The Dissertation Committee for Aibo Tian
certifies that this is the approved version of the following dissertation:

# Automatic Data Integration with Generalized Mapping Definitions

Committee:

Daniel P. Miranker, Supervisor

Suzanne Barber

William R. Cook

Craig A. Knoblock

Raymond J. Mooney

Bruce W. Porter

# Automatic Data Integration with Generalized Mapping Definitions

**by**

**Aibo Tian, B.E.; M.S.Comp.Sci.**

**DISSERTATION**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**DOCTOR OF PHILOSOPHY**

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2014

To my parents and Wen.

# Acknowledgments

Five years, eighteen hundred days, forty thousand hours - that is how long it takes to become a PhD. I cannot reach this far without the guidance of my advisor, Daniel Miranker. Dan is encouraging and optimistic especially during my difficult time. He is like a friend, standing by my side, supporting me to achieve my goals. He is full of energy and patience. I am glad that I spent the past years working with him.

I would like to thank all my committee members Suzanne Barber, William Cook, Craig Knoblock, Raymond Mooney, and Bruce Porter for their insightful feedback and inspiring discussions.

I would also like to thank Peter Stone and Matthew Lease for helping me during my early PhD career. I had a lot of fun during the year working on robot soccer in Peter's lab. Matt devoted a big chunk of his time to help me extend a course project to a publication.

My research has been largely supported by the members of our research group. I want to thank Juan Sequeda, Mayank Kejriwal, Lee Thompson, and Nathan Clement for their valuable feedback on my research. Specifically, Juan is like my older brother. He is always energetic, and willing to help me on every aspect of my research. I also want to thank the undergraduates in our lab: Albert Haque, Slavcho Slavchev, and Colin Forage.

I would like to thank all my mentors during my internships in HP Labs and Google: Xuemei Zhang, David Ross, Yong Sheng, Pei Yin, and Arthur Asuncion. They gave me a taste of real industry problems.

I want to thank all my friends Xiaohu Shen, Hongkun Yang, Xue Chen, Na Meng, Chao Ruan, and Xu Wang for their support during my study. Particularly, I want to thank John Edwards for helping me when I first came here, and teaching me US cultures.

Finally, this work would not have been possible without the support of my parents and my girlfriend Wen Li. Particularly, Wen is always supportive and has faith in me. She always stands by my side to share both ups and downs during my PhD journey. She is my source of happiness.

Aibo Tian

*The University of Texas at Austin*
*August 2014*

# Automatic Data Integration with Generalized Mapping Definitions

Publication No. _____

Aibo Tian, Ph.D.
The University of Texas at Austin, 2014

Supervisor: Daniel P. Miranker

Data integration systems provide uniform access to a set of heterogeneous structured data sources. An essential component of a data integration system is the mapping between the federated data model and each data source. The scale of interconnect among data sources in the big data era is a new impetus for automating the mapping process. Despite decades of research on data integration, generating mappings still requires extensive labor.

The thesis of this research is that the progress on automatic data integration has been limited by a narrow definition of mapping. The common mapping process is to find correspondences between pairs of entities in the data models, and create logic expressions over the correspondences as executable mappings. This does not cover all issues in real world applications.

This research aims to overcome this problem in two ways: (1) generalize the common mapping definition for relational databases; (2) address

the problem in a more general framework, the Semantic Web. The Semantic Web provides flexible graph-based data models and reasoning capabilities as in knowledge representation systems. The new graph data model introduces opportunities for new mapping definitions. The comparison of mapping definitions and solutions for both relational databases and the Semantic Web is discussed.

In this dissertation, I propose two generalizations of mapping problems. First, the common schema matching definition for relational databases is generalized from finding correspondences between pairs of attributes to finding correspondences consisting of relations, attributes, and data values. This generalization solves real world issues that are not previously covered. The same generalization can be applied to ontology matching in the Semantic Web. The second piece of work generalizes the ontology mapping definition from finding correspondences between pairs of entities to pairs of graph paths (sequences of entities). As a path provides more context than a single entity, mapping between paths can solve two challenges in data integration: the missing mapping challenge and the ambiguous mapping challenge.

Combining the two proposed generalizations together, I demonstrate a complete data integration system using the Semantic Web techniques. The complete system includes the components of automatic ontology mapping and query reformulation, and semi-automatically federates the query results from multiple data sources.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Data integration systems provide uniform access to a set of heterogeneous structured data sources [29]. In large enterprises, data integration consumes about 40% of IT budget [14]. Data integration research has been continuously active for more than 30 years.

The research of data integration largely focuses on relational databases. Recently, the Semantic Web is emerging as a common framework to share and reuse data in the Web [17]. The promise of the Semantic Web is to represent data on the web as a globally linked database. In the Semantic Web, both data and metadata (ontology) are modeled as labeled directed graphs. In fact, those graphs are linked together as one graph. This representation provides flexibility to operate on both data and metadata. In this dissertation, both relational database schemas and ontologies are used to define data models.

An essential component of a data integration system is the mapping between the federated data model and each data source. The common mapping process contains two steps: (1) find correspondences between pairs of entities with the same type, called schema (ontology) matchings; (2) create logic expressions over the correspondences as executable mappings. This mapping

definition has the following limitations:

1. The matchings in step (1) are usually restricted to be between the same type of entities, which only cover part of real world cases.

2. Step (2) usually requires extensive labor.

3. The mappings are usually generated based on schema only, and may have ambiguity.

4. The two-step process propagates errors in both steps.

The thesis of this research is that the progress on automatic data integration has been limited by a narrow definition of mapping. The research aims to develop generalization of mapping definitions and resolve the listed limitations of the common mapping process. Since both relational databases and the Semantic Web are considered in this research, a comparison between them is discussed. The comparison covers both data models and mapping solutions.

The departure from the conventional mapping definition raises the challenge of finding test cases for evaluation. We collected test sets from four application domains for relational databases and three application domains for the Semantic Web. These test sets are available as public benchmarks. Requests for copies of these test sets have already been received.

## 1.1 Overview of Dissertation

This dissertation comprises three pieces of work. The first two pieces generalize the mapping problems in different aspects, and implement automatic systems for mapping generation. The first piece resolves limitation 1 of the common mapping process, and the second piece resolves limitations 2, 3, and 4. The third piece of work demonstrates a complete data integration system, using the two automatic systems for mapping.

### 1.1.1 Mapping Different Types of Database Elements

In relational database, schema matching is the task of finding correspondences between two database entities. Typically, automatic schema matching algorithms are only concerned with finding attribute correspondences. However, real world data integration problems often require matchings whose arguments span all three types of elements in relational databases: relation, attribute and data value. In a seminal paper, Krishnamurthy, Litwin, and Kent demonstrate this requirement using real world stock price databases as examples [54].

We introduce the definitions and semantics of three additional correspondence types concerning both schema and data values. These correspondences cover the higher-order mappings identified in [54]. It is shown that these correspondences can be automatically translated to tuple-generating dependency (TGD), which is the common mapping representation in data integration systems. We show that existing algorithms of generating universal

solutions for data exchange and reformulating queries for data integration can be applied to the TGDs representing the correspondences in this paper. Thus, this research is compatible with data integration applications that leverage TGDs.

Two methods for automatically identifying these correspondences are developed. One requires a limited number of duplicates across data sources. The other is a general instance-based method with no such requirement. Experiments conducted on four real world test sets demonstrate the effectiveness of the methods.

### 1.1.2 Mapping Paths in the Semantic Web

The second mapping system, QODI (short for Query-driven Ontology-based Data Integration), is designed for the Semantic Web. QODI is distinguished in that the ontology mapping algorithm dynamically determines a partial mapping specific to the reformulation of each query. The query provides application context not available in the ontologies alone; thereby the system is able to disambiguate mappings for different queries. Instead of representing the mappings as correspondences between ontology entities, QODI decomposes the query into a set of paths, and represents the mappings as correspondences between paths. Since the path similarity is not dependent on the precise alignment of entities, the missing mapping challenge is resolved. The path-based solution also simplifies the query reformulation problem as path traversal.

Given an input query, QODI decomposes the query into a set of paths, and searches for a subgraph of the source ontology, such that the set of path correspondences has the highest confidence. The confidence is measured using similarity of bag-of-words features. QODI exploits efficient heuristic search algorithms, which guarantee to find optimal solutions.

Using test sets from three real world applications, QODI achieves favorable results compared with AgreementMaker, a leading ontology matcher, and an ontology-based implementation of the mapping methods detailed for Clio, the state-of-the-art relational data integration and data exchange system.

### 1.1.3 Complete Data Integration System

In the Semantic Web literature, most existing research focuses on sub-problems of data integration, such as ontology matching [35], instance matching [69], and query rewriting [55]. Each sub-problem has its own assumptions that may not be compatible with others. Problems on complete data integration system have not been well studied.

This piece of work details the architecture of Alamo, a system with complete data integration workflow using the Semantic Web techniques. As a complete data integration system, Alamo can automatically generate executable ontology mappings in compile time, and reformulate queries in run time. It also supports federation of query results from multiple data sources.

Alamo combines both previously mentioned mapping systems to automatically generate ontology mappings. Thus, it can handle the use cases

that are covered by both systems. For query rewriting, Alamo uses the algorithm proposed by Le et. al., which guarantees to find an equivalent query reformulation [55]. Federation of query results from multiple data sources is done through the semi-automatic creation of entity name service (ENS), which groups instances that refer to the same real objects together.

## 1.2 Organization

This dissertation is organized as follows. Chapter 2 discusses the background knowledge and the comparison between relational databases and the Semantic Web. Chapter 3 describes our efforts of collecting test sets. Chapter 4 details the generalization of schema (ontology) matchings, and Chapter 5 details the generalization of ontology mappings. Chapter 6 demonstrates the complete data integration system in the Semantic Web. Finally, Chapter 7 discusses the future work, and Chapter 8 concludes the dissertation.

# Chapter 2

# Background

## 2.1 Data Models

### 2.1.1 Relational Database

**Relational Model** The relational data model is based on first-order predicate logic [7]. In the relational data model, data is represented in terms of *tuples* (rows), grouped into *relations* (tables). A *schema* is a finite set of relations. A relation is a finite set of *attributes*. A tuple is a finite set that contains a value for each attribute of a relation.

A relational database allows specification of some constraints. The two most common constraints are primary key, which is a set of attributes that functionally determines all other attributes in a relation, and foreign key, which is a set of attributes that enforces a link between two relations.

**SQL** The standard query language for a RDBMS is SQL. SQL is a first order language, which has capabilities close to that of relational algebra.

### 2.1.2 The Semantic Web

The Semantic Web technology consists of a set of languages: a graph data model (RDF) to publish data; an ontology (RDFS/OWL) to represent

knowledge based on Description Logic; and a query language (SPARQL). This set of languages forms an inheritance stack. RDFS inherits from RDF, and OWL inherits from RDFS. RDFS and OWL ontologies can be represented as RDF graphs.

**RDF** The Resource Description Framework (RDF) is a data model for representing information about resources on the world wide web [4]. The RDF specification considers three types of values: resource identifiers (URIs) to denote web resources, literals to denote values such as strings, and anonymous resources (blank nodes) which are existentially quantified variables that can be used to make statements about unknown (but existent) resources.

Formally, an RDF graph is a finite set of RDF triples, and an RDF triple is a tuple:

$$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$$

where $s$ is the *subject*, $p$ the *predicate* and $o$ the *object*. $U$, $L$, and $B$ denote the set of all URIs, the set of all literals and the set of all blank nodes, respectively. These three sets are disjoint.

*Example* 2.1.1. The statement "Database is taught by Dan" can be represented in RDF as the following triple:

$$(\text{course:database}, \text{course:teacher}, \text{people:dan})$$

where course:database, course:teacher, and people:dan represent URIs.

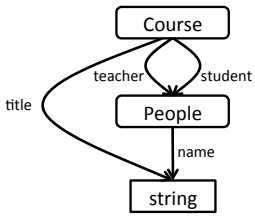



Figure 2.1: Ontology example. Round rectangles are classes, rectangles are datatypes, and edges represent properties.

**RDF Schema (RDFS)** RDF Schema extends RDF as a schema language for RDF and a lightweight ontology language [5]. RDFS defines three types of entities: *classes*, *object properties*, and *datatype properties*. Classes represent concepts in a domain. Object properties are relationships between classes. Datatype properties are relationships between classes and datatypes. RDFS also allows to state that a certain object is an instance of a certain class. In addition, RDFS can describe hierarchies between classes and properties in order to express semantic relationships. Finally, it is possible to relate the domain and range of a property to a certain class.

**Web Ontology Language (OWL)** The Web Ontology Language (OWL) is a language to describe ontologies with higher expressive power [3]. As extensions of RDF, both RDFS and OWL ontologies are directed labeled graphs.

*Example* 2.1.2. Figure 2.1 shows an example ontology. "Course" and "People" are classes. "teacher" and "student" are object properties. "title" and "name" are datatype properties. "string" is a datatype.

**SPARQL** SPARQL is the standard language for querying RDF data [6]. SPARQL is a graph pattern matching query language and is similar to SQL. It contains a set of triple patterns called basic graph patterns. Triple patterns are similar to RDF triples with the exception that the subject, predicate or object can be variables (denoted by a leading "?"). The result of a basic graph pattern query is a list of all variable bindings that cause a query pattern to match a subgraph of an RDF graph.

*Example* 2.1.3. Consider the following RDF triples:

(course:database, course:teacher, people:dan)

(course:database, course:title, "Database")

(people:dan, people:name, "Dan")

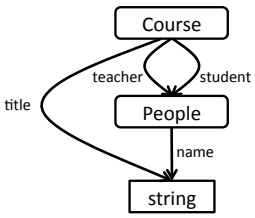
The following SPARQL query asks for the titles of courses that are taught by a professor, whose name is "Dan".

```
SELECT ?title
WHERE {
  ?course  course:teacher  ?teacher .
  ?teacher  people:name  "Dan" .
  ?course  course:title  ?title .
}
```

The basic graph pattern consists of three triple patterns. Matching these patterns with the RDF triples, the answer of the query is "Database".

```
CREATE TABLE Course(id int PRIMARY KEY,title varchar(255));
CREATE TABLE People(id int PRIMARY KEY,name varchar(255));
CREATE TABLE Teacher(c_id int,p_id int,
       FOREIGN KEY (c_id) REFERENCES Course(id),
       FOREIGN KEY (p_id) REFERENCES People(id));
CREATE TABLE Student(c_id int,p_id int,
       FOREIGN KEY (c_id) REFERENCES Course(id),
       FOREIGN KEY (p_id) REFERENCES People(id));
```

(a) A database in SQL.




(b) An equivalent ontology.

Figure 2.2: Example of translation between database and ontology.

### 2.1.3 Comparison

**Translation between Data Models** The data models of relational databases and the Semantic Web share similarity. Doan et. al. states that "relational schemas can be viewed as ontologies with restricted relationship types" [30].

Relational databases can be translated to RDF by direct mapping, which is a W3C standard [1]. Sequeda, Arenas, and Miranker introduce a direct mapping that extends the functionalities of W3C direct mappings to translate relational schemas to OWL ontologies [80]. The schema translation is defined as a set of Datalog rules (see Appendix 1). They are summarized as follows. A table is translated to a class unless the table represents a binary relationship, then it is translated to an object property. Foreign keys are translated to object properties while attributes are translated to datatype properties. These translations have been implemented in a system, Ultrawrap [81].

*Example* 2.1.4. The ontology in Figure 2.2b is translated from the relational

database in Figure 2.2a. The relations "Course" and "People" are translated to classes. The attributes "title" and "name" are translated to datatype properties. The binary relations "Teacher" and "Student" that only contain two foreign keys are translated to object properties.

**Expressive Power** An OWL ontology has more expressive power than a relational schema. Shvaiko and Euzenat assert that an ontology encompasses several data models, such as database schema and classifications [82]. One key difference is class hierarchy. An ontology can express class hierarchy through built-in property "rdfs:subClassOf", while a relational schema cannot. Specifically, an OWL ontology may represent a taxonomy, such as gene ontology[1].

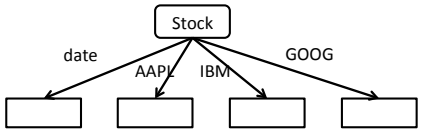
**Higher Order Queries** SQL, the standard query language in relational databases, cannot express high-order logic. For example, it is impossible to include variables for metadata, such as relations and attributes. Higher-order logic is important especially in data integration problems, as pointed out by Krishnamurthy, Litwin and Kent [54]. Federating views declaratively over heterogeneous databases require special, higher-order, syntactic features. Metadata-variables bridge schematic discrepancies where information in one source appears as explicit data and the same information in another source has been integrated into the schema definition.

[1]http://www.geneontology.org/

**Stock**

| date | AAPL | IBM | GOOG |
|---|---|---|---|
| 1/2/13 | 549 | 196 | 723 |
| 1/3/13 | 542 | 195 | 723 |

(a) Relational database




(b) Ontology

Figure 2.3: Example database and equivalent ontology of stock prices.

In the semantic web, both schema and data are represented as RDF graphs in triples. The standard query language, SPARQL, supports variables at any of the three positions in a triple pattern. Thus, SPARQL can express higher-order queries. Although the same higher-order logic issues in [54] are still higher-order in the Semantic Web, the federating SPARQL queries are first-order.

*Example* 2.1.5. Figure 2.3 shows a relational database of stock prices, and an ontology that is equivalent to the database. Suppose a user wants to ask "which stock has the price $196 on the date 1/2/13". This question cannot be formulated as a SQL query, because the variables are attributes. A SPARQL query can represent this question as:

```
SELECT ?code
WHERE {
  ?stock  ?code  "196" .
  ?stock  stock:date  "1/2/13" .
}
```

## 2.2 Executable Mappings for Data Integration and Data Exchange

A data integration system offers uniform access to a set of autonomous and heterogeneous data sources [29, 48, 25, 31, 84, 46]. Data exchange is the problem of transforming instances of a data source to instances of the target [38, 9]. In relational database, Clio is the state-of-the-art semi-automatic data integration and exchange system [37]. In the Semantic Web, Karma is a recent semi-automatic data exchange system [51]. Specifically, Karma is built to map structured data sources (relational databases, xml, etc.) to populate existing ontologies.

For both applications, the correspondences between entities are not enough, since they are not executable mappings (the ones that represent transformations of instances) [13]. Thus, how to represent, generate, and use executable mappings are interesting research topics.

### 2.2.1 Relational Database

In the data exchange community, the commonly used mapping representation is *tuple-generating dependency (TGD)* [11]. Let us denote the target schema as $\mathcal{T}$ and the source schema as $\mathcal{S}$. A TGD is in the form:

$$\forall \mathbf{x}(\phi_{\mathcal{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathcal{T}}(\mathbf{x}, \mathbf{y}))$$

where $\phi_{\mathcal{S}}(\mathbf{x})$ is a conjunction of atomic formulas over $\mathcal{S}$, $\psi_{\mathcal{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathcal{T}$.

Course

| title | instructor | Room |
|---|---|---|
| Machine learning | John | 101 |
| Database | Dan | 102 |

(a) Target database

Course

| id | title |
|---|---|
| c1 | Database |

People

| id | name |
|---|---|
| p1 | Albert |
| p2 | Peter |
| p3 | Dan |

Teacher

| c_id | p_id |
|---|---|
| c1 | p3 |

Student

| c_id | p_id |
|---|---|
| c1 | p1 |
| c1 | p2 |

(b) Source database

Figure 2.4: Database examples to illustrate TGD.

The generation of TGDs from correspondences of attributes has been studied in Clio [37]. Schema mapping in Clio is done in 2-steps: finding initial mappings between attributes; and associating mappings by logical inference through referential constraints. The mapping association is done by a modification of the chase algorithm [61].

*Example* 2.2.1. Consider the schema in Figure 2.4a as the target schema, and the schema in Figure 2.4b as the source schema. A TGD example is as follows:

$$\forall\, title,\ instructor\ (\ Course(c\_id, title)\ \wedge\ Teacher(c\_id, p\_id)\ \wedge\ People(p\_id, instructor) \rightarrow \exists\, room\ Course(title, instructor, room))$$

In data integration, the executable mappings can be represented in three formalisms. Global-as-view (GAV) represents each relation in the target schema as a view of the source schemas. Local-as-view (LAV) represents each relation in the source schemas as a view of the target schema [57]. The combination of GAV and LAV is Global-local-as-view (GLAV), which includes a

mapping from a view of the target schema to a view of the source schemas [39]. TGDs are equivalent in expressiveness to GLAV formalism.

### 2.2.2 The Semantic Web

Karma uses a similar logic representation as TGD to represent executable mappings [51]. The mapping process consists of three steps. First, find correspondences between columns in the source and classes and datatype properties in the target. Second, construct a graph based on the target ontology and the correspondences, and extract the minimal tree by a variation of the Steiner Tree algorithm [52]. Finally, the executable mappings are generated based on the minimal tree.

Rivero et. al. propose automatic algorithms to translate correspondences to language-independent executable mappings as construct SPARQL queries [78]. Specifically, for each entity correspondence, the algorithm expands each entity to a graph based on restrictions, and group all correspondences within the graphs as a kernel. The considered restrictions include domain, range, subclass, and subproperty. Finally, each kernel is transformed to a construct SPARQL query.

Correndo et. al. study the problem of query rewriting based on entity correspondences [22]. One intermediate step is to translate entity correspondences to an internal representation, which can be seen as executable mappings. The internal representation maps each triple in the target ontology to conjunction of triples in the source ontology. Given the internal representation,

a query rewriting algorithm is provided.

### 2.2.3 Comparison

Due to the similarity between the relational models and the data models in the Semantic Web, the algorithms to generate executable mappings may share similarity.

In Clio system, the matchings are associated by logical inference through referential constraints. Thus, part of the mapping process in Clio exploits an implicit graph representation of the relational schema. Recall in Section 2.1.3 that the translation from relational schemas to ontologies creates an explicit graph representation that includes integrity constraints. Thus, Clio's mapping association in a relational schema is similar to path following in an ontology. This implies the algorithms for the relational databases may be applied to the Semantic Web with a small number of changes.

## 2.3 Schema Matching and Ontology Matching

An essential component of data integration systems is schema matching (in relational database) [12] or ontology matching (in the Semantic Web) [35]. The schema matching problem is usually defined as finding correspondences between attributes of two schemas. Similarly, the ontology matching problem is defined as generating correspondences between entities of two ontologies.

Based on the rules of direct mapping, ontologies and schemas are aligned in the translation, such as relations to classes, attributes that are

| | Resources | | | Techniques | | | Input |
|---|---|---|---|---|---|---|---|
| | Schema | Instance | External | ML | User | Aggregate | Ontology otherwise Relational |
| COMA [27] | √ | | | | | √ | √ |
| GLUE [30] | √ | √ | | √ | | √ | √ |
| BayesOWL [70] | √ | | √ | √ | | | √ |
| OMEN [66] | √ | | | √ | | | √ |
| Falcon-AO [75] | √ | | | | | | √ |
| Gligorov et. al. [42] | | √ | √ | | | | √ |
| RiMOM [58] | √ | | | | | √ | √ |
| AgreementMaker [23] | √ | | | | √ | √ | √ |
| LogMap [49] | √ | | | | | | √ |
| Karma [51] | √ | √ | | √ | √ | | √ |
| Parundekar et. al. [71] | | √ | | | | | √ |
| Sarasua et. al. [79] | √ | √ | | | √ | | √ |
| Duan et. al. [32] | √ | √ | | √ | | | √ |
| QODI [Chapter 5] | √ | | √ | | | | √ |
| Alamo [Chapter 6] | √ | √ | √ | | | | √ |
| LSD [28] | √ | √ | | √ | | √ | |
| Cupid [60] | √ | | | | | | |
| SimFlood [64] | √ | | | | | | |
| Ichise et. al. [45] | | √ | | √ | | | |
| Kand et. al. [50] | | √ | | | | | |
| iMap [26] | √ | √ | √ | √ | √ | √ | |
| DUMAS [16] | | √ | | | | | |
| Madhavan et. al. [59] | √ | √ | √ | √ | | | |
| Bohannon et. al. [18] | | √ | | | | | |
| Warren et. al. [90] | | √ | | | | | |
| SPHINX [10] | | √ | | √ | √ | | |
| McCann et. al. [63] | √ | √ | | | √ | | |
| Clip [76] | √ | | | | √ | | |
| Muse [8] | √ | √ | | | √ | | |
| Dai et. al. [24] | | √ | | | | | |
| Elmeleegy et. al. [34] | | | √ | | | | |
| HAMSTER [68] | | | √ | | | | |
| MWeaver [74] | | √ | | | √ | | |
| Tian et. al. [Chapter 4] | √ | √ | | | | | |

Table 2.1: A list of schema (ontology) matchers.

foreign keys to object properties, and attributes that are not foreign keys to datatype properties. Thus, the schema matching and the ontology matching problems are similar. Recent review articles, one on schema matching [13] and the other on ontology matching [82], make note of the overlap between schema and ontology matching problems. As remarked by Shvaiko and Euzenat, " they often share similar matching solutions" [82].

### 2.3.1 Common Matching Techniques

Existing schema (ontology) matchers can be classified in different dimensions. In this dissertation, we consider three dimensions: resources, techniques, and inputs to matchers. The resources can be schema information, instances, and external knowledge. We consider three recent techniques: machine learning, user involvement, and matcher aggregation. The inputs to matchers can be either ontologies or relational schemas. Table 2.1 positions a list of schema matchers into the three dimensions.

**Schema-based** Schema contains many clues for matching. The two types of obvious clues are labels (name-based) and structures (structure-based).

The name-based techniques consider two entities as matched if they share similar labels, descriptions, or comments. Euzenat and Shvaiko list many string distances to measure similarity [35]. Techniques from information retrieval are used [75]. Different from natural language processing and information retrieval which mainly process natural languages, databases may contain computer-generated strings. Stemming techniques specific for computer-generated strings may be necessary for matching.

Structure-based techniques consider two entities as matched if they have similar relationships or have relationships to similar elements [13, 66]. Local information, such as similarities of subclasses and properties, can be aggregated [35]. Global structure can be used by propagating local structure information [64, 36].

**Instance-based** Instances provide confident evidences for matching. Constraint characterization can be performed to summarize rules from instances. The rules may include numerical value ranges, date, and phone numbers [77]. Entities with similar rules indicate possible matchings. Another direction is to use machine learning techniques and statistics to classify whether two sets of instances belong to the same class or not [28, 30, 26, 45] Machine learning methods are more flexible, since there are no predefined rules.

**External Knowledge** The information is called external knowledge, if it is neither from schema nor from instances. Different kinds of external knowledge can be incorporated to facilitate matching. Thesauri, such as WordNet [65], are commonly used to estimate string similarity [41]. Past matchings can also provide valuable information for future matchings [28, 27]. Information can be crawled from the web [59, 42, 70]. Database query logs can indicate hints for matching [34, 68].

