now briefly describe the rewriting process that makes use of the information about identifiers in the table semantics. In particular, we have proposed in [3] an ad-hoc approach to deriving inverse rules for each predicate in the CM, in terms of the tables in the schema. Formally, these are converted to Skolem functions, giving rise to formulas such as

$\mathcal{O}$:Person($f(pname, age)$) :- $\mathcal{T}$:person($pname, age$).

when inverting a semantic specification such as

$\mathcal{T}$:person($pname, age$) $\rightarrow$ $\mathcal{O}$:Person(x),
$\mathcal{O}$:hasName($x, pname$), $\mathcal{O}$:hasAge(x, age).

However, different tables give rise to different Skolem functions, which cannot then be joined. For this purpose, we use the *key* information about table semantics (see Section 2) in order to "merge" the various Skolem functions. So if we knew that hasName is the key of entity Person, and the formula Φ contains Person(x)$\wedge$ hasName(x, z) then we can in fact use z instead of x as the internal identifier, and treat hasName as the identity relation.

As a result, we can rewrite the query q in Example 3.3 to queries that mention tables only. In our case, these include the following:

q'_1: ans(v_1,v_2) :- $\mathcal{T}$:writes(v_1, y), $\mathcal{T}$:soldAT(y, v_2).
q'_2: ans(v_1,v_2) :- $\mathcal{T}$:Person(v_1), $\mathcal{T}$:writes(v_1, y), $\mathcal{T}$:Book(y), $\mathcal{T}$:soldAT(y, v_2), $\mathcal{T}$:Bookstore(v_2).
q'_3: ans(v_1,v_2) :- $\mathcal{T}$:Person(v_1) $\mathcal{T}$:writes(v_1, y), $\mathcal{T}$:soldAT(y, v_2), $\mathcal{T}$:Bookstore(v_2).

Since q'_1 does not mention tables person(pname) and bookstore(sid) that are linked by the correspondences, and q'_2 is contained in q'_3, q'_1 and q'_2 are eliminated. The body of the query q'_3, converted to relational algebra in the standard way, is returned as the algebraic expression. □

4 Experimental Results

We now report on experimental results that evaluate the performance of the proposed approach. We show that this approach works reasonably in a number of cases, and achieves better results than the RIC-based technique for discovering a complex mapping expression among marked elements in the schemas. The implementation is in Java and all experiments were performed on a PC-compatible machine with a Pentium IV 2.4GH CPU and 512MB memory.

Datasets: We considered a variety of domains. For each, a pair of relational schemas developed independently was used for testing. We ensured that the CMs associated with the pair of schemas were also mutually independent, by using different domain ontologies or the different ER conceptual models used for deriving the independent schemas. We describe them briefly below. All the schemas and CMs used in our experiments are available at the project's website. [4]

Schema	#tables	associated CM	#nodes in CM	#mappings tested	time (sec)
DBLP1	22	Bibliographic	75	6	0.072
DBLP2	9	DBLP2 ER	7		
Mondial1	28	factbook	52	5	0.424
Mondial2	26	mondial2 ER	26		
Amalgam1	15	amalgam1 ER	8	7	0.14
Amalgam2	27	amalgam2 ER	26		
3Sdb1	9	3Sdb1 ER	9	3	0.105
3Sdb2	9	3Sdb2 ER	11		
UTCS	8	KA onto.	105	2	0.384
UTDB	13	CS dept. onto.	62		
HotelA	6	hotelA onto.	7	5	0.158
HotelB	5	hotelB onto.	7		
NetworkA	18	networkA onto.	28	6	0.106
NetworkB	19	networkB onto.	27		

Table 1. Characteristics of Test Data

The first three pairs of schemas were obtained from Clio's test datasets [15]. DBLP 1&2 are the relational schemas for the DBLP bibliography. They are associated with the Bibliographic ontology and an ER model reverse engineered from the DBLP2 schema, respectively. Mondial 1&2 are databases about countries and their various features, where Mondial1 is associated with the CIA factbook ontology and Mondial2 is reverse engineered. Amalgam 1&2 are test schemas developed by students, and used in the Clio evaluations. They are associated with different conceptual models. The schemas 3Sdb 1&2 are two versions of a repository of data on biological samples explored during gene expression analysis [10]. UTCS and UTDB are databases for the CS department and the DB group at the University of Toronto. They were used in our previous study of semantics discovery, so their semantics are available now. Finally, we chose two pairs of ontologies from the I3CON conference.[5] These ontologies were used for the ontology alignment competition and demonstrate a certain degree of modeling heterogeneity. We forward engineered them into relational schemas for testing our techniques. As shown in Table 1, the test data have a variety of complexities.