**Machine Learning** Machine learning has been widely used in schema (ontology) matchers. The instance-based matching problem can be seen as a classification task of pairs of columns [28]. The aggregation of multiple matchers can be automatically learned [30]. Uncertainty in ontology matching has been studied [70, 66]. Active learning has been used to reduce human labor [10].

**User Involvement** Fully automatic matching systems may not achieve perfect accuracy. Thus, user interaction may be an essential part of some applica-

tions [76, 8, 10, 74]. Crowdsourcing is emerging as a platform to utilize human power, and is a great source for schema and ontology matchings [63, 79, 91].

**Matcher Aggregation** One matching algorithm may be only effective to limited real cases. Aggregating multiple models together is more robust than individual models [77, 28]. Recent matching systems are all built by combining basic matchers [58, 60, 23, 47, 27]. Automatic parameter tuning techniques are exploited to aggregate multiple models [62, 56, 72, 40].

### 2.3.2 Comparison

**Goals of Matching** The applications of schema matching for relational databases are always related to both schemas and the data under schemas. Although the schema mapping is expressed in terms of elements in the schema, the applications are always related to data.

The definition of ontologies is more general than database schemas. An ontology can be used to represent a knowledge base, a taxonomy, or a classification. The ontologies designed for these purposes may contain all valuable information within the schema. The knowledge in life science is usually represented as ontologies. In the OAEI contest, there is specifically a track to match adult mouse anatomy to a human anatomy (a part of NCI Thesaurus) [2]. These two ontologies themselves contain all the anatomy information. Given these ontologies, the purpose of the matching is not to facilitate the transferring of data. Instead, the matching itself can be used directly to link the

taxonomies or knowledge bases. Technically, these ontologies usually contain class hierarchy and other logic assertions which are not supported by relational schemas.

**Logic Reasoning** Compared with relational data models, an OWL ontology is capable of expressing more logic assertions. However, few ontology matchers exploit reasoning over the assertions to improve accuracy. LogMap is a matcher that is scalable and capable of reasoning [49]. LogMap represents both ontologies and matchings as Horn logic, and checks unsatisfiability. The scalability is achieved by indexing techniques.

# Chapter 3

# Test Data

This section details the collected test cases. The first set highlights the requirement of mapping different types of relational database elements. The second set includes systematically generated query sets in addition to ontologies and mapping groundtruth. The query sets enable the evaluation of ontology mapping specific for data integration applications. The test suites are available on our website[1].

## 3.1 Databases Requiring Mapping Different Types of Elements

Most of the existing schema matching test sets only contain examples of the common matching definition as correspondences between attributes. We identify test sets to highlight the requirement of mapping different types of relational database elements in real world problems. This test set comprises four application domains: Ecommerce, Stock Market, College Enrollment, and Video Game. Table 3.1 illustrates the statistics.

The Ecommerce test set contains two schemas that are in commercial

[1] http://ribs.csres.utexas.edu

| Dataset | Schema | # Attributes | # Tuples |
|---|---|---|---|
| Ecommerce | Opencart | 3 | 109 |
| | Subrion | 39 | 15 |
| Stock | Chwab | 4 | 250 |
| | Euter | 3 | 750 |
| | Ource | 2 | 250 |
| Enrollment | Statistics | 26 | 358 |
| | Ranking | 5 | 50 |
| | Ranking_2009 | 4 | 10 |
| Game | Vgchartz | 11 | 20000 |
| | Dbpedia | 3 | 48132 |

Table 3.1: Statistics of the relational test sets.

product_attribute

| product_id | attribute_name | language_name | text |
|---|---|---|---|
| 1 | exterior | English | silver |
| 1 | interior | English | other |
| 1 | engine | English | 3.5L V6 MPI SOHC 24V |
| 1 | fuel | English | gasoline |
| 1 | mileage | English | 42953 |

(a) Open source ecommerce software

autos

| id | mileage | exterior_color | engine | transmission | fuel_type | ... |
|---|---|---|---|---|---|---|
| 4 | 240000 | Blue | 6 cylinders | Manual | Gasoline | |
| 5 | 160000 | Black | 4 cylinders | Automatic | Gasoline | |
| 6 | 100000 | Blue | 4 cylinders | Automatic | Gasoline, Gas | ... |
| 7 | 32000 | Grey | 4 cylinders | Semi-automatic | Gasoline | |
| 9 | 150000 | Gold | 12 cylinders | Manual | Gasoline | |

(b) Auto classified software

Figure 3.1: Two commercial use schemas in our Ecommerce test set. This test set demonstrates the needs of correspondences between attributes and data values in real applications.

use. The first one is from the backend database of an open source ecommerce software[2], populated with used car listings from a car dealer website[3]. The

[2]http://www.opencart.com/

[3]http://www.centraltxautos.com/

second one is from the backend database of an automobile ecommerce software[4] that includes demo data. Both databases are very large, containing 115 and 37 tables respectively. We only take the product-attribute tables as shown in Figure 3.1. The schema from the second database contains 39 attributes in a single relation describing car information. The integer foreign keys are substituted with real values if each key refers to a unique value.

The Stock Market test set contains real stock prices of three IT companies (AAPL, GOOG, and IBM) in 2012, downloaded from Yahoo! Finance[5]. The data is transformed in the formats of the three real world schemas described in a seminal paper by Krishnamurthy, Litwin, and Kent [54].

The College Enrollment test set includes a schema of college enrollment statistics downloaded from the National Center for Education Statistics[6], and two schemas of college enrollment rankings crawled from Wikipedia[7]. The statistics schema contains the enrollment number of 358 colleges from 1990 to 2010. One ranking schema contains the top 10 enrolled colleges in 2009. The other combines the top 10 enrolled colleges from 2008 to 2012.

The Video Game test set includes a schema of the global sales of computer games crawled from VGChartz[8], and a schema of game information queried from DBpedia[9]. The schema from VGChartz contains 20000 games,

[4]http://www.subrion.com/product/autos.html
[5]http://finance.yahoo.com/
[6]http://nces.ed.gov/
[7]http://www.wikipedia.org/
[8]http://www.vgchartz.com/
[9]http://dbpedia.org/

| | Ecommerce | Stock | Enrollment | Game |
|---|---|---|---|---|
| attribute-attribute | 1 | 2 | 2 | 1 |
| attribute-value | 5 | 3 | 3 | 3 |
| relation-attribute | 0 | 1 | 1 | 0 |
| relation-value | 0 | 2 | 4 | 0 |
| total | 6 | 8 | 10 | 4 |

Table 3.2: The number of unique groundtruth correspondences of different types.

and the schema from DBpedia contains 48132 triples.

For all test sets, we manually determine groundtruth correspondences. Table 3.2 shows the number of unique groundtruth correspondences of different types.

## 3.2 Ontologies Associated with Query Workloads

Ontology mapping test sets usually only contain ontologies and mapping groundtruth. Although those test sets are able to evaluate mapping generation as a standalone application, they do not cover real use cases of mapping, such as data integration. Thus, we collect test sets that include both ontologies and queries associated to each ontology. The queries enable the evaluation of mapping generation for data integration applications.

This test set comprises three application domains: Life Science, Bibliography, and Conference Organization. The test cases include an ontology created by an international standards body, two ontologies created from direct mapping relational databases, and three ontologies used in OAEI [2]. Table 3.3

| Dataset | Ontology | Class | Class hierarchies | OP | DP |
|---|---|---|---|---|---|
| Life Science | DSW | 18 | 7 | 18 | 53 |
| | SPECIFY | 11 | 0 | 33 | 380 |
| Bibliography | DBLP | 17 | 7 | 9 | 42 |
| | UMBC | 15 | 11 | 4 | 42 |
| Conference | SIGKDD | 49 | 39 | 14 | 11 |
| | SOFSEM | 60 | 46 | 29 | 22 |

Table 3.3: Statistics of the test sets. OP and DP represent object property and datatype property, respectively.

shows the statistics.

The Life Science domain consists of Darwin Core and Specify. Darwin Core is an ontology at the center of the standardization efforts of the Global Biodiversity Information Foundation (GBIF), an organization concerned with cataloging the impacts of climate change. Darwin Core contains 18 classes, and 71 properties. The Specify ontology was created from direct mapping the SQL schema of the database in the Specify biological collections software package[10]. Specify is used to manage over 200 field specimen collections. The specify ontology has 11 classes, and 413 properties. The Bibliography domain comprises the UMBC ontology from OAEI, and an ontology that models DBLP, generated from the direct mapping of a relational database of DBLP metadata through Ultrawrap [81]. Class hierarchies are manually added. DBLP ontology has 17 classes, and 51 properties. The Conference domain consists of two ontologies SIGKDD and SOFSEM from OAEI.

[10]http://specifysoftware.org/

```
BASE <http://ribs.csres.utexas.edu/specify/>
Select ?v
Where {
 ?c0 <locality#Latitude1> ?v.
 ?c0 rdf:type <locality>.
}
```

(a) PathOnly query, asking for the latitude of all locations.

```
BASE <http://ribs.csres.utexas.edu/specify/>
Select ?v0 ?v1 ?v2 ?v3 ?v4 ?v5 ?v6
Where {
 ?c0 <determination#DeterminedDate> ?v0.
 ?c0 <determination#Remarks> ?v2.
 ?c1 <taxon#Name> ?v3.
 ?c2 <taxon#Name> ?v4.
 ?c3 <agent#DateOfBirth> ?v5
 ?c4 <agent#DateOfBirth> ?v6.
 ?c1 rdf:type <taxon>.
 ?c3 rdf:type <agent>.
 ?c0 <determination#Qualifier> ?v1.
 ?c0 <determination#ref-TaxonID> ?c1.
 ?c0 <determination#ref-PreferredTaxonID> ?c2.
 ?c0 <determination#ref-CreatedByAgentID> ?c3.
 ?c0 <determination#ref-ModifiedByAgentID> ?c4.
 ?c0 rdf:type <determination>.
 ?c2 rdf:type <taxon>.
 ?c4 rdf:type <agent>.
}
```

(b) ClassAll query, asking for the dates, remarks, and qualifiers of all determination of taxons, as well as the birthdays of the agents that determine the taxons.

Figure 3.2: Real SPARQL queries generated for the Specify ontology.

Sets of test queries are created as follows. First, we identify groundtruth path mappings between each pair of ontologies. Subsequently, a computer program systematically generates two kinds of SPARQL queries for each ontology. 1) A *PathOnly* query has a query graph consisting of only one path in the groundtruth. 2) A *ClassAll* query has a query graph consisting of all paths (at least two) that share a source in the groundtruth. A ClassAll query is the most complicated query with one conjunction over the source. In English specification, a PathOnly query asks for all values of a single attribute of a concept, and a ClassAll query asks for all values of all attributes of a concept. Figure 3.2 shows examples of real PathOnly and ClassAll queries generated for the Specify ontology, as well as the meaning of both queries.

# Chapter 4

# Mapping Different Types of Database Elements

Schema matching is an essential component of information integration systems [12, 13]. Most existing automatic schema matchers define the matching problem as determining correspondences between attributes. This definition has practical limitations. In a seminal paper, Krishnamurthy, Litwin, and Kent show that interoperability of databases requires higher-order capabilities over both metadata and data [54]. They assert that higher-order schematic discrepancies are frequent. We observe this problem in our experiences of integrating two commercial ecommerce databases. Figure 4.2 shows two car listing databases. One is from a general purpose open source ecommerce software that has been downloaded more than 300,000 times[1]. The other is an ecommerce software specific to automobiles[2]. The fifth row of the first table contains "mileage" as a data value and the number "42953" as another data value. In the second database, "mileage" is the name of the second column, and the numbers are data values in that column. The attribute correspondences cannot cover this matching of mileage.

[1] www.opencart.com
[2] www.subrion.com

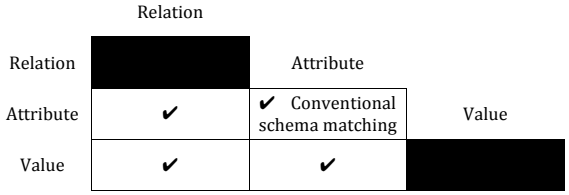



Figure 4.1: Six possible correspondence types. The checked correspondence types are detailed in this work. The definition of correspondence is based on a given pair of relations, so correspondence between relations is ignored. Correspondence between values does not involve metadata; hence it is not considered.

product_attribute

| product_id | attribute_name | language_name | text |
|---|---|---|---|
| 1 | exterior | English | silver |
| 1 | interior | English | other |
| 1 | engine | English | 3.5L V6 MPI SOHC 24V |
| 1 | fuel | English | gasoline |
| 1 | mileage | English | 42953 |

(a) Open source ecommerce software

autos

| id | mileage | exterior_color | engine | transmission | fuel_type | ... |
|---|---|---|---|---|---|---|
| 4 | 240000 | Blue | 6 cylinders | Manual | Gasoline | |
| 5 | 160000 | Black | 4 cylinders | Automatic | Gasoline | |
| 6 | 100000 | Blue | 4 cylinders | Automatic | Gasoline, Gas | ... |
| 7 | 32000 | Grey | 4 cylinders | Semi-automatic | Gasoline | |
| 9 | 150000 | Gold | 12 cylinders | Manual | Gasoline | |

(b) Auto classified software

Figure 4.2: Two commercial use schemas in our Ecommerce data set. This data set demonstrates the needs of correspondences between attributes and data values in real applications.

A relational database contains three types of elements: relation, attribute, and data value. Any pair of elements can form a correspondence, resulting in six possible correspondence types (see Figure 4.1). The conventional schema matching definition only considers one possibility. We call the correspondences between elements of the same type as *same-type correspondences*, and similarly *different-type correspondences*.

**Stock**

| date | stkCode | clsPrice |
|---|---|---|
| 1/2/13 | AAPL | 549 |
| 1/2/13 | IBM | 196 |

(a) Database D: stock as [D]ata value

**Stock**

| date | AAPL | IBM | GOOG |
|---|---|---|---|
| 1/2/13 | 549 | 196 | 723 |
| 1/3/13 | 542 | 195 | 723 |

(b) Database C: stock as attribute ([C]olumn)

**IBM**

| date | clsPrice |
|---|---|
| 1/2/13 | 196 |
| 1/3/13 | 195 |

(c) Database R: stock as [R]elation

Figure 4.3: Example databases of stock prices.

Different-type correspondences are frequently required in real world data integration applications. In our experiences of integrating the two car listing databases, we observe that without identifying different-type correspondences the matching accuracy cannot exceed 0.81. The open source ecommerce software has 115 relations, and the automobile ecommerce software has 37 relations. After manually inspecting the databases to find correspondences concerning products and transactions, we find 26 correspondences in total, and 5 of them are different-type correspondences. Especially, these different-type correspondences match the product-property tables as shown in Figure 4.2, which are the most important matchings to integrate the two databases. As a general purpose ecommerce software, the first database does not have prior information of products. Thus the car properties, such as mileage and color, are designed as data values. The second database is specific for cars, so the properties can be designed as attributes.

Figure 4.3 shows three stock price databases from the seminal paper, which are asserted in commercial use [54]. We will use these databases as examples throughout the work. The three databases contain similar information, but are organized differently. For example, the stock "IBM" is a [D]ata value in database D, an attribute ([C]olumn) in database C, and a [R]elation in database R. An element in a database is denoted as its name prepended by the database name and a colon, e.g. "C:IBM".

The same-type correspondence between two attributes $a$ and $b$ has clear semantics. If two tuples $s$ and $t$ satisfy the correspondence, the value of $a$ in $s$ equals the value of $b$ in $t$. The semantics of different-type correspondences are not clear. In Figure 4.3, correspondence between attribute "C:IBM" and data value "D:IBM" indicates a relationship between static metadata and dynamic data. The meaning of such a relationship is not obvious.

We observe that a different-type correspondence takes on clear semantics if combined with an appropriate same-type correspondence. For example, consider the different-type correspondence between attribute "C:IBM" and data value "D:IBM". When combined with the same-type correspondence between the data values of attributes "C:IBM" and "D:clsPrice", it represents the matching of IBM stock prices. "IBM" and its stock price "196" in both databases can now be matched by specifying this *compound correspondence*.

In this work, the definitions and semantics of three types of *compound correspondences* are introduced [85]. The compound correspondence types are named by their different-type correspondences as attribute-value, relation-

attribute, or relation-value. The checked cells (except correspondence between attributes) in Figure 4.1 represent these different-type correspondences.

Two algorithms to automatically generate compound correspondences are detailed. Both algorithms exploit database instances. One algorithm generates correspondences using duplicates in the two databases, an idea demonstrated to be effective in the DUMAS system [16]. The other algorithm is more general in that it does not use duplicates. Comparing with DUMAS and an instance-based schema matcher, experiments conducted on four real world data sets demonstrate the effectiveness of the two methods.

A common schema mapping representation in information integration applications is *tuple-generating dependency (TGD)*. To demonstrate that compound correspondences can be used in both data exchange and data integration applications, we propose algorithms to automatically translate these correspondences to TGDs with constants. We slightly change the chase algorithm [61] to generate universal solutions based on these TGDs for data exchange. Further more, discussion on applying existing query rewriting algorithms to these TGDs for data integration is provided.

## 4.1 Problem Definition

This work considers relational data models. A schema is a finite set of relations. We denote schemas as $\mathcal{A}, \mathcal{B}$, and relations as $A, B$. A relation is a finite set of attributes. A tuple is a finite set that contains a value for each attribute. Attributes are represented as $a, b$, and tuples are represented as $s, t$.

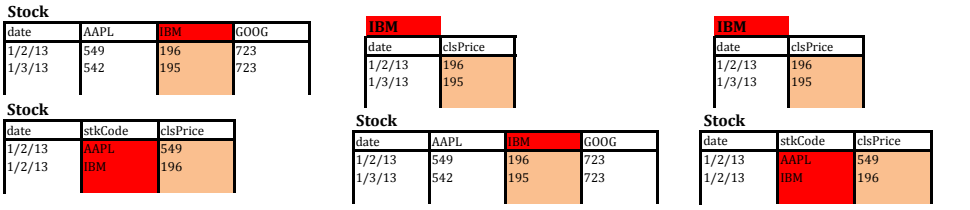
Given an attribute $a$ and a tuple $t$, the value of $a$ in $t$ is denoted as $t^a$.

In order to define compound correspondences, we introduce the term *database element set*. Intuitively, the database element set of a relation $A$ is the set of all elements of $A$ that can occur in correspondences. A database element set contains relation names, attribute names, and data values. Introducing data values in correspondences raises the issue of how to represent data values. A straightforward way is to use actual data values. However, data values are dynamic and can change frequently. A second way, which is used in this work, is to use a symbolic representation. We use notation $\overline{a}$ to represent the set of all possible data values of attribute $a$. For example, $\overline{\text{D:stkCode}}$ represents "AAPL", "IBM" and other stock codes. The program using the correspondences needs to match the exact data values.

**Definition 4.1.1** (database element set)**.** Given a relation $A$, and the set of attributes $\{a_i | a_i \in A, i = 1, \ldots, |A|\}$, the database element set is $\Theta_A$, where $\Theta_A = \{A\} \cup \{a_i | a_i \in A\} \cup \{\overline{a_i} | a_i \in A\}$. $A$ is used interchangeably to represent the name of relation $A$ and the set of all attributes in $A$, and $\overline{a_i}$ is the symbol representing the set of all possible data values of attribute $a_i$.

To define compound correspondence, we first define *primitive correspondence* as a correspondence between two elements in the database element sets.

**Definition 4.1.2** (primitive correspondence)**.** Given two relations $A$ and $B$, a primitive correspondence is a pair $(e_1, e_2)$, where $e_1 \in \Theta_A$, and $e_2 \in \Theta_B$. $\Theta_A$




(a) Attribute-value (b) Relation-attribute (c) Relation-value

Figure 4.4: Shading indicates compound correspondences.

and $\Theta_B$ are database element sets of $A$ and $B$, respectively.

A compound correspondence is a pair of primitive correspondences, with the constraint that the second element is always defined as a primitive correspondence between data values.

**Definition 4.1.3** (compound correspondence)**.** Given two relations $A$ and $B$, the three types of compound correspondences are defined as follows:

- Attribute-value correspondence: for an attribute $a \in A$ and attributes $b_1, b_2 \in B$, an attribute-value correspondence is a pair $((a, \overline{b_1}), (\overline{a}, \overline{b_2}))$, where $(a, \overline{b_1})$ and $(\overline{a}, \overline{b_2})$ are primitive correspondences;
- Relation-attribute correspondence: for an attribute $a \in A$ and an attribute $b \in B$, a relation-attribute correspondence is a pair $((A, b), (\overline{a}, \overline{b}))$, where $(A, b)$ and $(\overline{a}, \overline{b})$ are primitive correspondences;
- Relation-value correspondence: for an attribute $a \in A$ and attributes $b_1, b_2 \in B$, a relation-value correspondence is a pair $((A, \overline{b_1}), (\overline{a}, \overline{b_2}))$, where $(A, \overline{b_1})$ and $(\overline{a}, \overline{b_2})$ are primitive correspondences.

Figure 4.4 illustrates some compound correspondences of the stock price examples. $((\text{C:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{C:IBM}}, \overline{\text{D:clsPrice}}))$ is an attribute-value correspondence. $((\text{R:IBM}, \text{C:IBM}), (\overline{\text{R:clsPrice}}, \overline{\text{C:IBM}}))$ is a relation-attribute correspondence, and $((\text{R:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{R:clsPrice}}, \overline{\text{D:clsPrice}}))$ is a relation-value correspondence.

The semantics of compound correspondences are defined as follows. For completeness, the semantics of the conventional attribute-attribute correspondence are also presented. The notation in Definition 4.1.3 is employed. $s$ and $t$ represent tuples of relations $A$ and $B$. The value of an attribute $a$ in $s$ is denoted $s^a$.

- Attribute-attribute correspondence $(a, b)$

  $s$ and $t$ satisfy correspondence $(a, b)$, if $s^a = t^b$.

  *Example:* (C:date, D:date) is an attribute-attribute correspondence. The first tuple of database C and the first tuple of database D satisfy the correspondence.

- Attribute-value correspondence $((a, \overline{b_1}), (\overline{a}, \overline{b_2}))$

  $s$ and $t$ satisfy correspondence $((a, \overline{b_1}), (\overline{a}, \overline{b_2}))$, if $a = t^{b_1}$ and $s^a = t^{b_2}$.

  *Example:* The first tuple of database C and the second tuple of database D satisfy $((\text{C:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{C:IBM}}, \overline{\text{D:clsPrice}}))$, since "IBM" occurs as the name of attribute "C:IBM" and as the value of attribute "D:stkCode", and "196" occurs as the value of both attributes "C:IBM" and "D:clsPrice".

- Relation-attribute correspondence $((A, b), (\overline{a}, \overline{b}))$

  $s$ and $t$ satisfy correspondence $((A, b), (\overline{a}, \overline{b}))$, if $A = b$ and $s^a = t^b$.

  *Example:* The first tuple of database R and the first tuple of database C satisfy $((\text{R:IBM}, \text{C:IBM}), (\overline{\text{R:clsPrice}}, \overline{\text{C:IBM}}))$, because "IBM" occurs as the name of relation "R:IBM" and as the name of attribute "C:IBM", and "196" occurs as the value of both attributes "R:clsPrice" and "C:IBM".

- Relation-value correspondence $((A, \overline{b_1}), (\overline{a}, \overline{b_2}))$

  $s$ and $t$ satisfy correspondence $((A, \overline{b_1}), (\overline{a}, \overline{b_2}))$, if $A = t^{b_1}$ and $s^a = t^{b_2}$.

  *Example:* The first tuple of database R and the second tuple of database D satisfy $((\text{R:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{R:clsPrice}}, \overline{\text{D:clsPrice}}))$, since "IBM" occurs as the name of relation "R:IBM" and as the value of attribute "D:stkCode", and "196" occurs as the value of both attributes "R:clsPrice" and "D:clsPrice".

The semantics of attribute-value and relation-value correspondences involve two data values in tuple $t$ at the same time. These semantics show the dependency between the two primitive correspondences in a compound correspondence.

Given the definition of correspondences, a *schema matching* is a set of *relation matchings*, each of which includes a set of correspondences.

**Definition 4.1.4** (relation matching)**.** Given two relations $A$ and $B$, a relation matching $M$ is a triple $(A, B, \Sigma)$, where $\Sigma$ is a set of correspondences between

$A$ and $B$.

**Definition 4.1.5** (schema matching)**.** Given two schemas $\mathcal{A}$ and $\mathcal{B}$, a schema matching $\mathcal{M}$ is a set of relation matchings, each of which is between a relation of $\mathcal{A}$ and a relation of $\mathcal{B}$.

## 4.2 Generating Correspondences

Data dependencies make automatically generating compound correspondences more difficult than the conventional attribute-attribute correspondences. For generating attribute-attribute correspondences, instance-based matchers need to only determine if two attributes have similar values. For compound correspondences, dependencies may not be evident for all values of an attribute. For example, to generate $((\text{C:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{C:IBM}}, \overline{\text{D:clsPrice}}))$, a matcher needs to compare the values of attribute "C:IBM" and the values of "D:clsPrice" when the values of "D:stdCode" is "IBM" for the same tuples.

Two algorithms are proposed to automatically generate correspondences. The first algorithm relies on the existence of duplicates in two databases. The second is a general instance-based method. Both methods require basic similarity measures. $S_{field}$ is a measure of similarity between two database elements. $S_{list}$ measures the similarity between two lists of elements. Similarity measures have been well studied [21]. Any similarity measure can be used.

### 4.2.1 Duplicate Method

The idea of the duplicate method is to find similar tuples from different databases, and determine schema matchings based on the similarity of values in those similar tuples [16].

In our problem, the objective is to find correspondences between different types of database elements. This requires a uniform representation of all database elements. We define a *schema tuple* as a uniform representation of both schema information and data values in a tuple.

**Definition 4.2.1** (schema tuple)**.** Given a relation $A$ and a tuple $t$, a schema tuple $\hat{t}$ is a set such that $\hat{t} = \{A\} \cup \{a_i | a_i \in A, \text{ for } i = 1, \ldots, |A|\} \cup t$, where $A$ is used interchangeably to represent the name of relation $A$ and the set of all attributes in $A$, and $a_i$ is an attribute of $A$.

For example, the schema tuple of the first tuple of database D is a set $\{\text{Stock}, \text{date}, \text{stkCode}, \text{clsPrice}, 1/2/13, \text{AAPL}, 549\}$.

Given the schema tuples from two databases, any general-purpose record linkage algorithms can be used to find similar pairs. The implementation details of this work are described in Section 4.2.3. Given a parameter $N$, the output of the record linkage algorithms are $N$ pairs of schema tuples ranked by similarity in descending order.

The next subproblem is to measure the confidence of every correspondence given the $N$ duplicate pairs. For each correspondence, the idea is to

compute a score to measure how likely a duplicate pair satisfy the correspondence based on the similarity measure $S_{field}$. The confidence measure of the correspondence is the average of the scores across all duplicate pairs. All correspondences above a confidence threshold are considered as predicted correspondences.

Given two relations $A$ and $B$, denote the attributes $a \in A$, and $b, b_1, b_2 \in B$. The tuple of $A$ in the $i$th duplicate pairs is denoted $I_i$, and the tuple of $B$ in the $i$th duplicate pairs is denoted $J_i$. The confidence of correspondence $\sigma$ is represented as $p(\sigma)$.

**Attribute-attribute correspondence** $(a, b)$

$$p((a, b)) = \frac{\sum_i S_{field}(I_i^a, J_i^b)}{N} \tag{4.1}$$

**Attribute-value correspondence** $((a, \overline{b_1}), (\overline{a}, \overline{b_2}))$ The attribute-value correspondences involve data values. Thus, we need to distinguish individual data values. The $N$ duplicate pairs are grouped by the unique data values of attribute $b_1$. A score is computed for each group, and the maximum score over all groups is the confidence of the correspondence.

$$p(((a, \overline{b_1}), (\overline{a}, \overline{b_2}))) = \max_{v \in J^{b_1}} \frac{\sum_{i, J_i^{b_1} = v} \sqrt{S_{field}(a, v) \cdot S_{field}(I_i^a, J_i^{b_2})}}{\sum_{i, J_i^{b_1} = v} 1} \tag{4.2}$$

where $J^{b_1}$ represents the set of unique values of attribute $b_1$ in all duplicate tuples.

**Relation-attribute correspondence** $((A, b), (\overline{a}, \overline{b}))$

$$p(((A, b), (\overline{a}, \overline{b}))) = \frac{\sum_i \sqrt{S_{field}(A, b) \cdot S_{field}(I_i^a, J_i^b)}}{N} \tag{4.3}$$

**Relation-value correspondence** $((A, \overline{b_1}), (\overline{a}, \overline{b_2}))$ The relation-value correspondences also need to distinguish individual data values.

$$p(((A, \overline{b_1}), (\overline{a}, \overline{b_2}))) = \max_{v \in J^{b_1}} \frac{\sum_{i, J_i^{b_1} = v} \sqrt{S_{field}(A, v) \cdot S_{field}(I_i^a, J_i^{b_2})}}{\sum_{i, J_i^{b_1} = v} 1} \tag{4.4}$$

### 4.2.2 Non-duplicate Method

The non-duplicate method is a general-purpose solution, which does not require duplicate data. As long as the two databases have data, the non-duplicate method can be used. As in the duplicate method, the task is to measure the confidence of every correspondence. All correspondences with confidence above a threshold are considered as predicted correspondences.

Instead of computing the average similarity of all duplicate pairs in the duplicate method, the non-duplicate method estimates the confidence using the similarity measure between lists of data values, $S_{list}$.

The list of all values of attribute $a$ in relation $A$ is represented as $I^a$, and the list of all values of attribute $b$ in relation $B$ is represented as $J^b$.

**Attribute-attribute correspondence** $(a, b)$

$$p((a, b)) = S_{list}(I^a, J^b) \tag{4.5}$$

**Attribute-value correspondence** $((a, \overline{b_1}), (\overline{a}, \overline{b_2}))$ Similar to the duplicate method, the tuples are grouped by the unique values of attribute $b_1$. A score is computed for each group, and the maximum score is the confidence of the correspondence.

$$p(((a, \overline{b_1}), (\overline{a}, \overline{b_2}))) = \max_{v \in J^{b_1}} \sqrt{S_{field}(a, v) \cdot S_{list}(I^a, J_v^{b_2})} \tag{4.6}$$

where $J_v^{b_2}$ represents the list of all values of attribute $b_2$ in the tuples having the value of attribute $b_1$ as $v$.

**Relation-attribute correspondence** $((A, b), (\overline{a}, \overline{b}))$

$$p(((A, b), (\overline{a}, \overline{b}))) = \sqrt{S_{field}(A, b) \cdot S_{list}(I^a, J^b)} \tag{4.7}$$

**Relation-value correspondence** $((A, \overline{b_1}), (\overline{a}, \overline{b_2}))$ The tuples are grouped by the unique values of attribute $b_1$, and the confidence is computed as the maximum score of all groups.

$$p(((A, \overline{b_1}), (\overline{a}, \overline{b_2}))) = \max_{v \in J^{b_1}} \sqrt{S_{field}(A, v) \cdot S_{list}(I^a, J_v^{b_2})} \tag{4.8}$$

### 4.2.3 Implementation

**Similarity measure** The field similarity, $S_{field}$, measures the similarity between two database elements. Any similarity measure can be used. To be comparable with the baseline, we use a similar measure as DUMAS. The similarity measure is SoftTF, which is a variation of SoftTFIDF by dropping the

IDF (IDF is not well defined in our problem) [21]. This soft measure takes into account similar terms, while conventional TF only considers equal terms.

The list similarity, $S_{list}$, measures the similarity between two lists of database elements. One choice is to cast the similarity as a supervised machine learning problem. This is not applicable to our problem due to lack of human labels. A second choice is to directly measure the intersection between the two lists. This may not perform well if there is no exact duplicate in the lists. We take a third choice, which uses statistics of the two lists to measure similarity. Statistics are summarizations of the lists, and not sensitive to individual values.

As demonstrated in iMap, different data types should have different similarity measures [26]. Thus, our measure of list similarity is specialized for different data types. The current implementation details three data types: string, number, and date. Given a list, all elements are attempted to be parsed to the three types. The type with the most successfully parsed elements is the estimated data type for the list. The statistics are extracted as a vector from all elements that can be parsed as the estimated data type. For string, the vector contains term frequencies (TF) after tokenizing all elements. For number, a twenty-bin histogram is generated to represent the distribution of all elements. Before generating the histogram, outliers are eliminated by sorting the elements and keeping the middle 95% data. For date, strings such as "January" and "Monday" are converted to numbers, and the vector contains the frequencies of numbers with one to four digits. Note that if an element has more than three numbers (a date at most contains year, month, and day)

or less than two numbers (should be a number instead of date) or more than one number with three or four digits (only year has three or four digits), that element is not parsed as a date. The similarity between two lists $l_1$ and $l_2$ that has the same estimated data type is computed as follows:

$$S_{list}(l_1, l_2) = \sqrt{S_{type}(l_1, l_2) \cdot S_{stat}(l_1, l_2)} \tag{4.9}$$

where $S_{type}(l_1, l_2)$ is the minimum of percentage of the elements that are parsed as the estimated data type in $l_1$ and $l_2$. $S_{stat}(l_1, l_2)$ is cosine similarity between the two vectors of statistics.

**Record linkage** Record linkage is the task of matching records that refer to the same entity across different data sources [20, 53]. It is also called duplicate detection if the task is within only one data source [33]. Any record linkage algorithm can be used in our duplicate methods. We use an algorithm similar to Bike and Naumann [16]. The algorithm is distinguished in the way that it can handle databases with different schema.

Given two relations $A$ and $B$, the algorithm computes similarity between each pair of schema tuples. Schema tuples are tokenized into a set of terms. TFIDF is used for weighting terms. Specifically, the weight of a term $\tau$ in a schema tuple $\hat{t}$ is computed as:

$$w'(\hat{t}, \tau) = \log(tf_{\hat{t},\tau} + 1) \cdot \log(\frac{n}{df_\tau} + 1)$$

where $tf_{\hat{t},\tau}$ is the term frequency of $\tau$ in $\hat{t}$, $df_\tau$ is the number of schema tuples in which $\tau$ appears, and $n$ is the total number of schema tuples. These weights

are normalized for each schema tuple as:

$$w(\hat{t}, \tau) = \frac{w'(\hat{t}, \tau)}{\sqrt{\sum_{\tau' \in \hat{t}} w'(\hat{t}, \tau')^2}}$$

where $\tau \in \hat{t}$ means $\tau$ is a term of $\hat{t}$ after tokenization. The similarity between two schema tuples $\hat{t}_1$ and $\hat{t}_2$ are computed as:

$$S_{tuple}(\hat{t}_1, \hat{t}_2) = \sum_{\tau \in \hat{t}_1 \cap \hat{t}_2} w(\hat{t}_1, \tau) \cdot w(\hat{t}_2, \tau)$$

These pairs of schema tuples are ranked by the similarity in descending order. Given a parameter $N$, the top $N$ pairs of schema tuples are predicted as duplicates.

When predicting attribute-value and relation-value correspondences, the value $v$ in (4.2) and (4.4) are required to occur in at least two, and at least 10% duplicate pairs to eliminate outliers. The value $v$ in (4.6) and (4.8) are required to occur in at least 10% tuples. In addition, the value $v$ is unlikely to occur in the attributes with many unique values. We prune the attributes with more than ten unique values to improve speed. If a predicted attribute-attribute correspondence has lower confidence than its corresponding compound correspondences, the attribute-attribute correspondence will be ignored.

## 4.3 Tuple-Generating Dependency

In this section, we introduce the algorithm to formulate compound correspondences as TGDs with constants. We also discuss how to use these

TGDs in data exchange and data integration applications.

### 4.3.1 Formulation

Given a target schema $\mathcal{A}$ and a source schema $\mathcal{B}$, a TGD is a first-order logic formula in the form:

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$$

where $\phi(\mathbf{x})$ is a conjunction of atomic formulas over $\mathcal{B}$, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathcal{A}$.

With the conventional schema matchings that only contain attribute correspondences, a TGD assigns variables to the attributes of $\mathcal{A}$ and $\mathcal{B}$. The matched attributes share the same variable.

Compound correspondences defined in this work involve matchings of relations, attributes, and data values. Representations of compound correspondences need higher-order logic as shown by Krishnamurthy, Litwin, and Kent, which cannot be expressed as TGDs [54]. We observe that relations and attributes in correspondences can be considered as constants. This is because if relations or attributes in databases are changed, the correspondences should be recomputed. Thus, if the correspondences are in use, the relations and attributes within the correspondences can be considered as constants. Instead of representing compound correspondences as higher-order logic, we represent them in first-order logic (TGDs) by substituting relations and attributes with their names as constants.

For example, $((\text{C:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{C:IBM}}, \overline{\text{D:clsPrice}}))$ can be represented in TGD as:

$$\forall x_1\ x_2\ x_3\ x_4\ (\text{C:Stock}(x_1, x_2, x_3, x_4) \rightarrow \exists y\ \text{D:Stock}(y, \text{“IBM”}, x_3))$$

where “IBM” is a constant.

### 4.3.2 Generation

For simplicity, we generate TGDs from one relation of the target schema $\mathcal{A}$ to one relation of the source schema $\mathcal{B}$. The generation of TGDs including more than one relation is discussed later. Given relations $A \in \mathcal{A}$, $B \in \mathcal{B}$, and a set of correspondences $\Sigma$ between $A$ and $B$, the task is to formulate $\Sigma$ as a set of TGDs.

The preliminary step of formulating TGDs is to group the correspondences in $\Sigma$ into subsets such that the correspondences in each subset are non-overlaping. A TGD will be generated for each subset. This is important for compound correspondences, since multiple correspondences may share elements. The correspondences $((\text{C:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{C:IBM}}, \overline{\text{D:clsPrice}}))$ and $((\text{C:AAPL}, \overline{\text{D:stkCode}}), (\overline{\text{C:AAPL}}, \overline{\text{D:clsPrice}}))$ are examples.

We first define the overlap of two correspondences. Correspondences can contain relations, attributes, and symbols representing data values. The overlap includes the common elements in the two correspondences. In addition, we are also concerned with the attributes underlying the symbols representing data values.

**Definition 4.3.1** (non-overlap correspondence)**.** Given two correspondences $\sigma_1$ and $\sigma_2$, $\Theta_1$ and $\Theta_2$ are the sets containing all database elements in $\sigma_1$ and $\sigma_2$, respectively. $\sigma_1$ and $\sigma_2$ are non-overlaping, if:

- $\Theta_1 \cap \Theta_2 = \emptyset$
- Denote $\{a_0, \ldots, a_i\}$ and $\{\overline{b_0}, \ldots, \overline{b_j}\}$ as the set of attributes and the set of symbols representing data values in $\Theta_1$, and $\{c_0, \ldots, c_k\}$ and $\{\overline{d_0}, \ldots, \overline{d_l}\}$ as the set of attributes and the set of symbols representing data values in $\Theta_2$. $\{a_0, \ldots, a_i\} \cap \{d_0, \ldots, d_l\} = \emptyset$ and $\{b_0, \ldots, b_j\} \cap \{c_0, \ldots, c_k\} = \emptyset$.

**Definition 4.3.2** (non-overlap correspondence set)**.** A set of correspondences $\Sigma$ is non-overlaping, if $\forall\ \sigma_1, \sigma_2 \in \Sigma$ that $\sigma_1 \neq \sigma_2$, $\sigma_1$ and $\sigma_2$ are non-overlaping.

Given a set of correspondences $\Sigma$, a *minimum non-overlap support* $\Sigma^s$ is defined as a set of subsets of $\Sigma$, each of which is a non-overlap correspondence set and is as large as possible.

**Definition 4.3.3** (minimum non-overlap support)**.** Given a set of correspondences $\Sigma$, a set $\Sigma^s$ is a minimum non-overlap support of $\Sigma$ if the following conditions are satisfied:

- $\forall\ \Delta \in \Sigma^s$, $\Delta \subseteq \Sigma$;
- $\forall\ \Delta \in \Sigma^s$, $\Delta$ is a non-overlap correspondence set;
- $\forall \Delta' \subseteq \Sigma$ that is non-overlap, $\exists \Delta \in \Sigma^s$ such that $\Delta' \subseteq \Delta$;

**Algorithm 1** Formulate correspondences in TGDs

**Input:** target relation $A$, source relation $B$, a minimum non-overlap support $\Sigma^s$

**Output:** a set of TGDs $\Omega$

```
for Δ ∈ Σ^s do
    // M is a map from an attribute to a variable or constant
    M = ∅
    Assign a unique variable to each attribute of A and B to M
    for σ ∈ Δ do
        M = SubstituteVariables(A, B, σ, M)
    end for
    // Generate TGD based on the map M
    ω = "∀"
    Add the variables of source attributes to ω
    Add B and the values of all attributes of B to ω
    Add "→ ∃" to ω
    Add the variables of target attributes that are not shared with source attributes to ω
    Add A and the values of all attributes of A to ω
    Add ω to Ω
end for
return Ω
```

- $\forall\ \Delta_1, \Delta_2 \in \Sigma^s$, $\Delta_1 \neq \Delta_2$, then $\Delta_1 \not\subset \Delta_2$ and $\Delta_2 \not\subset \Delta_1$.

Given the fact that the number of correspondences is usually small, the minimum non-overlap support can be found by brute force.

Algorithm 1 and 2 describe the details of generating TGDs given a minimum non-overlap support. The algorithms maintain a map data structure with all attributes of both $A$ and $B$ as keys, and strings as values. The map is initialized by assigning a unique variable to each attribute. The algorithms visit each correspondence, and substitute some variables in the map

**Algorithm 2** The SubstituteVariables function in Algorithm 1

**Input:** target relation $A$, source relation $B$, correspondence $\sigma$, map $M$
**Output:** $M$

```
function SubstituteVariables(A, B, σ, M)
    Let Θ_A be the database element set of A
    // ē represents the data values of attribute e
    if σ is attribute-attribute correspondence (e_3, e_4) or relation-attribute
correspondence ((e_1, e_2), (ē_3, ē_4)) then
        // Substitute target variables with source variables
        if e_3 ∈ Θ_A then
            M(e_3) = M(e_4)
        else
            M(e_4) = M(e_3)
        end if
    else
        // σ is attribute-value or relation-value ((e_1, ē_2), (ē_3, ē_4))
        // Substitute target variables with source variables
        if e_3 ∈ Θ_A then
            M(e_3) = M(e_4)
        else
            M(e_4) = M(e_3)
        end if
        // Substitute variables with constants
        M(e_2) = e_1
    end if
return M
end function
```

with other variables or constants according to the correspondence. Finally, a logic representation is generated based on the map. The variable substitutions for a relation-attribute correspondence $((e_1, e_2), (\overline{e_3}, \overline{e_4}))$ and a corresponding attribute-attribute correspondence $(e_3, e_4)$ are the same. This is due to the fact that the primitive correspondence $(e_1, e_2)$ is between a relation and an

attribute. Both relation and attribute are constants, so there is no variable to substitute. This fact shows that a relation-attribute correspondence does not provide more information than its corresponding attribute-attribute correspondence in a TGD. For attribute-value and relation-value correspondences, the substitution involves constants. Both variables of the target attributes and the source attributes may be substituted with constants.

In the examples in Figure 4.3, let databases D and C be the target and source databases, respectively. The set of correspondences $\Sigma$ is:

$$\begin{aligned}\Sigma = \{\ & (\text{C:date}, \text{D:date}),\\ & ((\text{C:AAPL}, \overline{\text{D:stkCode}}), (\overline{\text{C:AAPL}}, \overline{\text{D:clsPrice}})),\\ & ((\text{C:IBM}, \overline{\text{D:stkCode}}), (\overline{\text{C:IBM}}, \overline{\text{D:clsPrice}}))\ \}\end{aligned}$$

The set of generated TGDs is:

$$\begin{aligned}\Omega = \{\ & \forall x_1\ x_2\ x_3\ x_4\ (\text{C:Stock}(x_1, x_2, x_3, x_4) \rightarrow \text{D:Stock}(x_1, \text{“AAPL”}, x_2)),\\ & \forall x_1\ x_2\ x_3\ x_4\ (\text{C:Stock}(x_1, x_2, x_3, x_4) \rightarrow \text{D:Stock}(x_1, \text{“IBM”}, x_3))\ \}\end{aligned} \tag{4.10}$$

**TGDs with multiple relations** The problem of generating schema mappings with multiple relations has been discussed in Clio, which semi-automatically generates mappings as TGDs [37]. The mappings are created in two steps: finding initial correspondences between attributes and associating correspondences by logical inference through referential constraints. The correspondence association is done by a modification of the chase algorithm [61].

To generate schema mappings with multiple relations in our problem, we can directly apply the method of correspondence association in Clio to the TGDs between a pair of relations generated by Algorithm 1.

### 4.3.3 Applications

**Data Exchange** Data exchange is the problem of transforming an instance of a source schema to an instance of the target schema, given a mapping between the two schemas [38, 9]. Let us denote the target schema as $\mathcal{A}$ and the source schema as $\mathcal{B}$. A mapping, $\mathcal{M}$, is defined as a triple $(\mathcal{A}, \mathcal{B}, \Omega)$, where $\Omega$ is a set of *source-to-target dependencies*. In this definition, we ignore the constraints on the target schema, and only consider the source-to-target dependencies. The source-to-target dependencies are usually TGDs.

A target instance $I$ is a *solution* of a source instance $J$ under mapping $\mathcal{M}$, if $I$ and $J$ satisfy $\Omega$. There may exist more than one solution of a source instance. A *universal solution* is a target instance that contains no more and no less than what the mapping specification requires. As it is widely accepted in the data exchange literature, universal solutions reflect the source data better [9].

Generating universal solutions is an important task in data exchange. The chase algorithm is generally used to find universal solutions [61, 38]. Given a source instance, the chase algorithm iteratively visits a TGD in the schema mapping, and generates a target tuple. The process is stopped if no new target tuples can be generated.

**Algorithm 3** Generate universal solutions for data exchange

**Input:** mapping $(\mathcal{A}, \mathcal{B}, \Omega)$, instance $J$ of $\mathcal{B}$
**Output:** instance $I$ of $\mathcal{A}$

```
I = ∅
while new tuples can be added to I do
    Let ω ∈ Ω be ∀x(φ(x) → ∃yψ(x, y))
    Let α be a mapping from x to the constants in J such that φ(α(x)) ∈ J
    // α exist if the constants in J and φ(α(x)) are the same
    if α exists then
        Let β be a mapping from y to the variables in I such that
ψ(α(x), β(y)) ∈ I
        if β does not exist then
            Let γ be a mapping that maps each variable in y to a new variable
that is not in I
            Add ψ(α(x), γ(y)) to I
        end if
    end if
end while
return I
```

In our problem, TGDs include constants to represent compound correspondences. We extend the chase algorithm to generate universal solutions for our problem. Algorithm 3 is a slightly modified chase algorithm based on the version in [29]. The only change in Algorithm 3 is the check of whether the constants in the source instance and the TGDs are the same.

Given the TGDs in (4.10) and the instance of database C in Figure 4.3,

the universal solutions are:

$$(1/2/13, \text{AAPL}, 549)$$

$$(1/2/13, \text{IBM}, 196)$$

$$(1/3/13, \text{AAPL}, 542)$$

$$(1/3/13, \text{IBM}, 195)$$

**Data Integration** Data integration provides uniform access to a set of autonomous, heterogeneous structured data sources [29]. Given a target schema $\mathcal{A}$, a set of source schemas $\mathcal{B}_1, \mathcal{B}_2, \ldots$, schema mappings $\Omega$, and a user query $q_{\mathcal{A}}$ over $\mathcal{A}$, the task of data integration is to rewrite $q_{\mathcal{A}}$ to queries over the source schemas and answer $q_{\mathcal{A}}$ using the data in the data sources. The schema mappings can be represented in three formalisms. Global-as-view (GAV) represents each relation in the target schema as a view of the source schemas. Local-as-view (LAV) represents each relation in the source schemas as a view of the target schema. The combination of GAV and LAV is Global-local-as-view (GLAV), which includes a mapping from a view of the target schema to a view of the source schemas.

Query rewriting is an important task in data integration. The rewriting in GAV formalism is simply query unfolding. The rewriting in LAV formalism is more complex. It needs algorithms for answering queries using views, such as MiniCon [73]. The rewriting in GLAV formalism combines the rewriting in both GAV and LAV. The algorithm contains two steps: (1) rewrite the

target query using the views of the target schema in the schema mappings, (2) replace the views of the target schema by the views of the source schema and unfold the queries.

In our problem, the schema mappings are represented as TGDs, which are equivalent in expressiveness to GLAV formalism. Thus, the rewriting algorithms for GLAV can be applied to TGDs. MiniCon is a well-known rewriting algorithm that translates a conjunctive query to union of conjunctive queries [73]. It supports constants in both queries and mappings. Thus, the MiniCon algorithm can be directly applied to the TGDs in our solution.

In the examples in Figure 4.3, let databases D and C be the target and source databases, respectively. The schema mappings are given in (4.10). Consider a query $Q(x, y)$ over database D asking the prices of all stocks of date "1/2/13 ". For clarity, queries are written in relational calculus.

$$Q(x, y) \text{ :- D:Stock}(1/2/13, x, y)$$

The first step of the rewriting represents the query using union of two conjunctive views of database D in the TGDs:

$$Q'(x, y) \text{ :- D:Stock}(1/2/13,\text{AAPL}, y),\ x = \text{AAPL}$$
$$Q'(x, y) \text{ :- D:Stock}(1/2/13,\text{IBM}, y),\ x = \text{IBM}$$

The final rewritten query is union of two conjunctive views of the source schema

C:

$$Q''(x, y) \text{ :- C:Stock}(1/2/13, y, z_1, z_2),\ x = \text{AAPL}$$

$$Q''(x, y) \text{ :- C:Stock}(1/2/13, z_3, y, z_4),\ x = \text{IBM}$$

Comparing $Q(x, y)$ and $Q''(x, y)$, the target query is a conjunctive query, but the rewritten query is a union of two conjunctive queries. Recall that the schema mappings between database C and D include attribute-value correspondences. Although we represent the attribute-value correspondences in first-order logic, the semantics still need second-order logic. Thus, the rewritten query needs to enumerate (union) the attributes to express the second-order logic by first-order logic.

## 4.4 Experiments

### 4.4.1 Test Sets

The test sets are detailed in Section 3.1. There are four application domains: Ecommerce, Stock Market, College Enrollment, and Video Game. Table 4.1 shows the number of unique groundtruth correspondences of different types. Compound correspondences occur in all test sets, which demonstrates their importance.

| | Ecommerce | Stock | Enrollment | Game |
|---|---|---|---|---|
| attribute-attribute | 1 | 2 | 2 | 1 |
| attribute-value | 5 | 3 | 3 | 3 |
| relation-attribute | 0 | 1 | 1 | 0 |
| relation-value | 0 | 2 | 4 | 0 |
| total | 6 | 8 | 10 | 4 |

Table 4.1: The number of unique groundtruth correspondences of different types.

### 4.4.2 Baselines

DUMAS is the schema matching system that is closely related to our proposed duplicate method [16]. We implement DUMAS baselines as follows. First, the record linkage algorithm in Section 4.2.3 is applied to generate duplicate tuples, instead of schema tuples. A similarity matrix of all pairs of attributes is computed as in (4.1). All pairs with similarity above the threshold are the predicted correspondences. These baselines are named as DUMAS_x, where x is the number of duplicates used for matching. In the experiments, x is set to 5 and 10 as in our proposed methods.

We also implement a general-purpose instance-based matching baseline to compare with our proposed non-duplicate method. The instance-based matching baseline extracts the data of each attribute as a set, and compute a similarity matrix of all pairs of attributes as in (4.5). All pairs with similarity above the threshold are the predicted correspondences. The instance-based baseline is named Instance. Instance uses at most 1000 tuples from each schema for matching.

The proposed duplicate methods are Dup_5 and Dup_10, which use 5 and 10 duplicates, respectively. The proposed non-duplicate method is NonDup. NonDup uses the same tuples as Instance baseline. The main difference between the proposed methods and the baselines is that the baselines only predict attribute-attribute correspondences.

### 4.4.3 Metrics

We measure the precision, recall, and f-measure of correspondences. Denote the set of predicted correspondences as $P$, and the set of groundtruth correspondences as $T$. The precision (corr_p), recall (corr_r), and f-measure (corr_f) are defined as follows:

$$\text{corr\_p}(P,T) = \frac{|P| \cap |T|}{|P|}, \quad \text{corr\_r}(P,T) = \frac{|P| \cap |T|}{|T|}$$
$$\text{corr\_f}(P,T) = 2 \cdot \frac{\text{corr\_p}(P,T) \cdot \text{corr\_r}(P,T)}{\text{corr\_p}(P,T) + \text{corr\_r}(P,T)}$$

Per Section 4.3, a relation-attribute correspondence and its corresponding attribute-attribute correspondence are formulated to the same TGD. Thus, we do not distinguish a relation-attribute correspondence and its corresponding attribute-attribute correspondence in the evaluation.

Generating TGDs based on correspondences is a deterministic process. Thus we do not measure the correctness of TGDs in the experiments.

If a data set has more than two schemas, the experiments are conducted on each pair of schemas, and the average results are reported.