Methodology: We compared the semantic approach, presented in this paper, with the RIC-based technique illustrated in Example 1.1, which creates logical relations by chasing constraints and derives mappings from pairs of source-target logical relations covering some correspondences. Since in its raw form, the chase generates maximal

[4] http://www.cs.toronto.edu/~yuana/research/maponto/schemaMapping
[5] http://www.atl.external.lmco.com/projects/ontology/i3con.html

sets of columns, we first applied a heuristic that removed any unnecessary joins — ones that did not introduce new attributes not covered by correspondences. (This is one optimization also described in [8].)

The comparison tries to focus on the intrinsic abilities of the methods. Each experiment consists of a manually created non-trivial "benchmark" mapping between some pair of schemas (a trivial mapping is from a single source table to a single target table), together with correspondences involving column names in it. These manually-created mappings are used to compare the mapping performance of the different methods.

Measures: We use $precision$ and $recall$ to measure the performance of the methods. For a given schema pair, let P be the set of mappings generated by a method for a given set of correspondences. Let R be the set of manually-created mappings for the same given set of correspondences. The two measures are computed as: $precision = \frac{|P \cap R|}{|P|}$ and $recall = \frac{|P \cap R|}{|R|}$. We compute the average precision and average recall over all tested mapping cases.

We believe it is instructive to give more details about how we calculate these measures. For each test case, R contains the manually-created non-trivial benchmark mapping expression consisting of a connection in the source and a connection in the target. In evaluating the generated mappings for each method, we seek for the same pair of connections, considering others that do not match the benchmark completely as "incorrect" mappings. For instance, in Example 1.1, even if there were target tables for author2, store2, and the RIC-based technique recovered mappings $\langle$person,author2$\rangle$ and $\langle$store,store2$\rangle$, recall and precision would have been 0 because no non-trivial mappings were found. (Note that the semantic method can also find trivial mappings.)

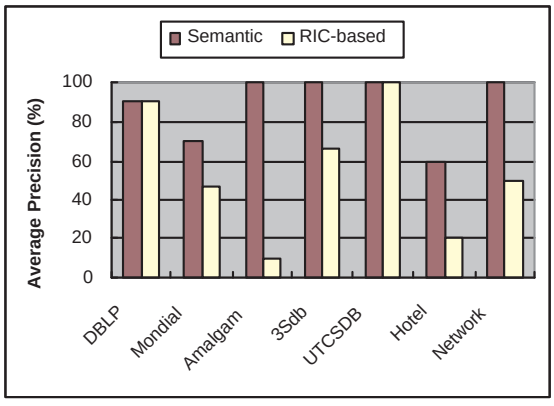

Figure 6. Average Precision

Results: First, the times used by the semantic approach for generating the mappings (in algebraic expressions) in the tested schemas are insignificant. The last column of Table 1 shows that it took less than one second. This is comparable with the RIC-based technique, which also took less than one second for mapping generation in our experiments. Next, in terms of the measures, Figure 6 compares the average precisions of semantic and the RIC-based technique for all the domains. Figure 7 compares the average recalls.

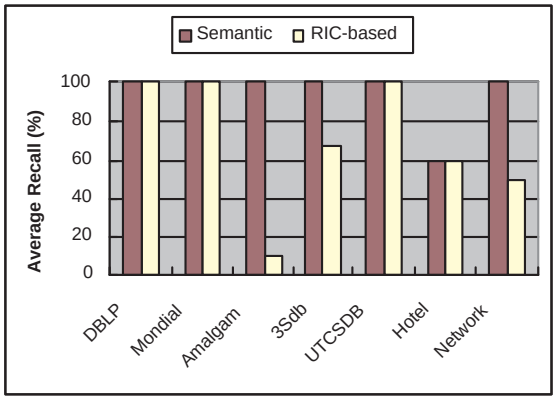

Figure 7. Average Recall

The results show that in general, the semantic approach performed at least as well as the RIC-based technique for the test datasets. The measures of recall show that the semantic approach did not miss any correct mappings that were predicted by the RIC-based technique (since it got *all* the mappings sought), and made significant improvements in some cases. Moreover, Figure 6 shows that the semantic approach had significantly improved precision in some cases.

Discussion: Many of the experimental schemas we have found do not have complicated semantics, and therefore would not provide differing results unless somewhat complex correspondences and mappings were sought. Most of the differences in this particular experimental set were due to situations such as the one illustrated in Example 1.2 (i.e., utilizing the semantics of ISA relationships). And it may be noteworthy that the schemas in Amalgam, where the semantic technique fared best, were not designed by professionals but by undergraduate database students.