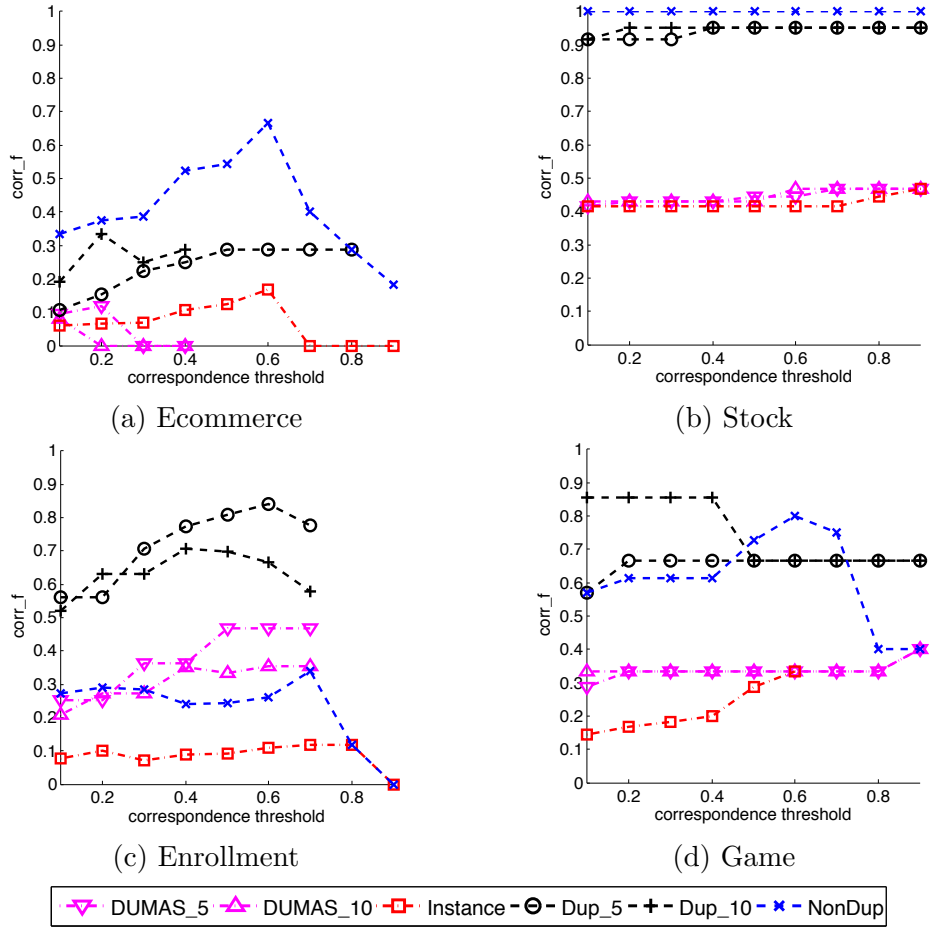
### 4.4.4 Results

The threshold to determine predicted correspondences has a large impact to all methods (predicted correspondences must have higher confidence than the threshold). We vary the threshold from 0.1 to 0.9 to give a comprehensive comparison.

A fair comparison should consider both precision and recall at the same time. Figure 4.5 shows the correspondence f-measures given various thresholds. F-measure is a tradeoff between precision and recall, and higher f-measure indicates better performance. Figure 4.6, 4.7, 4.8, and 4.9 show both precision and recall given different correspondence thresholds.

In summary, at least one of the proposed methods dominates all baselines in all four data sets. Comparing the duplicate methods with the non-duplicate methods, the duplicate methods usually achieve higher precision while the non-duplicate methods achieve higher recall. Their overall performance depends on individual data set.

The Ecommerce data set is challenging for all methods. First, the data of the two schemas are completely from independent data sources. There is no duplicate data. Second, one of the relations is large, containing 39 attributes. The data values of those attributes span many data types, including integer, real number, date, binary, url, and string. As shown in Figure 4.5, every proposed method performs better than all baselines. The top proposed method (NonDup with 0.6 threshold) has 0.667 of f-measure, while the top baseline




(a) Ecommerce

(b) Stock

(c) Enrollment

(d) Game

(e) Legend

Figure 4.5: Correspondence f-measures (corr_f, vertical axis) with various correspondence thresholds (horizontal axis). Some points in the figures may be missing due to the fact that no predicted correspondence has confidence higher than the threshold. Higher value means better performance.

(Instance with 0.6 threshold) only has 0.167. NonDup dominates both Dup_5 and Dup_10 too. The NonDup achieves higher f-measure because of higher recall as shown in Figure 4.6.

| | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|---|
| DUMAS_5 | corr_p | .07 | .09 | .00 | .00 | - | - | - | - | - |
| | corr_r | .17 | .17 | .00 | .00 | - | - | - | - | - |
| DUMAS_10 | corr_p | .05 | .00 | .00 | .00 | - | - | - | - | - |
| | corr_r | .17 | .00 | .00 | .00 | - | - | - | - | - |
| Instance | corr_p | .04 | .04 | .04 | .08 | .10 | .17 | .00 | .00 | .00 |
| | corr_r | .17 | .17 | .17 | .17 | .17 | .17 | .00 | .00 | .00 |
| Dup_5 | corr_p | .08 | .14 | .33 | .50 | 1.0 | 1.0 | 1.0 | 1.0 | - |
| | corr_r | .17 | .17 | .17 | .17 | .17 | .17 | .17 | .17 | - |
| Dup_10 | corr_p | .13 | .33 | .50 | 1.0 | - | - | - | - | - |
| | corr_r | .33 | .33 | .17 | .17 | - | - | - | - | - |
| NonDup | corr_p | .20 | .23 | .24 | .35 | .38 | .50 | .33 | .25 | .20 |
| | corr_r | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | .50 | .33 | .17 |

Figure 4.6: Precision (corr_p) and recall (corr_r) of the Ecommerce dataset obtained by varying correspondence threshold (columns) from 0.1 to 0.9. Some cells have "-" due to the fact that no predicted correspondence has confidence higher than the threshold.

| | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|---|
| DUMAS_5 | corr_p | .42 | .44 | .44 | .44 | .50 | .50 | .67 | .67 | .67 |
| | corr_r | .42 | .42 | .42 | .42 | .42 | .42 | .42 | .42 | .42 |
| DUMAS_10 | corr_p | .44 | .44 | .44 | .44 | .44 | .67 | .67 | .67 | .67 |
| | corr_r | .42 | .42 | .42 | .42 | .42 | .42 | .42 | .42 | .42 |
| Instance | corr_p | .42 | .42 | .42 | .42 | .42 | .42 | .42 | .50 | .67 |
| | corr_r | .42 | .42 | .42 | .42 | .42 | .42 | .42 | .42 | .42 |
| Dup_5 | corr_p | .92 | .92 | .92 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | corr_r | .92 | .92 | .92 | .92 | .92 | .92 | .92 | .92 | .92 |
| Dup_10 | corr_p | .92 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | corr_r | .92 | .92 | .92 | .92 | .92 | .92 | .92 | .92 | .92 |
| NonDup | corr_p | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | corr_r | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

Figure 4.7: Precision (corr_p) and recall (corr_r) of the Stock dataset. See Figure 4.6 for explanations.

| | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|---|
| DUMAS_5 | corr_p | .18 | .18 | .33 | .33 | .67 | .67 | .67 | - | - |
| | corr_r | .42 | .42 | .42 | .42 | .42 | .42 | .42 | - | - |
| DUMAS_10 | corr_p | .14 | .22 | .22 | .31 | .50 | .67 | .67 | - | - |
| | corr_r | .42 | .42 | .42 | .42 | .25 | .25 | .25 | - | - |
| Instance | corr_p | .04 | .06 | .04 | .05 | .06 | .07 | .08 | .08 | .00 |
| | corr_r | .42 | .42 | .25 | .25 | .25 | .25 | .25 | .25 | .00 |
| Dup_5 | corr_p | .41 | .41 | .61 | .72 | .80 | .83 | .83 | - | - |
| | corr_r | .92 | .92 | .92 | .92 | .92 | .92 | .83 | - | - |
| Dup_10 | corr_p | .37 | .52 | .52 | .61 | .80 | .83 | .83 | - | - |
| | corr_r | .92 | .92 | .92 | .92 | .75 | .67 | .58 | - | - |
| NonDup | corr_p | .17 | .18 | .18 | .16 | .16 | .17 | .25 | .08 | .00 |
| | corr_r | 1.0 | 1.0 | .83 | .58 | .58 | .58 | .58 | .25 | .00 |

Figure 4.8: Precision (corr_p) and recall (corr_r) of the Enrollment dataset. See Figure 4.6 for explanations.

| | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|---|
| DUMAS_5 | corr_p | .33 | .50 | .50 | .50 | .50 | .50 | .50 | .50 | 1.0 |
| | corr_r | .25 | .25 | .25 | .25 | .25 | .25 | .25 | .25 | .25 |
| DUMAS_10 | corr_p | .50 | .50 | .50 | .50 | .50 | .50 | .50 | .50 | 1.0 |
| | corr_r | .25 | .25 | .25 | .25 | .25 | .25 | .25 | .25 | .25 |
| Instance | corr_p | .10 | .13 | .14 | .17 | .33 | .50 | - | - | - |
| | corr_r | .25 | .25 | .25 | .25 | .25 | .25 | - | - | - |
| Dup_5 | corr_p | .67 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | corr_r | .50 | .50 | .50 | .50 | .50 | .50 | .50 | .50 | .50 |
| Dup_10 | corr_p | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | corr_r | .75 | .75 | .75 | .75 | .50 | .50 | .50 | .50 | .50 |
| NonDup | corr_p | .40 | .44 | .44 | .44 | .57 | .67 | .75 | 1.0 | 1.0 |
| | corr_r | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | .75 | .25 | .25 |

Figure 4.9: Precision (corr_p) and recall (corr_r) of the Game dataset. See Figure 4.6 for explanations.

The data of all schemas in the Stock data set come from the same source, so the comparison is mainly to highlight the importance of predicting

all types of correspondences instead of only attribute-attribute. All proposed methods achieve f-measure above 0.9, while all baselines are under 0.5. The big gaps between the proposed methods and the baselines demonstrate the importance of predicting all types of correspondences.

For the Enrollment data set, Dup_5 and Dup_10 dominate all baselines. NonDup performs better than or as well as Instance baseline, but worse than DUMAS_5 and DUMAS_10. This is because most of the enrollment data are numbers of enrollment for different colleges in different years. The numbers are too similar to be distinguished by the non-duplicate methods. Dup_5 performs better than Dup_10 on the Enrollment data set. This is because the number of real duplicates in the data set is small. Thus, more duplicates introduce more errors to Dup_10.

For the Game data set, all proposed methods perform better than or as well as the top baseline. The Dup_5 and Dup_10 still dominate all baselines with big gaps. The NonDup method achieves the highest f-measure with threshold 0.5 to 0.7, and the duplicate methods perform better for other thresholds.

The performance of the proposed duplicate methods is stable and robust on the Stock, Enrollment, and Game data sets, which contain some duplicates. Even on Enrollment data set with most of the data as numbers, the duplicate methods still achieve the highest performance. The drawback of these methods is the requirement of duplicate data from two schemas. However, our experimental results demonstrate that the number of required duplicate data

is small. The top method only needs 5 duplicates. This small requirement enables duplicate methods to be used in broad applications.

The non-duplicate method is general for all schemas with data. On the Ecommerce, Stock and Game sets, NonDup performs better than the duplicate methods given certain correspondence thresholds. The performance is highly dependent on the capabilities of the underlying similarity measures. On Enrollment data set, NonDup is dominated by Dup_5 and Dup_10 with large gaps, because the similarity measures are confused by the numeric data.

The proposed duplicate methods usually achieve higher precision, while the proposed non-duplicate method achieves higher recall. This is illustrated by the Ecommerce and Game data sets in Figure 4.6 and 4.9. For the Ecommerce data set, the duplicate methods can have as high as 1.0 of precision, but lower than 0.4 of recall. The non-duplicate method can have 1.0 of recall, but lower than 0.5 of precision. This indicates that the duplicate and non-duplicate methods should be used in applications with different requirements. The duplicate methods give high accuracy, while the non-duplicate method gives high coverage.

Correspondence threshold has a large impact on performance. Higher threshold gives higher precision, but lower recall. The optimal choice of threshold is dependent on data sets. For the schemas with higher data similarity, such as the Stock data set, a higher threshold should be chosen. On the contrary, a lower threshold gives better performance for the schemas with lower data similarity, such as the Game data set.

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.00 | 0.00 |
| a-v | 0.00 | 0.00 |
| r-a | - | - |
| r-v | - | - |

(a) Ecom, DUMAS_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.00 | 0.00 |
| a-v | 0.00 | 0.00 |
| r-a | - | - |
| r-v | - | - |

(b) Ecom, DUMAS_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.17 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | - | - |
| r-v | - | - |

(c) Ecom, Instance

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.00 | 0.00 |
| a-v | 1.00 | 0.20 |
| r-a | - | - |
| r-v | - | - |

(d) Ecom, Dup_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.00 | 0.00 |
| a-v | 0.00 | 0.00 |
| r-a | - | - |
| r-v | - | - |

(e) Ecom, Dup_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.20 | 1.00 |
| a-v | 0.71 | 1.00 |
| r-a | - | - |
| r-v | - | - |

(f) Ecom, NonDup

Figure 4.10: Precision (corr_p) and recall (corr_r) for each type of correspondences of the Ecommerce dataset. a-a is the typical attribute correspondence. a-v, r-a, and r-v are attribute-value, relation-attribute, and relation-value correspondences. If a correspondence type is not existed in the groundtruth, we set both precision and recall as "-" for fair comparison.

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.75 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | 1.00 | 1.00 |
| r-v | 0.00 | 0.00 |

(a) Stock, DUMAS_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | 1.00 | 1.00 |
| r-v | 0.00 | 0.00 |

(b) Stock, DUMAS_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.63 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | 1.00 | 1.00 |
| r-v | 0.00 | 0.00 |

(c) Stock, Instance

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 1.00 | 0.67 |
| r-a | 1.00 | 1.00 |
| r-v | 1.00 | 1.00 |

(d) Stock, Dup_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 1.00 | 0.67 |
| r-a | 1.00 | 1.00 |
| r-v | 1.00 | 1.00 |

(e) Stock, Dup_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 1.00 | 1.00 |
| r-a | 1.00 | 1.00 |
| r-v | 1.00 | 1.00 |

(f) Stock, NonDup

Figure 4.11: Precision (corr_p) and recall (corr_r) for each type of correspondences of the Stock dataset. See Figure 4.10 for explanations.

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | 1.00 | 1.00 |
| r-v | 0.00 | 0.00 |

(a) Enroll, DUMAS_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | 0.00 | 0.00 |
| r-v | 0.00 | 0.00 |

(b) Enroll, DUMAS_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.11 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | 0.00 | 0.00 |
| r-v | 0.00 | 0.00 |

(c) Enroll, Instance

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 1.00 | 0.67 |
| r-a | 1.00 | 1.00 |
| r-v | 1.00 | 1.00 |

(d) Enroll, Dup_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 1.00 | 0.33 |
| r-a | 0.00 | 0.00 |
| r-v | 1.00 | 1.00 |

(e) Enroll, Dup_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.11 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | 0.00 | 0.00 |
| r-v | 0.67 | 1.00 |

(f) Enroll, NonDup

Figure 4.12: Precision (corr_p) and recall (corr_r) for each type of correspondences of the Enrollment dataset. See Figure 4.10 for explanations.

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.50 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | - | - |
| r-v | - | - |

(a) Game, DUMAS_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.50 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | - | - |
| r-v | - | - |

(b) Game, DUMAS_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 0.50 | 1.00 |
| a-v | 0.00 | 0.00 |
| r-a | - | - |
| r-v | - | - |

(c) Game, Instance

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 1.00 | 0.33 |
| r-a | - | - |
| r-v | - | - |

(d) Game, Dup_5

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 1.00 | 0.33 |
| r-a | - | - |
| r-v | - | - |

(e) Game, Dup_10

| | corr_p | corr_r |
|---|---|---|
| a-a | 1.00 | 1.00 |
| a-v | 0.75 | 1.00 |
| r-a | - | - |
| r-v | - | - |

(f) Game, NonDup

Figure 4.13: Precision (corr_p) and recall (corr_r) for each type of correspondences of the Game dataset. See Figure 4.10 for explanations.

Figure 4.10, 4.11, 4.12, and 4.13 show the precision and recall of each type of correspondences. Given the limited space, we only show the results

using 0.6 as the correspondence threshold, since most methods achieve top performance using that threshold. Comparing our methods with the baselines, our methods are capable to generate all types of correspondences. The baselines can only generate attribute correspondences (baselines may get non-zero for relation-attribute correspondences, because we do not distinguish the correctness between relation-attribute correspondences and their corresponding attribute correspondences). Dup_5 has the highest precision of all types of correspondences for Stock, Enrollment, and Game. NonDup has the highest recall of all types of correspondences for Ecommerce, Stock, and Game.

### 4.4.5 Error Categorization

To detail insight into the proposed methods, we categorize the main problems observed in the experiments.

1. Attribute-value and relation-value correspondences have similar confidence to their corresponding attribute-attribute correspondences. By definition, the elements of an attribute-value correspondence $((a, \overline{b_1}), (\overline{a}, \overline{b_2}))$ form two primitive correspondences $(a, \overline{b_1})$ and $(\overline{a}, \overline{b_2})$. The elements in an attribute-attribute correspondence $(a, b_2)$ form the same primitive correspondence $(\overline{a}, \overline{b_2})$. These two types of correspondences usually have similar confidence. For example, an attribute-value correspondence may have high confidence 0.9, but the corresponding attribute-attribute correspondence may have confidence 0.8 or even 0.95. This tends to predict more correspondences, reducing precision. Relation-value correspondences have the same issue.

2. Duplicate methods cannot discover all correspondences with limited duplicate data. These methods determine correspondences from limited number of duplicate data (5 and 10 in our experiments). These limited number of duplicate data may not cover all cases of correspondences. In the Stock data set, the top duplicate data only cover company AAPL and GOOG, but no IBM. Thus, the correspondences involved IBM cannot be predicted, which reduces recall. One possible solution is to increase the number of duplicates. However, as the number of duplicates increases, there will be more errors in record linkage. A possible future work should select duplicate data by not only considering confidences, but also diversity in terms of generating correspondences.

3. The similarity measures in both duplicate and non-duplicate methods need improvement. The distinguishing capability of similarity measure is difficult to choose. For one extreme end, the similarity measure may only consider exactly matched data. For the other extreme end, the similarity measure may just consider the data types. Different distinguishing capability determines the balance between precision and recall. In our experiments, the implemented similarity measure performs well on the Stock and Game set, but poorly on the Ecommerce set for the duplicate methods and on the Enrollment set for the non-duplicate method. This indicates that different applications may need similarity measures with different capabilities.

## 4.5 Related Work

### 4.5.1 Schema Matching

Most instance-based schema matching systems extract the values of all tuples for each attribute as a set, and compute the confidence of a correspondence either by similarity measures or machine learning algorithms [28, 30]. The DUMAS system proposes another approach that first finds similar tuples, and determines schema matchings based on those tuples [16]. Although DUMAS has the limitation that the two databases need to share duplicate tuples, the experiments show that they only need a few duplicates (fewer than 10) to achieve reasonable accuracy. In schema matching literature, most systems define the matching task as finding correspondences between attributes. In this paper, we define three additional correspondence types as compound correspondences.

iMap system introduces a special attribute-value correspondence [26]. They focus on the case where target attributes have binary data values. We define the attribute-value correspondences for general cases. In addition, we also define relation-attribute and relation-value correspondences.

### 4.5.2 Record Linkage

Record linkage is the task of matching records that refer to the same entity across different data sources [20, 53]. It is also called duplicate detection if the task is within only one data source [33]. In the relational database context, the goal of record linkage is to find pairs or groups of tuples that

refer to the same real-world identity. Bilke and Naumann consider a tuple as a set of strings, and scores a pair of tuples by a similarity measure of the two sets [16]. The similarity measure is the cosine similarity between two TFIDF feature vectors.

## 4.6 Discussion and Future Work

Schema matching is conventionally defined as finding correspondences between attributes. As shown in our real world data sets of four application domains, this conventional definition has limitations. We introduce three types of compound correspondences consisting of relations, attributes, and data values. These compound correspondences complement the typical attribute correspondences. The two proposed algorithms can automatically generate these correspondences. The correspondences can be formulated to TGDs, and used in data exchange and data integration applications.

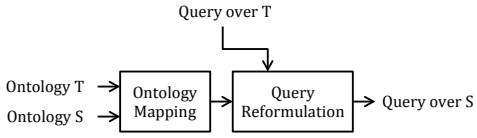
Many potential research questions are emerged following this work. First, a ranking algorithm of duplicate pairs considering diversity in addition to similarity is important to improve the recall of duplicate methods. Second, similarity measures for non-string data types is a promising direction to improve schema matching systems.

# Chapter 5

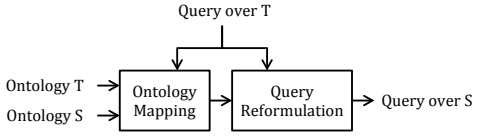
# Mapping Paths in the Semantic Web

This work details the mapping algorithms of the Query-driven Ontology-based Data Integration (QODI) [87, 88, 89]. QODI considers two OWL ontologies: the target ontology, which is the federated data model, and the source ontology. SPARQL queries are issued over the target ontology by users, and translated to queries over the source ontology. Although QODI is designed to integrate RDF data, a primary motivation is the integration of relational data. Several of our test cases comprise relational databases virtualized as RDF, and SQL schemas translated to ontologies [80, 81].

In the typical organization of an Ontology-based Data Integration system (OBDI), ontology mapping is a separate and prerequisite step of query reformulation (see Figure 5.1a). Ontology matchers may be introduced to automatically determine corresponding entities [13, 35]. In this work, an entity refers to a class or a property. We tested AgreementMaker [23], one of the top finishers in 2010 Ontology Alignment Evaluation Initiative (OAEI) [2]. The highest accuracy of AgreementMaker on our test sets is less than 42%. Inspection of these results revealed two dominant challenges: *ambiguous mapping* and *missing mapping*. We create a small example to illustrate the challenges.

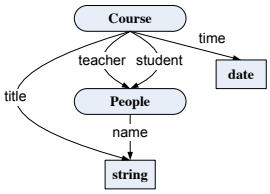



(a) Traditional

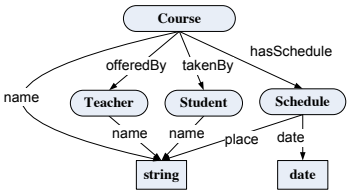



(b) The proposed, QODI

Figure 5.1: Diagram of OBDI systems with traditional and the proposed ontology mapping.




(a) Ontology $T$




(b) Ontology $S$

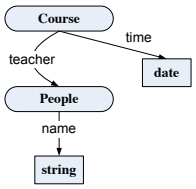
Figure 5.2: Ontology examples about the domain of course. Oval vertices represent classes, and rectangular vertices are datatypes. Edges represent object properties, or datatype properties.

Figure 5.2 shows a target ontology $T$, and a source ontology $S$ about courses. Figure 5.3 shows a SPARQL query $q$ which asks for the time of any course that is taught by Einstein.

**The ambiguous mapping challenge:** an entity in the target ontology has an ambiguous mapping if it can be mapped to more than one entity in the source ontology, and the correct choice is dependent on the application. In other words, there is not enough information in the ontologies alone to determine a correct mapping. An example of ambiguous mapping considers that *name* of class *People* in $T$ can be mapped to *name* of either class *Teacher* or *Student* in $S$. There is no basis for preferring one mapping or another. However, considering query $q$, clearly *Teacher* is preferred.

```
Prefix  course  :  < T/Course >
Prefix  people  :  < T/People >
Select  ?t
Where  {
  ?c  course : time  ?t  .
  ?c  course : teacher  ?p  .
  ?p  people : name  "Einstein"  .}
```

(a) SPARQL query




(b) Query graph

Figure 5.3: SPARQL query example and corresponding query graph. The SPARQL query asks for the time of any course taught by "Einstein".

Some matchers would identify this example as a *complex mapping* such that *name* of *People* maps to the union of both *name* of *Teacher* and *Student*, since both *Teacher* and *Student* can be identified as subclasses of *People*. In isolation of an application, the logic of the complex mapping is correct. But, if the example query is reformulated using both alternatives, the translated query will return the time of any course that either taught or taken by Einstein. The reformulation is incorrect. Thus, only after the query is known, is it possible to disambiguate the mapping.

Ambiguous mappings occur often. In our real world test sets, two out of three domains have ambiguity. In those, 10% to 30% of the query workload displays ambiguity.

**The missing mapping challenge:** some entities do not have any mapping, such as the class *Schedule* and property *hasSchedule* in $S$. Matchers can find out that both *Course* in $T$ and $S$ are mapped, and *time* and *date* are mapped. However, *Schedule* and *hasSchedule*, which are in the middle of the path from *Course* to *date*, do not have any mapping. Query $q$ cannot be

reformulated for execution on $S$ without including *Schedule* and *hasSchedule*.

We formally define *query-specific ontology mapping*. For each input query, the system determines a partial ontology mapping sufficient to reformulate the specific query. In effect, a query becomes a third argument to the ontology mapping algorithm (see Figure 5.1b). Note that using the query as context requires no extra input from users or experts. In QODI, both the input query and the source ontology are decomposed into paths, and mapping concerns identifying correspondences between paths instead of entities. Path similarity is estimated based on the feature vectors that are generated by representing each path as a bag of entity labels. Given an input query, QODI searches for a subgraph of the source ontology, such that the set of path correspondences has the highest confidence. QODI exploits heuristic search algorithms, which guarantee to find an optimal solution. By leveraging queries to provide context, the ambiguous mapping challenge is resolved. Since the path similarity is not dependent on the precise alignment of entities, the missing mapping challenge is resolved.

In our running example, the path that contains *People* and *name* in query $q$ also contains *teacher*. In ontology $S$, the path with *Teacher* has higher string vector similarity than the one with *Student*. The two path correspondences for the query should be:

$$
\begin{aligned}
&\{\text{Course,teacher,People,name,string}\} \\
=\ &\{\text{Course,offeredBy,Teacher,name,string}\} \\
&\{\text{Course,time,date}\} \\
=\ &\{\text{Course,hasSchedule,Schedule,date,date}\}
\end{aligned}
$$

QODI is evaluated on three real world application domains: Life Science, Bibliography, and Conference Organization. QODI outperforms all baselines on all test cases.

## 5.1 Problem Definition

The following section begins with graph definitions and culminates with the formal definition of the mapping problem.

### 5.1.1 Basic Graph Definition

An *ontology graph* is a representation of an ontology as a directed labeled graph, where classes and datatypes are vertices, and properties are edges (see Figure 5.2). Target and source ontologies are distinguished as $T$ and $S$, respectively. These notations are used interchangeably to denote ontologies and ontology graphs. To simplify handling inheritance relationships, rather than coding the logic of inheritance into the path-related algorithms, an ontology graph is expanded by replicating properties. If the domains or ranges of a property have subclasses, new edges with the same label as that property are created for each subclass.

**Definition 5.1.1** (source and sink)**.** In a directed labeled graph $G$, a source is a vertex with 0 in-degree, and a sink is a vertex with 0 out-degree. The sets of all sources and sinks of $G$ are denoted $SOURCE_G$ and $SINK_G$, respectively.

**Definition 5.1.2** (ss-path)**.** A source-to-sink path or ss-path is a path from a vertex $v_1$ to a vertex $v_2$ in a directed labeled graph $G$, where $v_1$ is a source and $v_2$ is a sink of $G$.

For convenience, we represent a path $p$ as an ordered list of vertices and edges, and define the length, denoted as $|p|$, as the sum of the number of vertices and edges in $p$.

**Definition 5.1.3** (ss-path-set)**.** The set of all possible ss-paths from source $v_1$ to sink $v_2$ in a directed labeled graph $G$ is called an ss-path-set (denoted as SS-PATH-SET$_{G,v_1,v_2}$).

**Definition 5.1.4** (graph-ss-path-set)**.** Given a directed labeled graph $G$, the set of all ss-paths (denoted as GRAPH-SS-PATH-SET$_G$) is the union of all ss-path-sets from all sources to all sinks in $G$.

**Definition 5.1.5** (query graph)**.** Given a SPARQL query $q$ over ontology $T$, a query graph (denoted as $Q$) is a subgraph of $T$ that corresponds to $q$.

The query graph of the SPARQL query in Figure 5.3a is shown in Figure 5.3b.

### 5.1.2 Assumptions

Basic assumptions are as follows:

1. All object properties and datatype properties have domains and ranges. This assumption simplifies the construction of ontology graphs. High quality manually designed ontologies will detail domains and ranges. Ontologies automatically translated from relational schemas include domains and ranges [80, 81].

2. We consider conjunctive SPARQL queries in the SELECT query form, and exclude variables from the predicates of triple patterns. For each variable, the class, which is the type that the variable is binding to, either can be inferred from the domains or ranges of predicates or is provided by rdf:type. Given these assumptions, there exists only one query graph for each query. If multiple query graphs are allowed, each of them can be mapped separately. For simplicity, we leave the relaxing of these assumptions for future work.

3. The sinks of a query graph only represent datatypes. This work concerns ontologies that describe database content and queries that retrieve information from databases. Retrieving database data ultimately requires the rewriting of datatype properties.

### 5.1.3 Query-Specific Ontology Mapping

The following definitions define query-specific ontology mapping, which is the core problem of this work. An ss-path correspondence records the map-

ping confidence between two ss-paths.

**Definition 5.1.6** (ss-path correspondence)**.** Given two directed labeled graphs $G$ and $G'$, an ss-path correspondence between two ss-paths $p$ and $p'$ (denoted by $\pi_{p,p'}$) is $< p, p', \alpha_{\pi_{p,p'}} >$, such that $p \in$ GRAPH-SS-PATH-SET$_G$, $p' \in$ GRAPH-SS-PATH-SET$_{G'}$, and $\alpha_{\pi_{p,p'}}$ is a confidence measure between 0 and 1.

A *match candidate* is a set of ss-path correspondences between the ss-paths in the query graph, and the ss-paths in a subgraph of the source ontology graph.

**Definition 5.1.7** (match candidate)**.** Given a query graph $Q$, a match candidate $\Omega_{Q,G}$ is a set of ss-path correspondences between the ss-paths in $Q$ and the ss-paths in a graph $G$, which is a subgraph of the source ontology $S$, if the following conditions are satisfied:

- The sinks of $G$ are datatypes;
- for each ss-path $p \in$ GRAPH-SS-PATH-SET$_Q$, there exists exactly one ss-path correspondence $\pi_{p,p'} \in \Omega_{Q,G}$, where $p' \in$ GRAPH-SS-PATH-SET$_G$;
- for each ss-path $p' \in$ GRAPH-SS-PATH-SET$_G$, there exists ss-path correspondences $\pi_{p,p'} \in \Omega_{Q,G}$, where $p \in$ GRAPH-SS-PATH-SET$_Q$;

- for each pair of ss-paths $p_1, p_2 \in$ GRAPH-SS-PATH-SET$_Q$, if they share a common source, then the two corresponding ss-paths $p'_1, p'_2 \in$ GRAPH-SS-PATH-SET$_G$ also share a common source, where $\pi_{p_1,p'_1} \in \Omega_{Q,G}$, $\pi_{p_2,p'_2} \in \Omega_{Q,G}$.

Definition 5.1.7 contains several constraints. First, all sinks of $G$ are required to be datatypes, because the sinks of the query graph $Q$ are also datatypes. Second, we are interested in a one-to-one mapping, which restricts each ss-path in $Q$ to be contained in exactly one correspondence. Third, if the ss-paths in $Q$ share a source, the mapped ss-paths in $G$ also share a source. We assign a confidence measure $\beta_{\Omega_{Q,G}}$, which is defined as the product of all ss-path correspondence confidence measures:

$$\beta_{\Omega_{Q,G}} = \prod_{\pi_{p,p'} \in \Omega_{Q,G}} \alpha_{\pi_{p,p'}}$$

The task of query-specific ontology mapping, *q-mapping*, is to find the match candidate with the highest confidence.

**Definition 5.1.8** (q-mapping)**.** Given two ontology graphs $T$, $S$, and a SPARQL query $q$ over $T$, the query-specific ontology mapping (denoted q-mapping($T$,$S$,$q$)) is the set of ss-path correspondences $\Omega_{Q,\bar{G}}$, where $Q$ is the query graph, and $\bar{G}$ is a subgraph of $S$, such that $\Omega_{Q,\bar{G}}$ is a match candidate, and $\beta_{\Omega_{Q,\bar{G}}} = \max_{G \subseteq S} \beta_{\Omega_{Q,G}}$.

## 5.2 QODI: Mapping and Reformulation

The goals of mapping include defining a similarity score between two ss-paths, and determining the highest scoring ss-path correspondences without an exhaustive search.

### 5.2.1 ss-path Similarity Measure

The ss-path similarity measure must be able to disambiguate uncertain mappings. Given a pair of ss-paths, the similarity is defined as a product of four factors: similarity between source classes, similarity between datatype properties, similarity between path labels, and a penalty for path length differences. The source class and datatype property determine the two ends of an ss-path. A path label, containing the labels of all entities except datatypes in an ss-path, is used to disambiguate uncertain mappings.

Similarity estimation of source classes and datatype properties has been well studied in prior work [23, 58, 35]. Any existing method may be used for this component. In the experiments, we evaluated both simple string distance and sophisticated ontology matchers. The similarity between all classes and datatype properties can be computed beforehand and stored as similarity matrices for lookup.

We borrow techniques from information retrieval to measure the similarity between path labels. For an ss-path, we process the labels of all entities except datatypes in the path using linguistic processing, and add the processed strings to a list. The linguistic processing includes tokenization by

punctuation, numbers, and uppercase letters (if the letter is not preceded by an uppercase letter); stop words removal; and stemming (using SimPack[1]). All strings are converted to lowercase. A feature vector is generated by indexing the list of strings, and using frequencies as features. Given that different labels may contain a different number of tokens, the frequency of a token is set to one over the number of tokens in a label. The path label similarity, $S_L$, is computed as the intersection between the two feature vectors.

$$S_L(p, p') = \frac{\sum_{i=1}^{m} \min(\mathbf{f}_i(p), \mathbf{f}_i(p'))}{\sum_{i=1}^{m} \mathbf{f}_i(p) + \mathbf{f}_i(p') - \min(\mathbf{f}_i(p), \mathbf{f}_i(p'))} \tag{5.1}$$

where $\mathbf{f}_i(p)$ is the $i$th element of the feature vector of ss-path $p$, and $m$ is the dimension of the feature vectors.

If two paths are similar, their lengths may not have a large difference. We use an exponential function to penalize the path length difference. During mapping generation, this penalization prevents infinite loops when the ontologies contain cycles. The ss-path similarity measure, $S_{SS}$, is defined as,

$$S_{SS}(p, p') \;=\; S_C(p, p')^{\frac{1}{n_p}} \cdot S_D(p, p') \cdot S_L(p, p') \cdot e^{-\eta \cdot |\,|p|-|p'|\,|} \tag{5.2}$$

where $S_C$ and $S_D$ are similarity measures for source classes and datatype properties, which are provided by matchers. $|p|$ is the length of path $p$, and $\eta$ is a non-negative real number. $n_p$ is the number of ss-paths in the query graph that share the same source with $p$. $n_p$ is introduced because the same similarity between sources will be multiplied $n_p$ times when measuring the confidence of a match candidate.

[1] files.ifi.uzh.ch/ddis/oldweb/ddis/research/simpack/

### 5.2.2 q-mapping

We denote the set of all possible match candidates of query graph $Q$ as $\mathcal{M}_Q$. $\bar{G}$, which is the subgraph of $S$ involved in the match candidate with the highest similarity, is determined by maximizing the confidence measure. q-mapping($T$,$S$,$q$) is the set of ss-path correspondences $\Omega_{Q,\bar{G}}$ between $Q$ and $\bar{G}$.

$$\begin{aligned}\bar{G} &= \arg\max_{\Omega_{Q,G} \in \mathcal{M}_Q} \beta_{\Omega_{Q,G}} \\ &= \arg_{G \subseteq S} \{ \prod_{c \in SOURCE_Q} \{ \max_{c' \in SOURCE_G} \{ \\ &\qquad \prod_{p \in \text{SS-PATH-SET}_{Q,c,*}} \{ \max_{p' \in \text{SS-PATH-SET}_{G,c',*}} S_{SS}(p,p') \} \} \} \} \end{aligned} \tag{5.3}$$

where $\beta_{\Omega_{Q,G}}$ is the confidence measure of the match candidate, and SS-PATH-SET$_{G,c,*}$ represents the set of all ss-paths with source $c$ in $G$.

Equation (5.3) specifies the mapping as: for each source vertex in the query graph, find a vertex in the source ontology as a source vertex, such that the product of all ss-path similarities is the maximum.

### 5.2.3 Solving the Maximization

Equation (5.3) does not specify how to solve the maximization. A naive algorithm may score all possible match candidates. However, the number of all possible paths can be exponential in the number of vertices for acyclic ontology graphs, and is infinite for cyclic ontology graphs. It is infeasible to compute similarity between all pairs of paths. Thus, we employ heuristic

search algorithms to reduce the computation.

We decompose the search problem into two phases: 1) given an ss-path in the query graph and a vertex in $S$, search for the ss-path in $S$ with the given vertex as source that has the highest similarity; 2) given a set of ss-paths that share a source in the query graph, find a set of paths in $S$ that share a source and have the highest product of similarities. Phase 1) is a subproblem of 2). Thus, we solve 1) then 2).

Phase 1) can be solved by a heuristic search algorithm similar to A* search. A* is commonly applied to find a minimal cost path in a graph [44]. A* requires a function that computes the cost of a partial path, and a heuristic cost function that estimates the cost of completing a path. The search is guaranteed to terminate with an optimal path if the heuristic is admissible. We cannot exploit the traditional structure of A* search. Our definition of path similarity considers all labels in a path as a bag of words. Thus, we can not decompose a partially computed answer into the sum of two functions. We define a single function that, given a partial path, will never overestimate the cost of a complete optimal path. With similar proof as A* search given below, our heuristic search is guaranteed to find an optimal path. The implementation of the search algorithm remains largely unchanged. Search states, representing partial paths, are saved in an open-list $\mathcal{P}$. $\mathcal{P}$ is initialized by the path that only contains one vertex (the given vertex). The paths in $\mathcal{P}$ are sorted in ascending order using our heuristic function. The search terminates when a path $\bar{p}$ containing a sink (datatype) is pulled from $\mathcal{P}$.

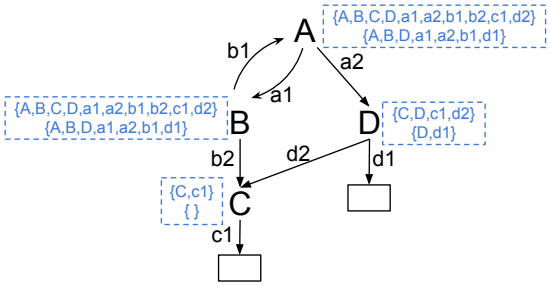



Figure 5.4: An example ontology graph with reachable label sets. The dashed boxes around each non-sink vertex contain the reachable label sets through the two datatype properties $c1$ and $d1$. For example, the reachable label set from $C$ through $c1$ is $\{C, c1\}$, and through $d1$ is empty.

We introduce two techniques to help create the heuristic cost function. First, the similarity between datatype properties ($S_D$), which is a factor of $S_{SS}$, is considered at the beginning of the search. A datatype property is the last edge in an ss-path, and connects to a datatype. Thus, a large amount of computation can be potentially wasted by the search before discovering the similarity between datatype properties is low. To address this, $\mathcal{P}$ is initialized by a set of paths, each of which only contains the given vertex and only leads to the sink through a specific datatype property. Following that, $S_D$ is a constant for each path. $S_C$ is also a constant, since the source vertex is given. Only the cost of adding new vertices and edges to the path needs to be considered.

The second technique is a preprocessing step that associates reachable label sets and shortest path lengths to each class. We define a reachable label set from a vertex through a datatype property as the union of the path labels of all possible paths from the vertex to a datatype through the datatype property. Each vertex of $S$ is associated with the reachable label sets, from itself through

**Algorithm 4** Generate reachable label sets.

**Input:** ontology graph $G$ without reachable label sets
**Output:** ontology graph $G$ with reachable label sets

```
for all vertex v of G do
    for all sink s of G do
        v.reachable[s] = ∅
    end for
end for
// q stores all vertices that will be expanded
Queue q is initialized to contain all sinks of G
while q ≠ ∅ do
    v = q.dequeue()
    for all parent vertex p of v do
        // changed records whether p.reachable is changed
        changed = false
        E_p is the set of all edges from p to v
        for all sink s of G do
            // prop is the set of entities that will be propagated
            prop = v.reachable[s] ∪ E_p ∪ {p}
            if p.reachable[s] ⊉ prop then
                p.reachable[s] = p.reachable[s] ∪ prop
                changed = true
            end if
        end for
        if changed == true then
            q.enqueue(p)
        end if
    end for
end while
for all vertex v of G do
    for all sink s of G do
        set v.reachable_labels[s] as the labels of v.reachable[s]
    end for
end for
return G
```

each datatype property. Figure 5.4 illustrates an example of reachable label sets. The reachable label sets are computed by recursively propagating the reachable label sets of each vertex to its parents. The algorithm terminates when the reachable label sets are not changed for all vertices. The pseudo code is detailed in Algorithm 4. The worst case complexity of this algorithm is quadratic in the number of vertices. Given that a reachable label set is a superset of the labels that may appear in an optimal path, an admissible heuristic can be defined to guarantee the optimality. In addition, each vertex of $S$ is also associated with the lengths of the shortest paths from itself to datatypes through each datatype property. The lengths of shortest paths are also used in the heuristic. Note that the preprocessing only need run once.

We denote the ss-path in the query graph as $r$, the path in $S$ that needs heuristic scoring as $p$, the last element of $p$ as $x$, and the objective datatype as $e$. The reachable label set from $x$ to $e$ is denoted as $L_{x,e}$, and the length of the shortest path from $x$ to $e$ is denoted as $l_{x,e}$. The heuristic cost function $h$ is defined as follows:

$$h(p) = - S_C(r,p)^{\frac{1}{n_r}} \cdot S_D(r,p) \cdot \frac{\sum_{i=1}^{m} \min(\mathbf{f}_i(r), \mathbf{f}_i(p) + \bar{\mathbf{f}}_i(r,p,L_{x,e}))}{\sum_{i=1}^{m} \mathbf{f}_i(r) + \mathbf{f}_i(p) - \min(\mathbf{f}_i(r), \mathbf{f}_i(p))} \cdot e^{-\eta \cdot g(r,p,l_{x,e})} \tag{5.4}$$

where $n_r$ is the number of paths in the query graph that share the same source as $r$, and $\mathbf{f}_i(p)$ is the $i$th element of the feature vector of path $p$. $\bar{\mathbf{f}}_i(r,p,L_{x,e})$

and $g(r, p, l_{x,e})$ are:

$$\bar{\mathbf{f}}_i(r, p, L_{x,e}) = \begin{cases} \max(\mathbf{f}_i(r) - \mathbf{f}_i(p),\ 0) & \text{, if } x \neq e \text{ and string } i \in L_{x,e} \\ 0 & \text{, otherwise} \end{cases} \tag{5.5}$$

$$g(r, p, l_{x,e}) = \begin{cases} \max(|p| + l_{x,e} - 1 - |r|,\ 0) & \text{, if } x \neq e \\ |\ |p| - |r|\ | & \text{, if } x = e \end{cases} \tag{5.6}$$

Comparing (5.4) with (5.2), $h$ is derived from the negation of $S_{SS}$ by substituting a real path by an estimation. Let us denote the path as $\bar{p}$, when the search terminates. Based on the termination condition, $x = e$. Substituting $x$ with $e$, $h(\bar{p}) = -S_{SS}(r, \bar{p})$. The following lemma and theorem prove that $\bar{p}$ is the best scoring path. Lemma 5.2.1 corresponds to the proof of admissibility of the heuristic in A* search.

**Lemma 5.2.1.** *Suppose the search has not terminated. For any optimal path $\tilde{p}$, there exists a path $p$ in the priority queue $\mathcal{P}$, which can be expanded to $\tilde{p}$, such that $h(p) \leq h(\tilde{p})$.*

*Proof.* $h$ is the negation of a product of four non-negative factors. We will prove that each factor of $h(p)$ is greater than or equal to the corresponding factor of $h(\tilde{p})$. Then $h(p) \leq h(\tilde{p})$.

The first two factors, $S_C$ and $S_D$, are the same for both $p$ and $\tilde{p}$.

Consider the third factor. Denote the last element in path $p$ as $x$. The reachable label set, $L_{x,e}$, contains the labels of all possible paths from $x$ to $e$, including those in $\tilde{p}$. Per the definition of $\bar{\mathbf{f}}_i$, the numerator in $h(p)$ is greater than or equal to that in $h(\tilde{p})$. Since $p$ is a sub-path of $\tilde{p}$, the denominator

in $h(p)$ is less than or equal to that in $h(\tilde{p})$. Thus the third factor of $h(p)$ is greater than or equal to the third factor of $h(\tilde{p})$.

Consider the fourth factor. $l_{x,e}$ is the length of the shortest path from $x$ to $e$, so $|p| + l_{x,e} - 1 \leq |\tilde{p}|$. Consider two cases:

1. $|p| + l_{x,e} - 1 \geq |r|$. Then $g(r, p, l_{x,e}) = |p| + l_{x,e} - 1 - |r|$, and $g(r, \tilde{p}, l_{e,e}) = |\tilde{p}| - |r|$. Thus, $g(r, p, l_{x,e}) \leq g(r, \tilde{p}, l_{e,e})$.

2. $|p| + l_{x,e} - 1 < |r|$. Then $g(r, p, l_{x,e}) = 0$, and $g(r, \tilde{p}, l_{e,e}) \geq 0$. Thus, $g(r, p, l_{x,e}) \leq g(r, \tilde{p}, l_{e,e})$.

Thus the fourth factor of $h(p)$ is greater than or equal to the fourth factor of $h(\tilde{p})$. □

**Theorem 5.2.2.** *When the search terminates, the path $\bar{p}$ is an optimal path.*

The proof of Theorem 5.2.2 can be derived from the proof of the similar theorem for A* search by substituting the sum of the two cost functions with our heuristic $h(p)$ [44].

If there is no path from the given vertex through any datatype property, there is no solution for the search. The search algorithm terminates by knowing the reachable label sets of the vertex are all empty. Otherwise, the algorithm will find an optimal path with finite length, because $h$ of a path with infinite length is infinitesimal due to the path length penalty. Most real world queries do not have cycles, so we prune cyclic paths during the search to further

reduce the computation. This heuristic can be disabled for the applications with cyclic queries.

Phase 2) involves selecting a vertex as a source, and jointly finding multiple optimal paths that share a source. For each possible source class, we exploit the heuristic proposed in phase 1) to estimate the product of the highest path similarities of all paths as the score for the class. The algorithm in phase 1) runs using each class as the given source in descending order of the estimated score. If the real score of a class is greater than or equal to the estimated score of the remaining classes, those classes can be pruned. This algorithm also terminates with an optimal solution. The proof is similar to the proof of Theorem 5.2.2.

If the query graph has multiple sources, the algorithm in phase 2) runs for each source separately.

### 5.2.4 Query Reformulation

We briefly explain the benefits of using q-mapping for query reformulation. A central challenge in query reformulation is missing mapping. In QODI, this challenge manifests as a mapping between a path in the query graph and a path in the source ontology graph. The determination of an ss-path correspondence anticipates that the paths may be of different lengths.

Given ss-path correspondences as mapping, the reformulation algorithm is simplified as traversing the mapped ss-paths, and generating a triple pattern for each graph edge. The URI of each edge in the ss-path is translated as the

**Algorithm 5** Query reformulation.

**Input:** SPARQL query $q_t$ on ontology $T$, and q-mapping $M$
**Output:** SPARQL query $q_s$ on ontology $S$

```
// Select clause
Copy the select clause from q_t to q_s
// Where clause
q_s += "Where {"
for all ss-path correspondence π_{p,p'} in M do
    for all property e in path p' do
        if e is datatype property then
            Assign a variable c to the domain of e
            Assign the value v of the datatype in p to the range of e
            q_s += c + e.URI + v
        else
            Assign a variable c_1 to the domain of e
            Assign a variable c_2 to the range of e
            q_s += c_1 + e.URI + c_2
        end if
    end for
end for
q_s += "}"
return q_s
```

predicate of a triple. The subject and object of the triple are variables or literals assigned to the domain and range of the edge, respectively. Assigning variables to classes that are shared by multiple paths is an open research topic. We do not elaborate on this topic. Algorithm 5 details the reformulation algorithm. For the query in Figure 5.3, a path correspondence and the resulting translated triple patterns are:

{Course,teacher,People,name,string}

= {Course,offeredBy,Teacher,name,string}

?c1 course:offeredBy ?c2 .

?c2 teacher:name “Einstein” .

## 5.3 Experimental Setup

The objectives of the evaluation are threefold: determine how capable the method is of generating mappings, how accurate the mappings are, and finally, how well the method resolves ambiguity.

### 5.3.1 Test Sets

The test sets are detailed in Section 3.2. There are three application domains: Life Science, Bibliography, and Conference Organization. The test cases include an ontology created by an international standards body, two ontologies created from direct mapping relational databases, and three ontologies used in OAEI [2].

Two kinds of SPARQL queries are generated for each ontology. 1) A *PathOnly* query has a query graph consisting of only one ss-path in the groundtruth. 2) A *ClassAll* query has a query graph consisting of all ss-paths (at least two) that share a source in the groundtruth. A ClassAll query is the most complicated query with one conjunction over the source. In English specification, a PathOnly query asks for all values of a single attribute of a

```
BASE <http://ribs.csres.utexas.edu/specify/>
Select ?v
Where {
 ?c0 <locality#Latitude1> ?v.
 ?c0 rdf:type <locality>.
}
```

(a) PathOnly query, asking for the latitude of all locations.

```
BASE <http://ribs.csres.utexas.edu/specify/>
Select ?v0 ?v1 ?v2 ?v3 ?v4 ?v5 ?v6
Where {
 ?c0 <determination#DeterminedDate> ?v0.
 ?c0 <determination#Remarks> ?v2.
 ?c1 <taxon#Name> ?v3.
 ?c2 <taxon#Name> ?v4.
 ?c3 <agent#DateOfBirth> ?v5
 ?c4 <agent#DateOfBirth> ?v6.
 ?c1 rdf:type <taxon>.
 ?c3 rdf:type <agent>.
 ?c0 <determination#Qualifier> ?v1.
 ?c0 <determination#ref-TaxonID> ?c1.
 ?c0 <determination#ref-PreferredTaxonID> ?c2.
 ?c0 <determination#ref-CreatedByAgentID> ?c3.
 ?c0 <determination#ref-ModifiedByAgentID> ?c4.
 ?c0 rdf:type <determination>.
 ?c2 rdf:type <taxon>.
 ?c4 rdf:type <agent>.
}
```

(b) ClassAll query, asking for the dates, remarks, and qualifiers of all determination of taxons, as well as the birthdays of the agents that determine the taxons.

Figure 5.5: Real SPARQL queries generated for the Specify ontology in the experiments.

concept, and a ClassAll query asks for all values of all attributes of a concept. For ontology $T$ in Figure 5.2, a PathOnly query could ask for name of all students taking courses, and a ClassAll query could ask for titles, time, and name of all students of all courses. Figure 5.5 shows examples of real PathOnly and ClassAll queries generated for the Specify ontology, as well as the meaning of both queries.

### 5.3.2 Baselines

We compare QODI against two kinds of baselines: ontology matching systems, and an ontology-based implementation of an existing relational data integration system.

For ontology matching baselines, a matcher computes the similarity between classes, object properties, and datatype properties. Given a query, each entity is translated to an entity in $S$ with the highest similarity.

Clio is a relational data integration and exchange system that is closely related to QODI [37]. Clio generates mappings between attributes, and finds associations between those mappings through foreign key constraints. We implement baselines with similar ideas as Clio. A matcher first generates mappings between datatype properties by picking the ones with the highest similarity. Given a query, the baselines find the match candidates that contain all the mapped datatype properties. Clio asks a user to pick one match candidate, which is not allowed in our automated setting. We approximate this process by first picking the match candidates with highest similarity between source classes, and then picking the one with the least summation of path lengths.

We use three matchers for all methods. One matcher is substring string similarity that measures the portion of the longest common substrings between entity labels. The second matcher is SMOA string similarity between entity labels [83]. The third is AgreementMaker configured as detailed in OAEI 2010 conference track [23].

### 5.3.3 Metrics

The assessments are reminiscent of recall and precision used in ontology matching and information retrieval. *valid_rate* is the metric similar to

recall, which is the proportion of queries with *complete q-mappings* generated, independent of correctness. We use # to represent *the number of*.