5 Related Work

The most directly related work is obviously Clio [13, 15], and we have already provided some comparison of the basic techniques. As mentioned earlier, the work presented here can be thought of as increasing recall by, among others, slightly generalizing the use of RICs to repeatedly merging functional relationships onto the entities in the CM, where ISA is also treated as a functional relationship. It can also increase precision by eliminating candidate logical relations which cannot be consistently satisfied (e.g., because of disjointness constraints) and eliminating mappings that pair re-

lationships with suspiciously different semantics (many-to-many with many-to-one, **partOf** with non-**partOf**).

Conceptual models have been used in developing graphical query interfaces for databases. A central problem is inferring a query when a user has marked some nodes on a CM diagram. We have already noted that the solution in [18] applies the concept of maximal object from relational database theory to find a default connection among a set of nodes in a CM diagram. Assuming a user wants only one object to be used to infer a meaningful connection, the solution in [17] uses the "minimum cost" object for the connection. The fundamental difference from the above efforts is that we aim at finding a pair of matched connections in different CM graphs.

6 Conclusions

We have proposed here an approach to discovering mapping expressions between a pair of relational schemas, which starts from simple table column correspondences, and also utilizes the semantics of the tables, expressed through connections to conceptual models. We first showed several cases where the current solutions to discovering mappings from correspondences (based on referential integrity and key constraints) do not produce the best results. We then developed algorithms for discovering plausible mappings at the conceptual level, and translated them into sketches of relational level mappings. Intuitively, this algorithm replaces the use of database constraints by the notion of "minimal functional tree" in the CM, which, interestingly, appear to be related through the theory of Universal Relations and lossless joins. Experimental results demonstrated that the semantic approach achieved a generally better performance in recovering complex mapping expressions on test datasets drawn from a variety of domains.

Given the additional significant work that has gone into the Clio tool, and that Clio provides an approach for debugging the mapping, it may be best to view the present work as being complementary and embedded: if the semantics of the schemas are available or can be reconstructed with low cost using our own tool [1, 2, 3], then the present technique could be used inside Clio to provide some better candidate mappings. The exact details of such a merger remain to be worked out.

We are currently working on the representation of all conjunctive queries over the CM, as well as safe negation in s-trees/s-graphs. As shown in [3], a more careful look at the tree provides hints about when joins should really be treated as outer-joins (e.g., when the minimum cardinality of an edge being traversed is 0, not 1); such information could be quite useful in computing more accurate mappings, expressed as nested tuple-generating dependencies.

We also plan to investigate the related problem of finding complex semantic mappings between two CMs/ontologies, given a set of element correspondences.

References

[1] Y. An, A. Borgida, and J. Mylopoulos. Constructing Complex Semantic Mappings between XML Data and Ontologies. In *ISWC'05*, pages 6–20, 2005.

[2] Y. An, A. Borgida, and J. Mylopoulos. Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences. In *ODBASE'05*, pages 1152–1169, 2005.

[3] Y. An, A. Borgida, and J. Mylopoulos. Discovering the Semantics of Relational Tables through Mappings. *Journal on Data Semantics*, VII:1–32, 2006.

[4] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB'05*, pages 1267–1270, 2005.

[5] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data Integration in Data Warehousing. *J. of Coop. Info. Sys.*, 10(3):237–271, 2001.

[6] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49–73, 2000.

[7] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT'03*, pages 207–224, 2003.

[8] A. Fuxman, M. Hernandez, H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested Mappings: Schema Mapping Reloaded. In *VLDB'06*, pages 67–78, 2006.

[9] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001.

[10] L. Jiang, T. Topaloglou, A. Borgida, and J. Mylopoulos. Incorporating Goal Analysis in Database Design: A Case Study from Biological Data Management. In *RE'06*, pages 196–204, 2006.

[11] Z. Kedad and M. Bouzeghoub. Discovering View Expressions from a Multi-Source Information System. In *CoopIS'99*, pages 57–68, 1999.

[12] V. M. Markowitz and A. Shoshani. Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach. *ACM TODS*, 17(3):423–464, September 1992.

[13] R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB'00*, pages 77–88, 2000.

[14] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *VLDB'98*, pages 122–133, 1998.

[15] L. Popa, Y. Velegrakis, R. J. Miller, M. Hernandes, and R. Fagin. Translating Web Data. In *VLDB'02*, pages 598–609, 2002.

[16] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10:334–350, 2001.

[17] J. A. Wald and P. G. Sorenson. Resolving the Query Inference Problem Using Steiner Trees. *ACM TODS*, 9(3):348–368, 1984.

[18] Z. Zhang and A. O. Mendelzon. A Graphical Query Language for Entity-Relationship Databases. In *ER'83*, pages 441–448, 1983.