**Definition 5.3.1** (complete q-mapping)**.** A q-mapping with a set of correspondences $\Omega_{Q,\bar{G}}$ is complete, if for every ss-path in the query graph $Q$, there exists a correspondence to an ss-path in $\bar{G}$ with non-zero confidence measure.

$$valid\_rate = \frac{\#\text{ queries with complete q-mappings generated}}{\#\text{ queries}} \tag{5.7}$$

For measuring the precision of mapping systems, we consider the case that a query is correctly mapped, and also the case that a query is partially correctly mapped.

$$query\_precision = \frac{\#\text{ correctly mapped queries}}{\#\text{ queries}} \tag{5.8}$$

$$path\_precision = \frac{\sum_q \text{percentage of correctly mapped ss-paths in } q}{\#\text{ queries}} \tag{5.9}$$

A measure of ambiguity can facilitate the analysis of experimental results. An accurate measure of ambiguity is difficult, since it has to anticipate all possible application scenarios. We define an approximate measure of ambiguity, which only considers mapping between datatype properties as the source of ambiguity, and considers two datatype properties as mapped if a matcher assigns them the highest similarity.

**Definition 5.3.2** (datatype ambiguous q-mapping)**.** Given a datatype property similarity measure $S_D$, a target ontology $T$, a source ontology $S$, a query

$q$ over $T$, and the set of ss-path correspondences $\Omega$ of q-mapping($T$,$S$,$q$), the mapping is datatype ambiguous if for at least one ss-path correspondence $\pi_{p_t,p_s} \in \Omega$, $S_D(p_t, p_s) = \max_p S_D(p_t, p)$, and there exists a datatype property $d \notin p_s$, such that the similarity between $d$ and the datatype property of $p_t$ equals $S_D(p_t, p_s)$.

*ambiguous_rate* is a measure of the proportion of queries that have datatype ambiguous q-mappings.

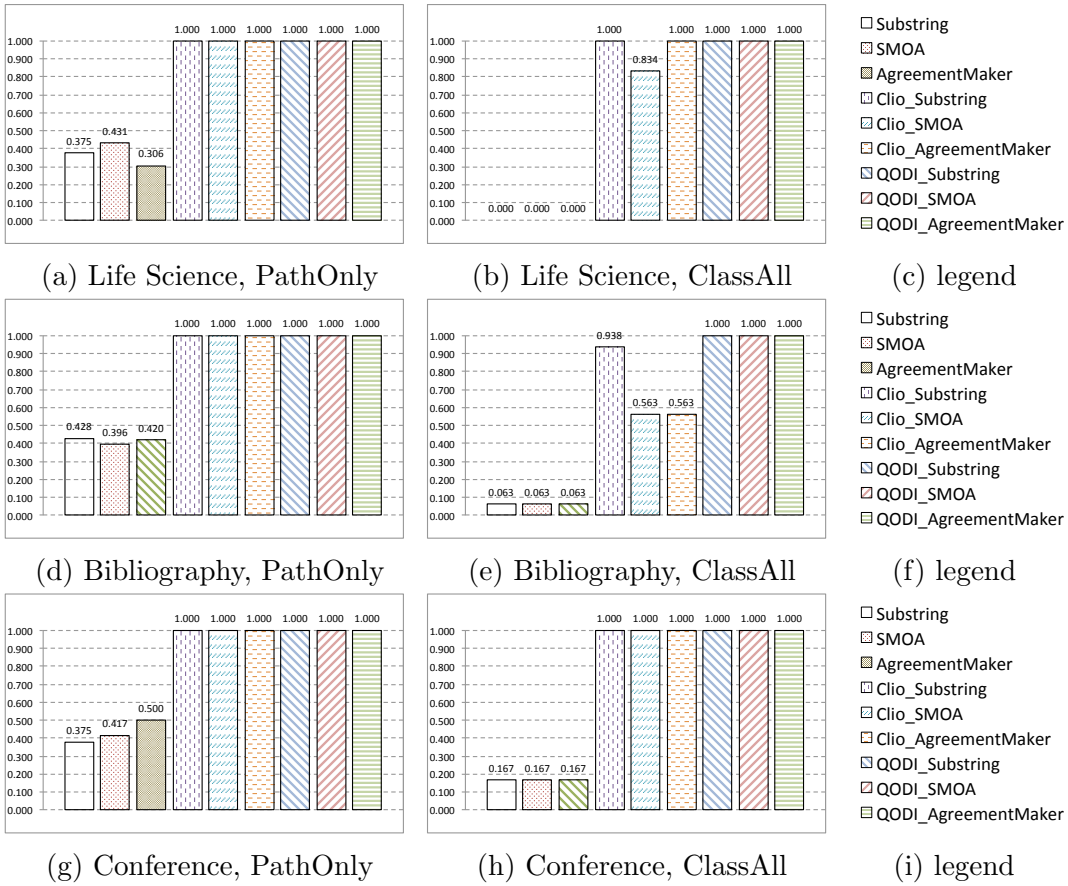
$$ambiguous\_rate = \frac{\#\text{ queries with datatype ambiguous q-mapping}}{\#\text{ queries}} \quad (5.10)$$

## 5.4 Experimental Results

Given a pair of ontologies, $O_1$ and $O_2$, the experiments are conducted on two directions of mappings: using $O_1$ as target and using $O_2$ as target. The results for the two mapping directions are shown separately for ambiguous_rate to distinguish the differences. For other metrics, the results are averaged. We set $\eta = 0.3$ based on the tuning on the Bibliography test set with PathOnly queries using Substring as matcher. Section 5.4.3 discusses the accuracy using different $\eta$.

### 5.4.1 valid_rate

Figure 5.6 shows the valid_rate for each test set. The three methods of QODI achieve 100% valid_rate for all test sets. This is because QODI does not determine any entity mapping beforehand. Each path correspondence is




(a) Life Science, PathOnly (b) Life Science, ClassAll (c) legend

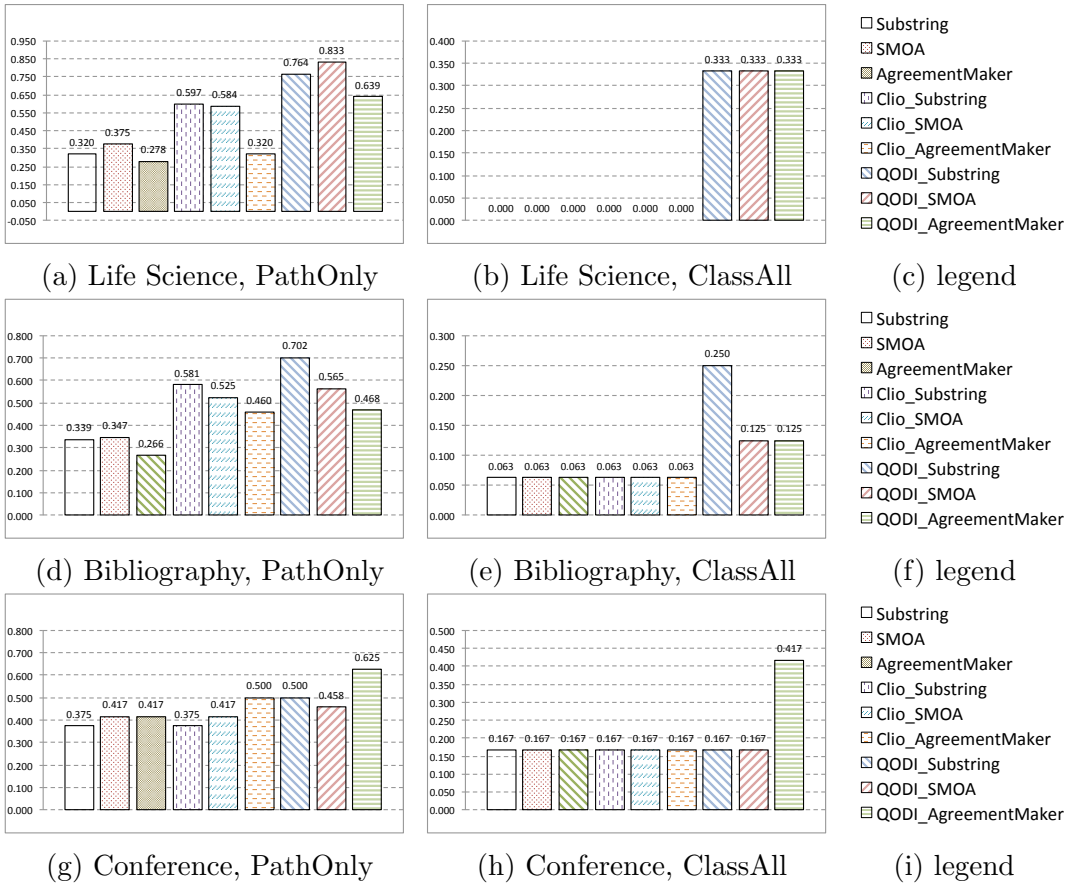
(d) Bibliography, PathOnly (e) Bibliography, ClassAll (f) legend

(g) Conference, PathOnly (h) Conference, ClassAll (i) legend

Figure 5.6: valid_rate for different test sets. Higher number means better performance.

assigned a confidence, and the mapped paths has the highest confidence.

The Clio baselines are able to generate complete mappings for two thirds of test sets. Bibliography and Life Science with ClassAll query set are the exceptions. Bibliography and Life Science have many datatype ambiguous q-mappings. For some ClassAll queries, Clio cannot find a complete q-mapping if the mapped entities are incorrectly selected from ambiguous mappings. The




(a) Life Science, PathOnly (b) Life Science, ClassAll (c) legend

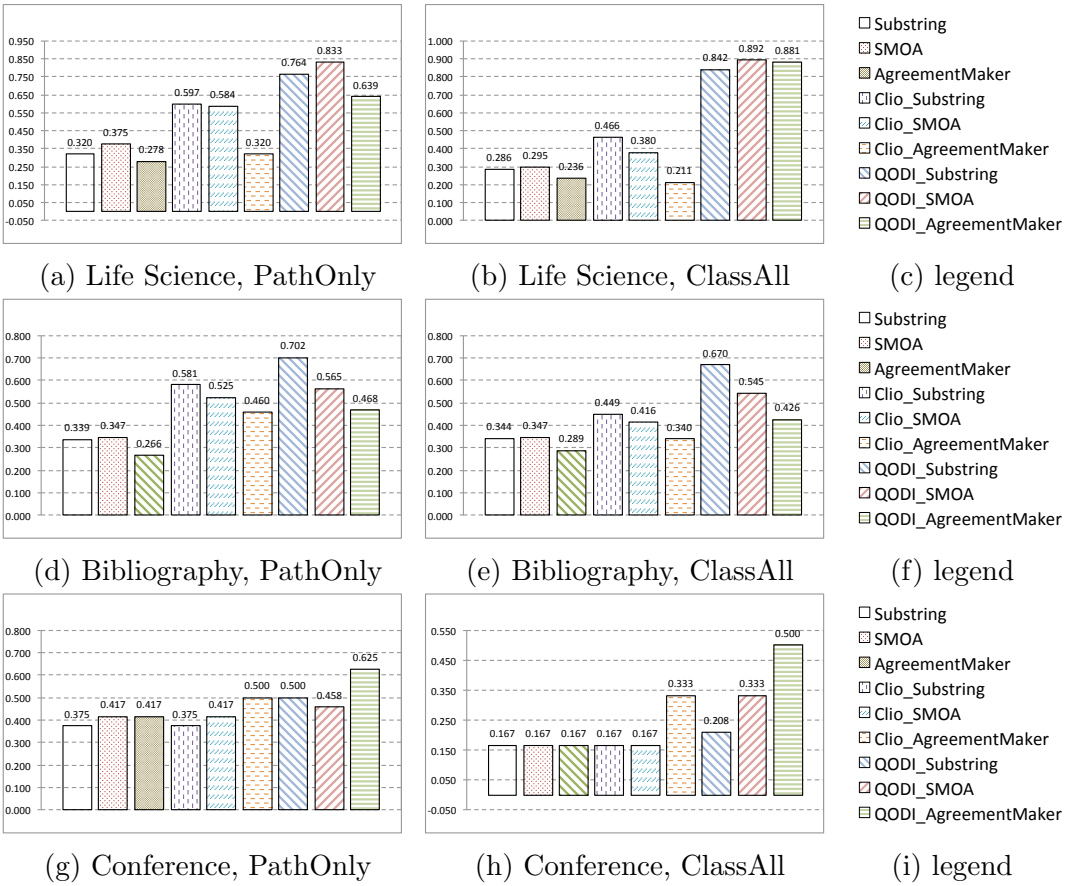
(d) Bibliography, PathOnly (e) Bibliography, ClassAll (f) legend

(g) Conference, PathOnly (h) Conference, ClassAll (i) legend

Figure 5.7: query_precision for different test sets.

comparison between QODI and Clio shows that disambiguation is important even for generating complete q-mappings regardless of correctness.

The ontology matching baselines can generate complete q-mappings for less than 50% of PathOnly queries, but barely generate complete q-mappings for ClassAll queries. The big gap between ontology matching baselines and Clio baselines demonstrates the importance of the missing mapping challenge.




(a) Life Science, PathOnly (b) Life Science, ClassAll (c) legend

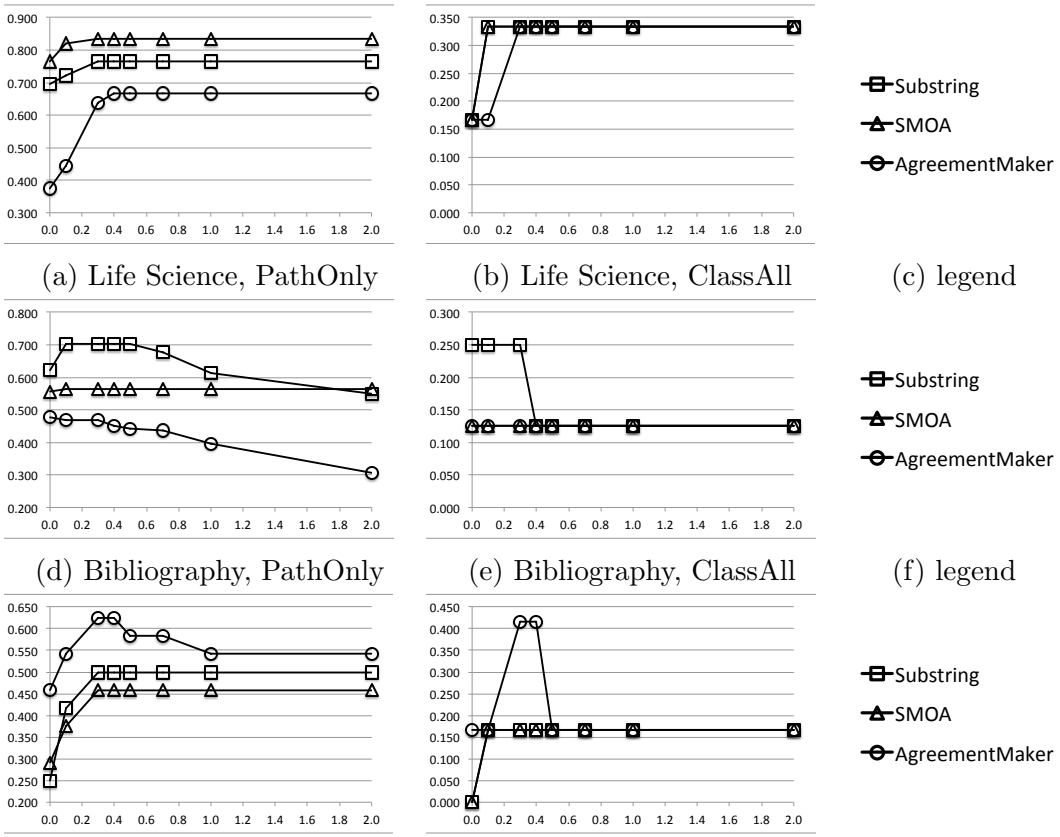
(d) Bibliography, PathOnly (e) Bibliography, ClassAll (f) legend

(g) Conference, PathOnly (h) Conference, ClassAll (i) legend

Figure 5.8: path_precision for different test sets.

### 5.4.2 Precision

Figure 5.7 and 5.8 show the precisions of all methods. For all test sets, at least one QODI method dominates all baselines in terms of both precision measures. For ClassAll query sets, there are big gaps between QODI and all baselines. QODI is the only system that achieves non-zero query_precision for the Life Science test set with ClassAll query set. For ClassAll query set, each query has more than one path that shares a source. On one hand, more paths




(a) Life Science, PathOnly (b) Life Science, ClassAll (c) legend

(d) Bibliography, PathOnly (e) Bibliography, ClassAll (f) legend

(g) Conference, PathOnly (h) Conference, ClassAll (i) legend

Figure 5.9: query_precision using different $\eta$ (horizontal axis).

may lead to poor mapping results since each path may be mapped incorrectly. On the other hand, the context from different paths may be used by QODI to map the correct source class shared by the paths. The precision results indicate the importance of resolving the ambiguous mapping challenge.

Comparing Clio with ontology matching baselines, for all test sets and all measures, at least one Clio baseline dominates or performs as well as ontology matching baselines.

| | L↑ | L↓ | B↑ | B↓ | C↑ | C↓ |
|---|---|---|---|---|---|---|
| ambiguous_rate | 0.333 | 0.111 | 0.242 | 0.113 | 0.000 | 0.000 |
| Substring | 0.333 | 0.250 | 0.267 | 0.000 | - | - |
| Clio_Substring | 0.417 | **0.500** | 0.667 | 0.000 | - | - |
| QODI_Substring | **0.500** | **0.500** | **0.933** | 0.000 | - | - |

Table 5.1: The ambiguous_rate (row 2) and query_precision of queries with datatype ambiguous q-mapping (row 3, 4, 5) using Substring as matcher. L↑ uses Darwin Core and L↓ uses Specify as the target ontology for the Life Science test set. B↑ uses UMBC and B↓ uses DBLP as the target ontology for the Bibliography test set. C↑ uses SIGKDD and C↓ uses SOFSEM as the target ontology for the Conference test set. If ambiguous_rate is zero, there is no query_precision for the queries with datatype ambiguous q-mapping. Higher query_precision means better performance.

### 5.4.3 Parameter Tuning

Figure 5.9 shows the query_precision using different path length penalty parameter $\eta$. With the same length difference, a large $\eta$ gives big penalty. As a special case, $\eta = 0$ does not have any penalty on the path length.

For most cases, the penalty improves query_precision comparing to the case of $\eta = 0$. However, if $\eta$ is too large, the query_precision can be decreased. With large $\eta$, the penalty of length dominates the similarities of source classes, datatype properties, and path labels in (5.2). Thus only the paths with the same lengths are considered as similar, ignoring the labels of the paths. For Life Science, most of the mapped paths in the groundtruth have the same length, so the precision is not decreased with large $\eta$.

| | L↑ | L↓ | B↑ | B↓ | C↑ | C↓ |
|---|---|---|---|---|---|---|
| ambiguous_rate | 0.194 | 0.000 | 0.177 | 0.000 | 0.000 | 0.000 |
| SMOA | 0.143 | - | 0.364 | - | - | - |
| Clio_SMOA | 0.429 | - | 0.364 | - | - | - |
| QODI_SMOA | **0.714** | - | **0.636** | - | - | - |

Table 5.2: The ambiguous_rate and query_precision of queries with datatype ambiguous q-mapping using SMOA as matcher. See caption of Table 5.1 for details.

| | L↑ | L↓ | B↑ | B↓ | C↑ | C↓ |
|---|---|---|---|---|---|---|
| ambiguous_rate | 0.139 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| AgreementMaker | 0.000 | - | - | - | - | - |
| Clio_AgreementMaker | 0.000 | - | - | - | - | - |
| QODI_AgreementMaker | **0.200** | - | - | - | - | - |

Table 5.3: The ambiguous_rate and query_precision of queries with datatype ambiguous q-mapping using AgreementMaker as matcher. See caption of Table 5.1 for details.

### 5.4.4 Ambiguity

As the primary motivation is the identification that mapping correctness may be query dependent (ambiguous), we assess how much of QODIs improved performance over Clio is explained by the presence of ambiguity and the respective systems ability to resolve it. In this section, we measure the ambiguous_rate of all test sets, and compute the query_precision of all methods over PathOnly queries with ambiguous mappings to measure the capability of disambiguation. If there is no ambiguity, the precision column is empty.

Per Definition 5.3.2 and Equation (5.10), ambiguous_rate is related to matchers. The result using each matcher is reported separately in Table 5.1,

5.2, and 5.3.

Two out of three test sets, Life Science and Bibliography, have non-zero ambiguous_rate. The ambiguous_rate measured with different matchers share similarities. All three matchers assert that Life Science with Darwin Core as target ontology has ambiguity, with rates from 0.139 to 0.333. Substring and SMOA agree on the ambiguity of Bibliography with UMBC as target ontology, with rates 0.242 and 0.177. For both L↑ and B↑, QODI achieves the highest query_precision on the queries with datatype ambiguous q-mappings. Comparing with Clio, the relative improvement of QODI is 66% and 75%. This shows that QODI is capable of disambiguation.

## 5.5 Discussion and Future Work

This work identifies ambiguous mappings as sources of errors in automatic data integration. In our experiments, two out of three test domains have ambiguity. Around 10% to 30% of queries involve ambiguous mappings in the test sets with ambiguity.

We introduce query-specific ontology mapping to resolve ambiguous mappings, and implement an OBDI system, QODI. For all test sets in our evaluation, at least one QODI method outperforms all baselines with both precision measures.

Future work consists of at least three possible directions. First, new methods of path similarities can be explored. In our current implementation,

contexts are extracted as path labels, and factors of the path similarity are multiplied as probabilities. Other methods that extract context from queries as well as weighting schemes that balance different factors should be studied. Second, although the focus of this presentation is algorithmic, the fundamental organization of QODI admits integration of user interaction for refinement. Users input may be integrated at different places. Similarity between entities is pre-computed and stored in matrices. Values in the matrices may be overwritten directly by users. When a q-mapping is generated, users may label the correctness of path correspondences. After executing the reformulated query, users may label the correctness of the query results. These labels can be exploited by machine learning algorithms to subsequently adjust both path mapping and similarity between entities. Third, path mappings can be accumulated over time as in pay-as-you-go systems. Although QODI may choose different entity mappings for different queries, the path mapping is static for a specific path. This motivates a design of workload to accumulate path mappings over time, such that a path mapped before need not be mapped in future.

# Chapter 6

# Complete Data Integration System

In the Semantic Web literature, most existing research focuses on subproblems of data integration. Ontology matching finds correspondences between ontology entities [35]. Instance matching assigns "owl:sameAs" to instances that refer to the same real world object [69]. Query rewriting algorithms are proposed for particular mapping representations [55]. Each subproblem has its own assumptions that may not be compatible with others. Problems on complete data integration system have not been well studied.

This work details Alamo, a semi-automatic data integration system using Semantic Web techniques. Figure 6.1 shows the system diagram of Alamo. The inputs of Alamo are the target ontology and a set of data sources. Users issue queries in terms of the target ontology through an application, and Alamo will translate them over data sources and return the federated query results as well as provenance. The existence of Alamo separates the application from data sources. Thus, new data sources can be added without modifying the application software, which is critical due to the scale and the distributed nature of RDF data. Alamo supports both RDF triple stores and relational databases as data sources. For relational databases, Alamo uses Ultrawrap to

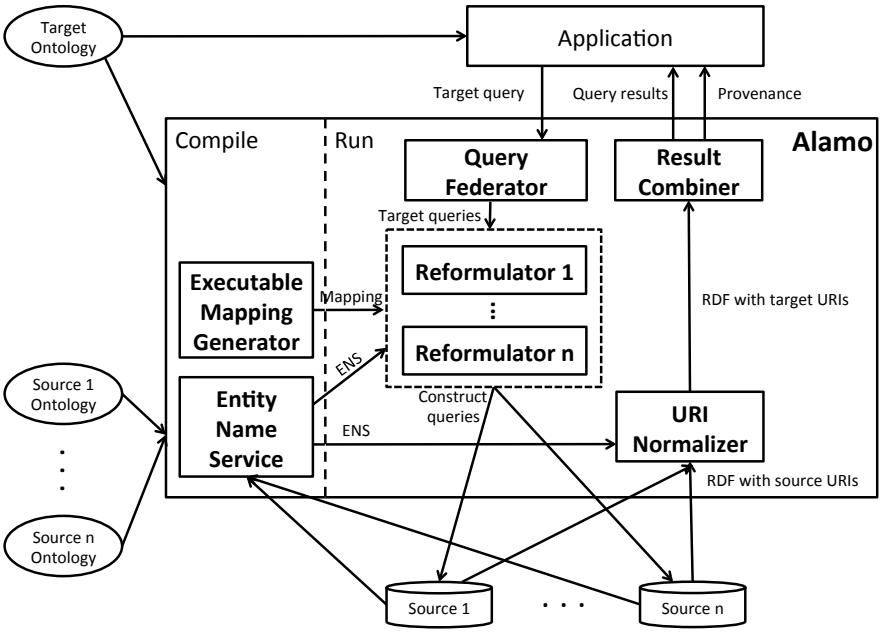
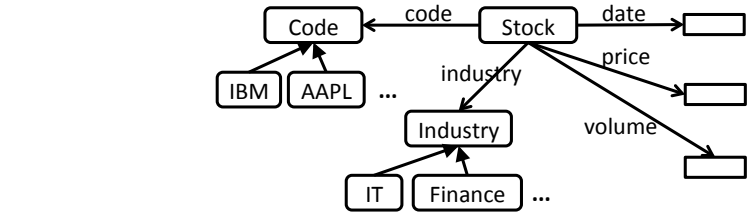



Figure 6.1: Alamo system diagram.

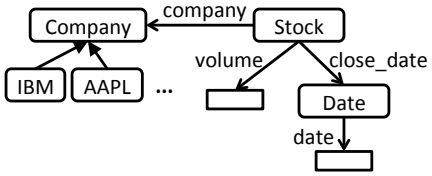
automatically translate to RDF and OWL [81]. For clarity, we assume data sources are RDF triple stores with ontologies as data models.

Alamo is a large system that consists of many components. My contribution focuses on the two automatic components central to the system: mapping generator and query reformulator. The rest of this chapter overviews the whole system, and details the mapping generator and query reformulator.

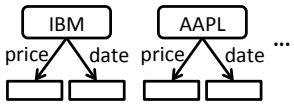
We will illustrate Alamo using the stock price examples in Figure 6.2, including the target ontology and two source ontologies as well as their RDF triples. These examples are derived from the real examples introduced in a seminal paper by Krishnamurthy, Litwin, and Kent [54]. The integration of these examples requires higher-order logic.




(a) Target ontology




(b) Source ontology 1




(c) Source ontology 2

| Subject | Predicate | Object |
|---|---|---|
| src1:stk1 | src1:company | src1:IBM |
| src1:stk1 | src1:volume | 5.6m |
| src1:stk1 | src1:close_date | src1:date1 |
| src1:date1 | src1:date | 1/3/2012 |

(d) RDF triple store 1

| Subject | Predicate | Object |
|---|---|---|
| IBM:ibm1 | IBM:price | 186.3 |
| IBM:ibm1 | IBM:date | 1/3/2012 |
| AAPL:aapl1 | AAPL:price | 411.2 |
| AAPL:aapl1 | AAPL:date | 1/3/2012 |

(e) RDF triple store 2

Figure 6.2: Stock price examples including the target ontology and two data sources. Source 1 contains trading volumes, and source 2 contains stock prices.

## 6.1 Compile Time

In the compile time, Alamo finds the relationship between the target ontology and source ontologies through executable mapping generator. The relationship between instances in different data sources are discovered through entity name service. The compile time components only need to run once for each new data source.

### 6.1.1 Executable Mappings

An executable mapping is a mapping representing transformations of instances [13]. In relational database, an executable mapping is a logic formula over relations and attributes. A common representation is the tuple-generating dependency (TGD), which is a first-order logic containing conjunctions of atomic formulas [11].

We define executable mappings in the Semantic Web following a similar representation of TGDs. An executable mapping is represented as a logic formula:

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y})) \tag{6.1}$$

where $\phi(\mathbf{x})$ is a conjunction of triples from the source and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of triples from the target.

To physically represent the mappings, we use construct SPARQL queries as Rivero et. al. [78]. There are two reasons to represent mappings as SPARQL queries: (1) SPARQL queries can be easily used in the Semantic Web applications, since SPARQL is the standard query language; (2) SPARQL queries are capable of expressing conjunctive logic formulas. Given an executable mapping, the head of the construct query contains the logic formulas from the target ($\psi$), and the body contains the formulas from the source ($\phi$). Figure 6.3 and 6.4 show some executable mappings between the target ontology and the two data sources in our example, respectively.

Using SPARQL query as the executable mapping representation, any

```
CONSTRUCT {
 ?x target:code ?y.
} WHERE {
 ?x src1:company ?y.
}
```

(a) stock code

```
CONSTRUCT {
 ?x target:volume ?y.
} WHERE {
 ?x src1:volume ?y.
}
```

(b) trading volume

```
CONSTRUCT {
 ?x target:date ?y.
} WHERE {
 ?x src1:close_date ?z.
 ?z src1:date ?y.
}
```

(c) trading date

Figure 6.3: Executable mappings between the target and source 1.

```
CONSTRUCT {
 ?x target:code target:IBM.
 ?x target:price ?y.
} WHERE {
 ?x IBM:price ?y.
}
```

(a) IBM stock price

```
CONSTRUCT {
 ?x target:code target:IBM.
 ?x target:date ?y.
} WHERE {
 ?x IBM:date ?y.
}
```

(b) IBM trading date

Figure 6.4: Executable mappings between the target and source 2. Due to space limit, only mappings related to IBM are shown. Similar mappings exist for other stocks.

ontology mapping generator that produces mappings in this format can be used in Alamo. However, generating executable mappings is more difficult than matching individual ontology entities, because the formulas in executable mappings can contain a set of conjunctive atomic formulas. Alamo generates the mappings by the two automatic mapping systems developed in previous chapters [87, 85]. The mappings can be also adjusted manually.

The first system is detailed in Chapter 4. It is extended from relational databases to the Semantic Web. For each class in the target ontology and each class in a source ontology, the nonduplicate-based method is used to find

correspondences between different types of entities, including classes, object properties, datatype properties, and data values. Each correspondence is then automatically translated to a SPARQL query as an executable mapping.

Both executable mappings in Figure 6.4 can be generated by this system. The mapping in Figure 6.4a expresses that the target property "price" is matched to property "price" of class "IBM" if the target property "code" has a value of "IBM". This example demonstrates the importance of dependency. If the target property "code" has "AAPL" as value, then the target property "price" should be mapped to property "price" of class "AAPL".

The second system is QODI, detailed in Chapter 5. Given a SPARQL query, QODI is designed to generate mappings specific to that query to resolve ambiguity. The choice of queries passed to QODI is the place where prior knowledge can be used. The queries that are frequently issued before are more likely to be seen in future. Thus, QODI can generate mappings for those frequent queries. Developers can also manually pick candidate queries as inputs of QODI. If there is no prior knowledge, QODI will generate a mapping for each query consisting of one triple in the target ontology by default. QODI had an assumption that only paths with datatype properties are considered. We extend QODI to consider all paths in Alamo.

All mappings in Figure 6.3 can be generated by QODI. Especially, the mapping in Figure 6.3c is between an entity in the target ontology and a set of entities in the source ontology, which corresponds to a sub-ontology.

| Group uri | Source 1 uri | Source 2 uri |
|---|---|---|
| target:group1 | src1:stk1 | IBM:ibm1 |

Figure 6.5: ENS between source 1 and source 2.

### 6.1.2 Entity Name Service

Aiming at integrating multiple data sources, a research question is how to combine query results given that instances from different data sources usually have different uri domains. As stated by Bouquet et. al., recognizing that information from different sources refers to the same real world entity is a prerequisite for combining information [19]. Alamo relies on the concept of entity name service (ENS) proposed by Bouquet et. al. [19]. The idea behind the project is to enable the systematic reuse of global identifiers for entities that are described in heterogeneous data collections on the web.

The objective of the ENS in Alamo is to group equivalent instances from individual data sources, and assign a target uri to each group. Although the instances in the same group are from different data sources, Alamo will join them to combine the query results. The details of ENS are beyond the scope of this dissertation.

Figure 6.5 shows the ENS generated based on the RDF triples of data sources. The instance "src1:stk1" from source 1 and instance "IBM:ibm1" from source 2 are grouped together, because they contain information of IBM stocks of the same date "1/3/2012".

## 6.2 Run Time

At run time, Alamo takes an input of a query in terms of the target ontology from the application, and returns the combined query results from all data sources as well as provenance.

The target query is passed to query federator first, which determines the sub-queries that will be reformulated in terms of individual data sources. Each data source is associated with a reformulator. Although the target query can be in any form (select, construct, etc.), a reformulator reformulates the query as a construct query, whose head contains triples in terms of the target ontology and body contains triples in terms of the data source. Executing these construct queries results in triples. Thus, combining the query results from different data sources is simply combining all triples together. The combined query results are processed by URI normalizer, which substitutes any source instance uri by group uri in ENS such that instances from different data sources can be joined together. Finally, result combiner executes the original target query over the combined triples, and returns the query results.

Figure 6.6 shows a target SPARQL query that asks the price and trading volume of stock "IBM" on "1/3/2012". We will show how Alamo executes this target query using the two data sources based on the executable mappings in Figure 6.3 and 6.4.

```
SELECT ?p ?v
WHERE {
 ?x target:code target:IBM .
 ?x target:date "1/3/2012".
 ?x target:price ?p .
 ?x target:volume ?v .
}
```

Figure 6.6: SPARQL query asking the price and trading volume of stock "IBM" on "1/3/2012".

### 6.2.1 Query Reformulation

The task of a reformulator is to translate the target query in terms of a data source based on executable mappings. The reformulated queries are always construct queries, no matter the query form in the target query. Since the results of construct queries are triples, combining the query results from different data sources is simply combining all triples together. The query reformulation process is defined as follows:

**Definition 6.2.1** (query reformulation)**.** Given a SPARQL query in terms of the target ontology $q_t$ and a data source $S$, query reformulation is a process to return a SPARQL query $q_s$, such that:

- $q_s$ is a construct query;
- the head of $q_s$ is the same as the body of $q_t$;
- $\exists\ q_e$ in terms of source $S$ and $q_e$ is equivalent to $q_t$, s.t. the body of $q_s$ is the same as the body of $q_e$.

```
CONSTRUCT {
 ?x target:code target:IBM.
 ?x target:date "1/3/2012".
 ?x target:volume ?v.
} WHERE {
 ?x src1:company src1:IBM.
 ?x src1:close_date ?y.
 ?y src1:date "1/3/2012".
 ?x src1:volume ?v.
}
```

(a) In terms of source 1

```
CONSTRUCT {
 ?x target:code target:IBM.
 ?x target:date "1/3/2012".
 ?x target:price ?p.
} WHERE {
 ?x IBM:date "1/3/2012".
 ?x IBM:price ?p.
}
```

(b) In terms of source 2

Figure 6.7: Reformulated SPARQL queries in terms of data sources.

There are two steps to automatically generate the reformulated queries in Definition 6.2.1. First, find a source query that is equivalent to the target query. Second, change the equivalent source query as a construct query.

To perform the first step, a naive algorithm is to consider all possible combinations of executable mappings, and output the union of valid ones. The complexity of this algorithm is too high to be used in a real system. We use the query rewriting algorithm proposed by Le et. al. [55]. This algorithm is similar to the bucket algorithm in relational database [57].

The idea of the query reformulation algorithm is to create a bucket for each property existed in the target query, and add all executable mappings containing that property in the bucket [55]. The reformulated query is the union of all valid Cartesian product of the mappings in the buckets with variable substitution. The reformulated query is proved to be equivalent as the target query.

| Subject | Predicate | Object |
|---|---|---|
| src1:stk1 | target:code | target:IBM |
| src1:stk1 | target:date | “1/3/2012” |
| src1:stk1 | target:volume | 5.6m |

(a) Results from source 1

| Subject | Predicate | Object |
|---|---|---|
| IBM:ibm1 | target:code | target:IBM |
| IBM:ibm1 | target:date | “1/3/2012” |
| IBM:ibm1 | target:price | 186.3 |

(b) Results from source 2

Figure 6.8: Executed results of the reformulated queries.

| Subject | Predicate | Object |
|---|---|---|
| target:group1 | target:code | target:IBM |
| target:group1 | target:date | “1/3/2012” |
| target:group1 | target:volume | 5.6m |
| target:group1 | target:price | 186.3 |

Figure 6.9: Normalized query results.

After obtaining the equivalent source query, the second step syntactically combines the target query and the equivalent source query to generate a construct query as the final reformulated query. The head of the construct query consists of all triples in the target query, and the body consists of all triples in the equivalent source query.

Figure 6.7 shows the reformulated queries in terms of data source 1 and data source 2 based on the executable mappings in Figure 6.3 and 6.4.

### 6.2.2 Query Results Combining

Each reformulated query is executed on its corresponding data source. Since the reformulated queries are construct queries, the results are RDF triples as shown in Figure 6.8. The instance uris in the results are from individual data sources, such as “src1:stk1” and “IBM:ibm1”, although they refer to the same real world object.

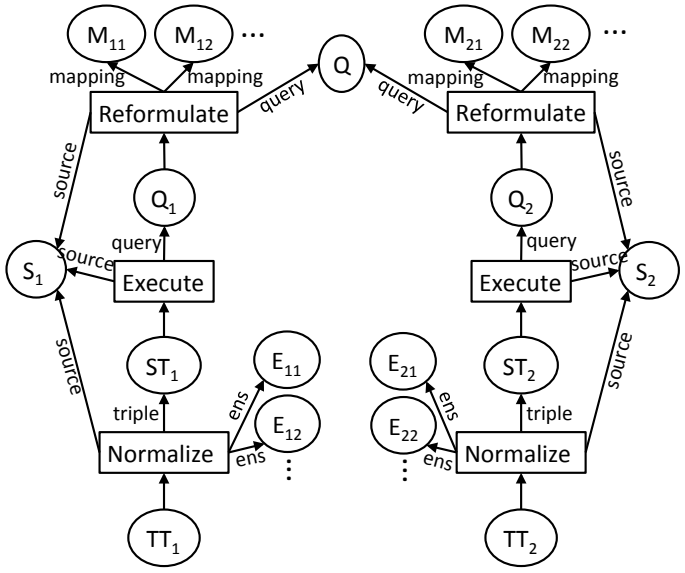



Figure 6.10: An example provenance produced by Alamo. Artifacts are represented by ellipses, and processes are represented by rectangles.

URI normalizer substitutes the data source uris of the RDF triples based on the ENS generated in compile time. After the normalization, instances can be joined by the group uris. The normalized RDF triples based on the ENS in Figure 6.5 is shown in Figure 6.9. These RDF triples do not contain any uri from data sources.

Result combiner finally executes the original target query on the normalized RDF triples and returns the results. In the example, the query in Figure 6.6 is executed based on the triples in Figure 6.9. The results of the query assign "186.3" to $?p$ and "5.6m" to $?v$.

### 6.2.3 Provenance

Along with the query results, Alamo also returns the provenance to facilitate the users to inspect the results. The provenance is represented using the Open Provenance Model (OPM) [67]. A provenance is a directed labeled graph that can be serialized to OWL, XML, etc. A vertex can be an *artifact* that models an immutable piece of state, a *process* that models an action, or an *agent* that models context enabling or facilitating the execution of a process. Vertices are distinguished by unique uris. Edges can be labeled as different *roles*. Both vertices and edges can have *annotations* to store information. Figure 6.10 shows an example provenance produced in Alamo considering two data sources. The vertices $S$ represent data sources, and $Q$ represent queries. $M$ and $E$ represent executable mappings and entries in ENS, respectively. $ST$ and $TT$ represent RDF triples. The actual values, such as mappings, queries, triples, etc., are stored as annotations of corresponding vertices.

## 6.3 Experiments

Comparing to the conventional mapping process, the objective of the experiments is to assess the impacts of our generalizations of mapping problems to the entire data integration system. The evaluation focuses on the accuracy of reformulated queries instead of executable mappings or query execution results for three reasons. (1) The accuracy of reformulated queries measures the performance of both executable mapping generator and query reformulator. These two components are fully automatic, and contain AI algo-

rithms. They are usually the main sources of errors. (2) It is difficult to fairly compare executable mapping accuracy, since different systems may generate a different number of mappings, and there is no unique groundtruth mapping (for example, the combination of two correct mappings is also correct). (3) Reformulated queries are usually not affected by instances. Thus, the evaluation does not depend on underlying instances, and in fact it does not require them.

### 6.3.1 Test sets

The evaluation is conducted on both test suites discussed in Chapter 3.

The first one is detailed in Section 3.2. It comprises three application domains: Bibliography, Conference Organization, and Life Science, and does not contain data instances. In the Bibliography test set, UMBC is the target ontology, and DBLP is the source ontology. In the Conference test set, SOFSEM is the target ontology, and SIGKDD is the source ontology. In the Life Science test set, Darwin Core is the target ontology, and Specify is the source ontology.

The advantage to use these test sets is that each ontology is associated with sets of SPARQL queries. Specifically, each ontology is associated with two kinds of SPARQL queries. 1) A *PathOnly* query has a query graph consisting of only one path. In English specification, a PathOnly query asks for all values of a single attribute of a concept. 2) A *ClassAll* query has a query graph consisting of all paths (at least two) that share a source. In English

specification, a ClassAll query asks for all values of all attributes of a concept. It is the most complicated query with one conjunction over the source. ClassAll queries are more difficult to be reformulated than PathOnly queries, because ClassAll queries contain many triples. If one triple is reformulated incorrectly, the whole reformulation will be wrong.

The second one is detailed in Section 3.1. We automatically translate the relational databases into ontologies and RDF files through Ultrawrap [81]. There are four application domains: Stock Market, Ecommerce, College Enrollment, and Video Game. In the Stock test set, Chwab is the target ontology, and Euter is the source ontology. In the Ecommerce test set, Subrion is the target ontology, and Opencart is the source ontology. In the Enrollment test set, Statistics is the target ontology, and Ranking is the source ontology. In the Game test set, Vgchartz is the target ontology, and Dbpedia is the source ontology.

We follow the same method to generate the PathOnly and ClassAll query sets for each test domain. In addition to these queries, each test domain is associated with RDF files as data instances. Thus, mappings between ontologies and data instances can be generated and used for query reformulation.

### 6.3.2 Baselines

The baseline implements the conventional mapping process, and uses the same query reformulation algorithm as in Alamo. An ontology matcher is used to generate correspondences between the same type of ontology entities,

and the executable mappings are generated using a recent method proposed by Rivero et. al. [78]. The executable mapping generator proposed by Rivero et. al. takes a set of entity correspondences and a set of restrictions of each ontology as inputs, and outputs a set of executable mappings as construct SPARQL queries [78]. Specifically, for each entity correspondence, the algorithm expands each entity to a graph based on restrictions, and group all correspondences within the graphs as a kernel. Finally, each kernel is transformed as a construct SPARQL query. The restrictions considered in the baseline are domain and range. The baseline is denoted as EntityKernel.

Both the baseline and Alamo require an ontology matcher to compute the similarity between entities. We use SMOA string similarity as the matcher for all methods [83].

### 6.3.3 Results on the Test Sets without Instances

The threshold to determine predicted correspondences has a large impact to all methods (predicted correspondences must have higher confidence than the threshold). We vary the threshold as 0.3, 0.5, and 0.7 to give a comprehensive comparison.

Table 6.1, 6.2, and 6.3 show the number of correctly reformulated queries versus the number of total queries given various thresholds. The number of correctly reformulated queries is the main measure to compare different data integration systems in the experiments. Table 6.4, 6.5, and 6.6 show the number of valid reformulation regardless of correctness. The number of valid

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 12 / 62 | 15 / 62 | 18 / 62 |
| Alamo | **39** / 62 | **39** / 62 | **33** / 62 |
| ClassAll query set | | | |
| EntityKernel | 1 / 8 | 1 / 8 | 0 / 8 |
| Alamo | 1 / 8 | 1 / 8 | 0 / 8 |

Table 6.1: Query reformulation correctness of the Bibliography test set. A cell is in the form x / y, where x is the number of correctly reformulated queries, and y is the number of total input queries. The three columns show the results when the correspondence threshold is 0.3, 0.5, and 0.7. EntityKernel is the baseline, and Alamo is the proposed system. PathOnly and ClassAll are two query sets.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 4 / 12 | 4 / 12 | 0 / 12 |
| Alamo | **5** / 12 | **7** / 12 | 0 / 12 |
| ClassAll query set | | | |
| EntityKernel | 1 / 3 | 1 / 3 | 0 / 3 |
| Alamo | 1 / 3 | 1 / 3 | 0 / 3 |

Table 6.2: Query reformulation correctness of the Conference test set. See Table 6.1 for explanations.

reformulation indicates the coverage of different data integration systems.

The proposed system Alamo dominates the baseline in all test sets with PathOnly queries. This is especially demonstrated in the Bibliography set (Table 6.1), and the Life Science set (Table 6.3). In the Bibliography set, Alamo correctly reformulates 33 to 39 queries, and the baseline correctly reformulates at most 18 queries. In the Life Science set, the number of correctly reformulated queries by Alamo is between 19 and 30, and the number

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 0 / 36 | 0 / 36 | 7 / 36 |
| Alamo | **30** / 36 | **25** / 36 | **19** / 36 |
| ClassAll query set | | | |
| EntityKernel | 0 / 3 | 0 / 3 | 0 / 3 |
| Alamo | 0 / 3 | 0 / 3 | 0 / 3 |

Table 6.3: Query reformulation correctness of the Life Science test set. See Table 6.1 for explanations.

is at most 7 by the baseline. The big gap between Alamo and the baseline emphasizes the difference of the underlying mapping generator. The baseline first generates entity correspondences, and transforms them to executable mappings. It is limited by the expressiveness and correctness of entity correspondences. In the test sets, one triple may be mapped to multiple triples, which cannot be expressed by entity correspondences. Alamo uses two mapping generators, including QODI. QODI is capable to find correspondences between sub-ontologies; thus, can resolve this issue.

Alamo performs similarly to the baseline on ClassAll queries. The performance of both systems on ClassAll queries is worse than that on PathOnly queries. A ClassAll query is more complex than a PathOnly query. For example, one ClassAll query of the Life Science test set contains 44 triples. If one triple is not reformulated correctly, the whole reformulation is wrong. We manually checked the reformulated queries. Wrong reformulation usually only involves one or two triples that are incorrectly mapped. With a small amount of human labors to correct mappings, the correctness of ClassAll query refor-

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 28 / 62 | 28 / 62 | 24 / 62 |
| Alamo | **42** / 62 | **42** / 62 | **42** / 62 |
| ClassAll query set | | | |
| EntityKernel | 1 / 8 | 1 / 8 | 0 / 8 |
| Alamo | 1 / 8 | 1 / 8 | 0 / 8 |

Table 6.4: Number of valid query reformulation of the Bibliography test set. A cell is in the form x / y, where x is the number of valid reformulated queries regardless of correctness, and y is the number of total input queries. The three columns show the results when the correspondence threshold is 0.3, 0.5, and 0.7. EntityKernel is the baseline, and Alamo is the proposed system. PathOnly and ClassAll are two query sets.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 5 / 12 | 5 / 12 | 0 / 12 |
| Alamo | **12** / 12 | **10** / 12 | **1** / 12 |
| ClassAll query set | | | |
| EntityKernel | 1 / 3 | 1 / 3 | 0 / 3 |
| Alamo | **2** / 3 | 1 / 3 | 0 / 3 |

Table 6.5: Number of valid query reformulation of the Conference test set. See Table 6.4 for explanations.

mulation will be drastically improved.

The correspondence threshold has different impacts on Alamo and the baseline. For Alamo, a smaller threshold usually achieves higher reformulation correctness. This is because one of Alamo's mapping generator, QODI, consists of an optimization over sub-ontologies. A smaller threshold provides more correspondences to be considered in the optimization. Thus, the mapping is highly likely to be better. For the baseline, the threshold determines the entity

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 36 / 36 | 15 / 36 | 13 / 36 |
| Alamo | 36 / 36 | **36** / 36 | **28** / 36 |
| ClassAll query set | | | |
| EntityKernel | 2 / 3 | 0 / 3 | 0 / 3 |
| Alamo | 2 / 3 | **2** / 3 | 0 / 3 |

Table 6.6: Number of valid query reformulation of the Life Science test set. See Table 6.4 for explanations.

correspondences as well as the number of executable mappings, because one entity correspondence is transformed as one executable mapping. A smaller threshold increases the coverage of the mappings, but decreases the accuracy. Thus, the reformulated queries tend to have more unrelated triples as union. A larger threshold produces fewer mappings. Thus, the reformulated queries may only cover part of the original queries. The choice of the threshold is a tradeoff for the baseline that has to be tuned based on individual problems.

As shown in Table 6.4, 6.5, and 6.6, the number of valid reformulated queries generated by Alamo is more than or equal to the number by the baseline. Similarly to query reformulation correctness, Alamo dominates the baseline on PathOnly queries, and the difference is small on ClassAll queries. The correspondence threshold has a big impact. Smaller threshold generates more entity correspondences and more executable mappings. Thus, the number of valid query reformulation will be larger.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 1 / 4 | 1 / 4 | 1 / 4 |
| Alamo | **4** / 4 | **4** / 4 | **4** / 4 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | **1** / 1 | **1** / 1 | **1** / 1 |

Table 6.7: Query reformulation correctness of the Stock test set. See Table 6.1 for explanations.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 0 / 6 | 0 / 6 | 0 / 6 |
| Alamo | **3** / 6 | **2** / 6 | **2** / 6 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | 0 / 1 | 0 / 1 | 0 / 1 |

Table 6.8: Query reformulation correctness of the Ecommerce test set. See Table 6.1 for explanations.

### 6.3.4 Results on the Test Sets with Instances

Table 6.7, 6.8, 6.9, and 6.10 show the number of correctly reformulated queries versus the number of total queries given various thresholds. Table 6.11, 6.12, 6.13, and 6.14 show the number of valid reformulation regardless of correctness.

Similar to the results on the previous test sets, Alamo usually achieves higher accuracy than the baseline on PathOnly queries. Particularly in the Stock test set, Alamo correctly reformulates all four PathOnly queries, and the baseline only reformulates one. These test sets are difficult, because they

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 0 / 4 | 0 / 4 | 0 / 4 |
| Alamo | 0 / 4 | **1** / 4 | **1** / 4 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | 0 / 1 | 0 / 1 | 0 / 1 |

Table 6.9: Query reformulation correctness of the Enrollment test set. See Table 6.1 for explanations.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 0 / 4 | 0 / 4 | 0 / 4 |
| Alamo | **3** / 4 | **2** / 4 | **1** / 4 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | 0 / 1 | 0 / 1 | 0 / 1 |

Table 6.10: Query reformulation correctness of the Game test set. See Table 6.1 for explanations.

require mappings between ontologies and data instances. One of Alamo's mapping generator is capable to find this kind of mappings. The baseline generates executable mappings based on correspondences between ontology entities, thus it cannot find mappings between ontologies and data instances.

Neither system achieves high accuracy on ClassAll query sets. Alamo is better than the baseline in the Stock test set, and performs the same in the other test sets. As explained before, ClassAll queries are difficult to reformulate since each query contains many triples. If one triple is incorrectly reformulated, the whole query is failed.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 1 / 4 | 1 / 4 | 1 / 4 |
| Alamo | **4** / 4 | **4** / 4 | **4** / 4 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | **1** / 1 | **1** / 1 | **1** / 1 |

Table 6.11: Number of valid query reformulation of the Stock test set. See Table 6.4 for explanations.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 0 / 6 | 0 / 6 | 0 / 6 |
| Alamo | **6** / 6 | **5** / 6 | **2** / 6 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | **1** / 1 | 0 / 1 | 0 / 1 |

Table 6.12: Number of valid query reformulation of the Ecommerce test set. See Table 6.4 for explanations.

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 0 / 4 | 0 / 4 | 0 / 4 |
| Alamo | **1** / 4 | **1** / 4 | **1** / 4 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | 0 / 1 | 0 / 1 | 0 / 1 |

Table 6.13: Number of valid query reformulation of the Enrollment test set. See Table 6.4 for explanations.

In terms of the number of valid query reformulation, Alamo dominates the baseline in all test sets on PathOnly queries, and slightly better on ClassAll

| | 0.3 | 0.5 | 0.7 |
|---|---|---|---|
| PathOnly query set | | | |
| EntityKernel | 0 / 4 | 0 / 4 | 0 / 4 |
| Alamo | **4** / 4 | **2** / 4 | **1** / 4 |
| ClassAll query set | | | |
| EntityKernel | 0 / 1 | 0 / 1 | 0 / 1 |
| Alamo | **1** / 1 | 0 / 1 | 0 / 1 |

Table 6.14: Number of valid query reformulation of the Game test set. See Table 6.4 for explanations.

queries. In the Stock test set, Alamo reformulates all PathOnly and ClassAll queries, and the baseline only reformulates one PathOnly query. In the Ecommerce test set, Alamo reformulates all PathOnly queries given 0.3 as threshold, and the baseline never reformulates any query. The big gap between Alamo and the baseline demonstrates the effectiveness of the two mapping generators used in Alamo.

## 6.4 Related Work

In the Semantic Web, Karma is a complete data exchange system that semi-automatically transfers structured data sources into the Semantic Web [51]. Karma uses a similar logic representation as TGD to represent executable mappings. The mapping process consists of three steps. First, find correspondences between columns in the source and classes and datatype properties in the target using CRF [43]. Second, construct a graph based on the target ontology and the correspondences, and extract the minimal tree by a variation of the Steiner Tree algorithm [52]. Finally, the executable mappings

are generated based on the minimal tree. There are mainly three differences between Karma and Alamo. The task of Karma is to transfer structured data sources into RDF, while Alamo is designed to answer queries using multiple structured data sources. The inputs to Karma are the target ontology and source databases (csv, xml, etc.), while the inputs to Alamo are all ontologies and data sources in the Semantic Web. To use databases in Alamo, Ultrawrap is used to automatically translate them to the Semantic Web [81]. The mapping process in Karma is semi-automatic, since CRF needs human labels for training. The two mapping generators in Alamo are automatic.

## 6.5 Discussion and Future Work

This work details the architecture of a complete ontology-based data integration system, Alamo. Alamo uses construct SPARQL queries as the representation of executable mappings. Any ontology mapper that can produce mappings in this representation can be integrated in Alamo. Given a user query, the query reformulation algorithm can produce equivalent queries as a union of triples based on executable mappings. To combine the query results from individual data sources, Alamo uses its ENS to join multiple instances.

Alamo implements the two mapping systems proposed in this dissertation to automatically generate executable mappings. To support our thesis that automatic data integration systems are limited by a narrow definition of mapping, we conduct experiments on seven application domains to compare Alamo to the baseline that implements the conventional mapping definition.

The best run of Alamo correctly reformulates 84 PathOnly queries out of 128 in total, and the best run of the baseline only correctly reformulates 26. The number of queries correctly reformulated by Alamo is three times more than that by the baseline. This big gap between the two methods demonstrates the importance of generalizing the conventional mapping definition. Although better than the baseline, the two mapping systems may not be enough to cover all real use cases. The architecture of Alamo enables the integration of additional mapping systems that produce construct SPARQL queries as executable mappings.

One drawback of the evaluation is the lack of real user queries in the test sets. Instead of creating random queries to mimic real queries, we choose to systematically and consistently create queries based on the criterion of number of paths. A PathOnly query has a query graph consisting of only one path. A ClassAll query has a query graph consisting of all paths (at least two) that share a source. PathOnly queries are the simplest queries. ClassAll queries are the most complicated queries with one conjunction over the source. These two kinds of queries can be deemed as the lower bound and upper bound of real user queries. A real user query can be seen as the combination of multiple PathOnly queries. Thus, the accuracy of query reformulation of PathOnly queries can be seen as the probability of the success of real user queries. In most of the cases, a ClassAll query is more complicated than a real user query. We manually inspect the results. Failure cases of ClassAll queries usually involve a small number of incorrect mappings. Take the Bibliography test set for example.

All methods fail to generate the mapping between the property "name" of the class "Person" in the target ontology and the property "author" of the class "Author" in the source ontology. That single incorrect mapping appears in all seven failure cases out of eight queries in total. Manually correction of this mapping will drastically improve the accuracy of ClassAll queries.

Future work of Alamo consists of at least two possible directions. First, the mapping generator is fully automatic in current implementation. A graphical interface that facilitates users to manually adjust incorrect mappings can drastically improve the accuracy. Second, query federator is an interesting research topic. A promising direction is to determine the sub-query passed to one data source based on the mappings and ENS of all data sources. Given the information from all data sources, the federator can make an optimal decision.

# Chapter 7

# Future Work

In this chapter, I discuss the research directions for the ontology mapping area in general. The future work specific to the proposed systems in this dissertation is discussed at the end of each chapter.

## 7.1 Generalized Mapping Definition

This dissertation consists of two work to generalize the common definition of ontology mapping. The first work can express the mappings between ontology entities and data instances. The second work can express the mappings between sub-ontologies. Both work still have some limitations in expressiveness. In the first work, the mappings between ontology entities and data instances have to be within the same class. In the second work, the mappings are generated only for paths in the ontologies.

A more general definition should allow to express the mappings between any sub-ontology and any data instance. The Semantic Web data model provides a starting point for this general definition. In the Semantic Web, both ontologies and data instances are represented as directed labeled graphs, and these two kinds of graphs are connected as one graph by a special edge

"rdf:type". Thus, a data source is essentially a large graph that contains both ontologies and instances. The mapping can be defined as correspondences between a target subgraph and a subgraph in the source in general. This definition subsumes both generalized definitions proposed in this dissertation.

Automatically generating mappings in this general definition is challenging. There are many potential problems to resolve. First, how to ensure that the generated mappings have valid semantics, especially for the mappings related to both ontologies and instances. Second, more general definitions lead to larger pool of possible solutions. Achieving reasonable accuracy is more difficult. Third, as the graph is very large, the computation complexity will be very high. Efficient graph matching algorithms have to be incorporated.

## 7.2 Human Involvement

Automatic mapping systems usually cannot produce perfect results. People should be involved for the applications that require high accuracy. Research in this direction should consider three questions.

**How to interact.** A straightforward way of involving people in data integration systems is to ask them to provide mappings or label the correctness of mappings. These labeled mappings have been used to train supervised machine learning models to predict more mappings [28]. However, existing machine learning based methods mostly only generate correspondences between entities. An interesting research direction is to apply these learning

algorithms to identify executable mappings. This comprises the tasks of formulating the executable mapping generation as a classification problem, and identifying meaningful features (e.g. structures of graphs). Active learning is known to be effective in reducing the number of labels [10, 86]. To apply active learning on ontology mapping problems, the research question is how to automatically pick candidate mappings to label. A possible solution is to pick the most uncertain mapping (the one with the lowest probability). An advanced solution should ask the label of the mapping that can contribute the largest gain of accuracy of the whole system.

**What to display.** People are usually asked to label mappings only based on ontologies. This kind of interface is suitable for experts, not general users. A future direction is to display richer information. Example information can be consequences of labels, including what are the reformulations of frequent input queries given the current labeled mappings, what are the executing results of those queries, etc. Sometimes, users need external information to understand some ontology entities. That information can be retrieved by searching the entities in public search engines, linking information from known websites (such as DBpedia), or finding similar entities in other ontologies. Given this information, users can have a better understanding of whether the labels are correct.

**Who to ask.** People with different expertise should all be able to participate in the mapping process. Experts can provide labels at any mapping stage.

Users can provide mapping labels in pay-as-you-go fashion, when they use data integration systems [25]. Recently, crowdsourcing has been used to annotate ontology mappings to reduce labor cost [63, 79, 91]. This approach has some problems. One problem is how to teach ordinary people to understand ontology mappings and annotate them correctly. Existing methods only consider entity correspondences, and only ask simple binary labels. Even for this basic task, the labels from the crowd are too noisy to use directly. Majority voting is commonly used to reduce the noise. Advanced techniques should be explored.

## 7.3 Similarity Measure

Similarity measure is a fundamental requirement of all ontology mapping systems. String similarity measure has been thoroughly studied. There are many string distances that can be used in different applications [21]. In addition to strings, data sources usually contain many other data types, such as number, date, boolean, category, etc. Chapter 4 implements similarity measures for number and date. A comprehensive study of similarity measures for all data types will be valuable to ontology mapping systems.

In addition to syntactic similarity measures, machine learning algorithms can be applied to this task [15]. These algorithms are usually supervised, and require labels. Unsupervised algorithms are promising, and should be explored. Natural language processing (NLP) algorithms can be applied to ontology mapping. These algorithms are especially important to the task of mapping multiple attributes that require concatenation (e.g. name to first

name and last name). Databases usually contain computer-generated strings. Stemming techniques specific for computer-generated strings are necessary for mapping.

Similarity measure is not restricted to single data value. As the mapping definition is generalized, similarity between structures is needed. For example, QODI exploits similarity measures of paths. A more general question is how to measure the similarity between graphs. The research of graph matching should be introduced to the ontology mapping community.

# Chapter 8

# Conclusion

This dissertation starts from a thesis that the progress on automatic data integration has been limited by a narrow definition of mapping. The common mapping process is to find correspondences between pairs of entities in data models, and create logic expressions over the correspondences as executable mappings. This does not cover all issues in real world applications.

My first results are to collect real world examples and show that they support my thesis. With the help of the whole research group, we collect two test suites. One suite includes relational databases in four application domains: ecommerce, stock price, college enrollment, and video game. This test set demonstrates that mappings can be between different types of database elements, including relation, attribute, and data value. The other test suite consists of the ontologies in three application domains: life science, bibliography, and conference organization. This set demonstrates that mappings are sometimes ambiguous, and defining mappings as pairs of entities is not enough. We also systematically generate queries for each ontology, such that the test set can be used as a benchmark for data integration systems instead of merely for ontology mapping.

Carefully observing these real world examples, I have intuitions of how to formulate the mapping definitions. I propose two generalizations in this dissertation. First, I introduce compound correspondences, which can specify correspondences between any type of database elements, consisting of relation, attribute, and data value. Most importantly, I provide semantics to these compound correspondences, and demonstrate that these correspondences can be translated to tuple-generating dependencies (TGDs). Thus, they can be used in any data integration system that accepts TGDs. The second generalization is for data sources in the Semantic Web. I introduce the query-specific ontology mapping, which takes a user query as input in addition to ontologies. User queries provide more context information, which can be used to disambiguate mappings. The mappings are defined as correspondences between pairs of graph paths (sequences of entities). By this definition, query reformulation is simplified.

Automatically generating mappings for the two generalized definitions is more difficult than that for the common ontology matching problem. I propose two methods to generate mappings for the first definition. Both methods estimate the probabilities of mapping candidates. One is a general instance-based method, and the other one requires a few duplicate instances. For the second generalized mapping definition, I propose the mapping system QODI. QODI is distinguished in that the ontology mapping algorithm dynamically determines a partial mapping specific to the reformulation of each query. Given an input query, QODI decomposes the query into a set of paths, and searches

for a subgraph of the source ontology, such that the set of path correspondences has the highest confidence. QODI exploits heuristic search algorithms, which guarantee to find an optimal solution. Both mapping systems are evaluated on our test suites, and achieve favorable results over state-of-the-art baselines.

My last piece of work is motivated by two reasons. (1) In the Semantic Web community, most research focus on sub-problems of data integration. Complete data integration systems have not been well studied. (2) To test the thesis, I want to integrate the two proposed mapping systems together, and evaluate their performance in a real complete data integration system. My last piece of work contributes to the ontology mapping and query reformulation components in a data integration system, Alamo. I define executable mappings, and use construct SPARQL queries as the physical representation. User queries can be automatically reformulated to different data sources based on the executable mappings.

Alamo implements the two mapping systems proposed in this dissertation. I conduct experiments on seven application domains to compare Alamo to the baseline that implements the conventional mapping definition. The best run of Alamo correctly reformulates 84 PathOnly queries out of 128 in total, and the best run of the baseline only correctly reformulates 26. This big gap between the two methods demonstrates the importance to generalize the conventional mapping definition, and supports my thesis.

# Appendix

# Appendix 1

# Datalog Rules of Direct Mapping

Sequeda, Arenas, and Miranker detail the datalog rules of mapping relational databases to ontologies and RDF instances [80]. The rules that translate relational schemas to ontologies are summarized as follows:

**Classes** A class is any relation that is not a binary relation.

$$Class(X) \leftarrow Rel(X), \neg IsBinRel(X)$$

where $Class(X)$, $Rel(X)$, and $IsBinRel(X)$ indicate that $X$ is a class, relation, and binary relation, respectively. An RDF triple is generated for each class.

$$Triple(U, \text{“rdf:type”}, \text{“owl:class”}) \leftarrow Class(X), ClassIRI(X, U)$$

where $ClassIRI(X, U)$ generates uri $U$ for relation $X$, and $Triple$ is used to collect all the triples of the RDF graph generated by the direct mapping.

**Object Properties** An object property is generated either from a binary relation ($OP_1$) or a foreign key ($OP_2$).

$$OP_1(X, D, R) \leftarrow BinRel(X, D, R)$$

$$OP_2(X, D, R) \leftarrow FK(X, D, R), \neg IsBinRel(D)$$

where $OP(X, D, R)$ represents object property $X$ with domain $D$ and range $R$, $BinRel(X, D, R)$ indicates $X$ is a binary relation referencing relations $D$ and $R$, and $FK(X, D, R)$ represents $X$ is a foreign key in relation $D$ referencing relation $R$. The following RDF triples are generated:

$$Triple(U, \text{“rdf:type”}, \text{“owl:ObjectProperty”})$$
$$\leftarrow OP(X, D, R), OP\_IRI(X, D, R, U)$$
$$Triple(U, \text{“rdfs:domain”}, W)$$
$$\leftarrow OP(X, D, R), OP\_IRI(X, D, R, U), ClassIRI(D, W)$$
$$Triple(U, \text{“rdfs:range”}, W)$$
$$\leftarrow OP(X, D, R), OP\_IRI(X, D, R, U), ClassIRI(R, W)$$

where $OP\_IRI(X, D, R, U)$ generates uri $U$ for object property $X$.

**Datatype Properties** A datatype property is an attribute in non-binary relations.

$$DP(X, D) \leftarrow Attr(X, D), \neg IsBinRel(D)$$

where $DP(X, D)$ indicates $X$ is a datatype property with domain $D$, and $Attr(X, D)$ indicates $X$ is an attribute in relation $D$. The following RDF

triples are generated:

$$Triple(U, \text{“rdf:type”}, \text{“owl:DatatypeProperty”})$$
$$\leftarrow DP(X, D), DP\_IRI(X, D, U)$$
$$Triple(U, \text{“rdfs:domain”}, W)$$
$$\leftarrow DP(X, D), DP\_IRI(X, D, U), ClassIRI(D, W)$$

where $DP\_IRI(X, D, U)$ generates uri $U$ for datatype property $X$.

# Bibliography


[1] Direct Mapping. http://www.w3.org/TR/rdb-direct-mapping/.

[2] OAEI. http://oaei.ontologymatching.org/.

[3] OWL. http://www.w3.org/TR/owl-features/.

[4] RDF. http://www.w3.org/TR/PR-rdf-syntax/.

[5] RDFS. http://www.w3.org/TR/rdf-schema/.

[6] SPARQL Language. http://www.w3.org/TR/rdf-sparql-query/.

[7] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[8] Bogdan Alexe, Laura Chiticariu, Renee J Miller, and Wang-Chiew Tan. Muse: Mapping understanding and design by example. In *IEEE 24th International Conference on Data Engineering*, pages 10–19. IEEE, 2008.

[9] Marcelo Arenas, Pablo Barcelo, Leonid Libkin, and Filip Murlak. Relational and xml data exchange. *Synthesis Lectures on Data Management*, 2(1):1–112, 2010.

[10] F. Barbançon and D.P. Miranker. Sphinx: Schema integration by example. *Journal of Intelligent Information Systems*, 29(2):145–184, 2007.

[11] Catriel Beeri and Moshe Y Vardi. A proof procedure for data dependencies. *Journal of the ACM (JACM)*, 31(4):718–741, 1984.

[12] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm, editors. *Schema Matching and Mapping*. Springer, 2011.

[13] P.A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11), 2011.

[14] Philip A Bernstein and Laura M Haas. Information integration in the enterprise. *Communications of the ACM*, 51(9):72–79, 2008.

[15] Mikhail Bilenko and Raymond J Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[16] A. Bilke and F. Naumann. Schema matching using duplicates. In *21st International Conference on Data Engineering*, pages 69–80. IEEE, 2005.

[17] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[18] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting context into schema matching. In *Proceedings of the 32nd international conference on Very large data bases*, pages 307–318. VLDB Endowment, 2006.

[19] Paolo Bouquet, Heiko Stoermer, Claudia Niederee, and A Maa. Entity name system: The back-bone of an open and scalable web of data. In *Semantic Computing, 2008 IEEE International Conference on*, pages 554–561. IEEE, 2008.

[20] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, (99):1–1, 2011.

[21] WW Cohen, P. Ravikumar, and SE Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI2003 Workshop on Information Integration on the Web IIWeb03*, pages 73–78, 2003.

[22] G. Correndo, M. Salvadores, I. Millard, H. Glaser, and N. Shadbolt. Sparql query rewriting for implementing data integration over linked data. In *Proceedings of EDBT*, page 4. ACM, 2010.

[23] I.F. Cruz, F.P. Antonelli, and C. Stroe. Agreementmaker: efficient matching for large real-world schemas and ontologies. *Proceedings of the VLDB Endowment*, 2(2):1586–1589, 2009.

[24] Bing Tian Dai, Nick Koudas, Divesh Srivastava, Anthony KH Tung, and Suresh Venkatasubramanian. Validating multi-column schema matchings by type. In *IEEE 24th International Conference on Data Engineering*, pages 120–129. IEEE, 2008.

[25] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD 2008*, pages 861–874. ACM, 2008.

[26] R. Dhamankar, Y. Lee, A.H. Doan, A. Halevy, and P. Domingos. imap: discovering complex semantic matches between database schemas. In *SIGMOD 2004*, pages 383–394. ACM, 2004.

[27] H.H. Do and E. Rahm. Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 610–621, 2002.

[28] A.H. Doan, P. Domingos, and A.Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD 2001*, pages 509–520. ACM, 2001.

[29] A.H. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Elsevier Science, 2012.

[30] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12(4):303–319, 2003.

[31] X.L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. *The VLDB Journal*, 18(2):469–500, 2009.

[32] Songyun Duan, Achille Fokoue, Oktie Hassanzadeh, Anastasios Kementsietsidis, Kavitha Srinivas, and Michael J Ward. Instance-based matching

of large ontologies using locality-sensitive hashing. In *The Semantic Web–ISWC 2012*, pages 49–64. Springer, 2012.

[33] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.

[34] H. Elmeleegy, M. Ouzzani, and A. Elmagarmid. Usage-based schema matching. In *IEEE 24th International Conference on Data Engineering*, pages 20–29. IEEE, 2008.

[35] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag New York Inc, 2007.

[36] J. Euzenat and P. Valtchev. Similarity-based ontology alignment in owl-lite. In *ECAI*, volume 16, page 333, 2004.

[37] R. Fagin, L. Haas, M. Hernández, R. Miller, L. Popa, and Y. Velegrakis. Clio: Schema mapping creation and data exchange. *Conceptual Modeling: Foundations and Applications*, pages 198–236, 2009.

[38] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[39] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of the National Conference on Artificial Intelligence*, pages 67–73, 1999.

[40] Avigdor Gal and Tomer Sagi. Tuning the ensemble selection process of schema matchers. *Information Systems*, 35(8):845–859, 2010.

[41] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: an algorithm and an implementation of semantic matching. *The semantic web: research and applications*, pages 61–75, 2004.

[42] Risto Gligorov, Warner ten Kate, Zharko Aleksovski, and Frank Van Harmelen. Using google distance to weight approximate ontology matches. In *Proceedings of the 16th international conference on World Wide Web*, pages 767–776. ACM, 2007.

[43] Aman Goel, Craig A Knoblock, and Kristina Lerman. Using conditional random fields to exploit token structure and labels for accurate semantic annotation. In *AAAI*, 2011.

[44] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[45] Ryutaro Ichise, Hiedeaki Takeda, and Shinichi Honiden. Integrating multiple internet directories by instance-based learning. In *international joint conference on artificial intelligence*, volume 18, pages 22–30, 2003.

[46] Marie Jacob and Zachary Ives. Sharing work in keyword search over databases. In *Proc. SIGMOD*, pages 577–588. ACM, 2011.

[47] Y.R. Jean-Mary, E.P. Shironoshita, and M.R. Kabuka. Ontology matching with semantic verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):235–251, 2009.

[48] S.R. Jeffery, M.J. Franklin, and A.Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD 2008*, pages 847–860. ACM, 2008.

[49] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *The Semantic Web–ISWC 2011*, pages 273–288. Springer, 2011.

[50] Jaewoo Kang and Jeffrey F Naughton. On schema matching with opaque column names and data values. In *International Conference on Management of Data: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, volume 9, pages 205–216, 2003.

[51] C. Knoblock, P. Szekely, J. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, and P. Mallick. Semi-automatically mapping structured sources into the semantic web. *The Semantic Web: Research and Applications*, pages 375–390, 2012.

[52] L Kou, George Markowsky, and Leonard Berman. A fast algorithm for steiner trees. *Acta informatica*, 15(2):141–145, 1981.

[53] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the 2006 ACM SIGMOD*

*international conference on Management of data*, pages 802–803. ACM, 2006.

[54] Ravi Krishnamurthy, Witold Litwin, and William Kent. Language features for interoperability of databases with schematic discrepancies. In *ACM SIGMOD Record*, volume 20, pages 40–49. ACM, 1991.

[55] Wangchao Le, Songyun Duan, Anastasios Kementsietsidis, Feifei Li, and Min Wang. Rewriting queries on sparql views. In *Proceedings of the 20th international conference on World wide web*, pages 655–664. ACM, 2011.

[56] Y. Lee, M. Sayyadian, A.H. Doan, and A.S. Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *The VLDB journal*, 16:97–122, 2007.

[57] Alon Y Levy, Anand Rajaraman, and Joann J Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 251–262, 1996.

[58] J. Li, J. Tang, Y. Li, and Q. Luo. Rimom: A dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.*, 21(8):1218–1232, 2009.

[59] J. Madhavan, P.A. Bernstein, A.H. Doan, and A. Halevy. Corpus-based schema matching. In *21st International Conference on Data Engineering*, pages 57–68. IEEE, 2005.

[60] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the International Conference on Very Large Data Bases*, pages 49–58, 2001.

[61] David Maier, Alberto O Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Systems (TODS)*, 4(4):455–469, 1979.

[62] A. Marie and A. Gal. Boosting schema matchers. *On the Move to Meaningful Internet Systems: OTM 2008*, pages 283–300, 2008.

[63] R. McCann, W. Shen, and A.H. Doan. Matching schemas in online communities: A web 2.0 approach. In *IEEE 24th International Conference on Data Engineering*, pages 110–119. IEEE, 2008.

[64] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *18th International Conference on Data Engineering*, pages 117–128. IEEE, 2002.

[65] G.A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[66] Prasenjit Mitra, Natalya F Noy, and Anuj R Jaiswal. Ontology mapping discovery with uncertainty. In *Proc. 4th International Semantic Web Conference (ISWC)*, volume 3729, pages 537–547, 2005.

[67] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, et al. The open provenance model core specification (v1. 1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[68] Arnab Nandi and Philip A Bernstein. Hamster: using search clicklogs for schema and taxonomy matching. *Proceedings of the VLDB Endowment*, 2(1):181–192, 2009.

[69] Andriy Nikolov, Alfio Ferrara, and François Scharffe. Data linking for the semantic web. *Int. J. Semant. Web Inf. Syst.*, 7(3):46–76, 2011.

[70] Rong Pan, Zhongli Ding, Yang Yu, and Yun Peng. A bayesian network approach to ontology mapping. *The Semantic Web–ISWC 2005*, pages 563–577, 2005.

[71] Rahul Parundekar, Craig A Knoblock, and José Luis Ambite. Discovering concept coverings in ontologies of linked data sources. In *Proc. ISWC*, pages 427–443. Springer-Verlag, 2012.

[72] Eric Peukert, Julian Eberius, and Erhard Rahm. A self-configuring schema matching system. In *IEEE 28th International Conference on Data Engineering (ICDE)*, pages 306–317, 2012.

[73] Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *The International Journal on Very Large Data Bases*, 10(2-3):182–198, 2001.

[74] Li Qian, Michael J Cafarella, and HV Jagadish. Sample-driven schema mapping. In *Proc. SIGMOD*, pages 73–84, 2012.

[75] Yuzhong Qu, Wei Hu, and Gong Cheng. Constructing virtual documents for ontology matching. In *Proceedings of the 15th international conference on World Wide Web*, pages 23–31. ACM, 2006.

[76] Alessandro Raffio, Daniele Braga, Stefano Ceri, Paolo Papotti, and Mauricio A Hernandez. Clip: a visual language for explicit schema mappings. In *IEEE 24th International Conference on Data Engineering*, pages 30–39. IEEE, 2008.

[77] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[78] Carlos R Rivero, Inma Hernández, David Ruiz, and Rafael Corchuelo. Generating sparql executable mappings to integrate ontologies. In *Conceptual Modeling–ER 2011*, pages 118–131. Springer, 2011.

[79] Cristina Sarasua, Elena Simperl, and Natalya F Noy. Crowdmap: crowdsourcing ontology alignment with microtasks. In *The Semantic Web–ISWC 2012*, pages 525–541. Springer, 2012.

[80] Juan F Sequeda, Marcelo Arenas, and Daniel P Miranker. On directly mapping relational databases to rdf and owl. In *Proceedings of the 21st international conference on World Wide Web*, pages 649–658. ACM, 2012.

[81] Juan F Sequeda and Daniel P Miranker. Ultrawrap: Sparql execution on relational data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 22:19–39, 2013.

[82] P. Shvaiko and J. Euzenat. Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2012.

[83] G. Stoilos, G. Stamou, and S. Kollias. A string metric for ontology alignment. *Proc. ISWC*, pages 624–637, 2005.

[84] Partha Pratim Talukdar, Zachary G Ives, and Fernando Pereira. Automatically incorporating new sources in keyword search-based data integration. In *Proc. SIGMOD*, pages 387–398. ACM, 2010.

[85] Aibo Tian, Mayank Kejriwal, and Daniel Miranker. Schema matching over relations, attributes, and data values. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*. ACM, 2014.

[86] Aibo Tian and Matthew Lease. Active learning to maximize accuracy vs. effort in interactive information retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 145–154. ACM, 2011.

[87] Aibo Tian, Juan Sequeda, and Daniel Miranker. Qodi: Query as context in automatic data integration. In *12th International Semantic Web*

*Conference*, pages 624–639. Springer Berlin Heidelberg, 2013.

[88] Aibo Tian, Juan Sequeda, and Daniel P Miranker. Queries, the missing link in automatic data integration. In *International Semantic Web Conference (Posters & Demos)*, 2012.

[89] Aibo Tian, Juan F Sequeda, and Daniel P Miranker. On ambiguity and query-specific ontology mapping. *Proc. ISWC Workshop on Ontology Matching, poster, Boston, US*, 2012.

[90] Robert H Warren and Frank Wm Tompa. Multi-column substring matching for database schema translation. In *Proceedings of the 32nd international conference on Very large data bases*, pages 331–342. VLDB Endowment, 2006.

[91] Chen Jason Zhang, Lei Chen, HV Jagadish, and Chen Caleb Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proceedings of the VLDB Endowment*, 6(9):757–768, 2013.

# Vita

Aibo Tian was born in Tianjin, China in 1986. He received the Bachelor of Engineering degree from the department of Automation, Tsinghua University in 2009. He then enrolled in the University of Texas at Austin to pursue the PhD degree in the department of Computer Science working on automatic data integration systems. He received the Master of Science degree in the department of Computer Science in 2012.

Permanent address: atian@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.