QODI: Query as Context in Automatic Data Integration

Aibo Tian, Juan F. Sequeda, Daniel P. Miranker
Department of Computer Science
The University of Texas at Austin
{atian, jsequeda, miranker}@cs.utexas.edu

ABSTRACT

QODI is an ontology-based data integration system (OBDI) comprising both ontology mapping and query reformulation methods. QODI is distinguished in that the ontology mapping algorithm dynamically determines a partial mapping specific to the reformulation of each query. The mapping algorithm decomposes a query into a set of paths, and compares the sets of paths with a similar decomposition of a source ontology. The query provides application context not available in traditional ontology matching; thereby the system is able to disambiguate mappings for different queries. The path-based solution also simplifies query reformulation, implicitly resolving the problem of missing entity mappings.

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

1. INTRODUCTION

Web-wide integration of structured data is being enabled by the emerging Semantic Web protocols that specify uniform query interfaces to the databases included in the deep web [23]. These developments were recently boosted by W3C ratification of standards for publishing relational database content as RDF [12, 3]. The scope of the deep web underscores the need for automating data integration. The Semantic Web technology stack enables an ontology to serve as a federating data model. Heterogeneous distributed database systems that use an ontology as a federating data model are called ontology-based data integration (OBDI) systems [43].

This paper details the development and performance of an automatic OBDI system, QODI (for Query-driven Ontology-based Data Integration) [41, 40]. QODI considers two OWL ontologies: the target ontology, which is the federating data model, and the source ontology. SPARQL queries are issued over the target ontology by users, and translated to the queries over the source ontology. Although QODI is designed to integrate RDF data sources, 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 [34, 42]. Figure 1 illustrates a pair of relational schemas about the domain of course. Figure 2 illustrates the ontologies per translation rules in the virtualization of those databases as Semantic Web data sources.

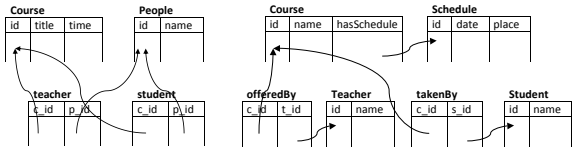
In the typical organization of an OBDI system, ontology mapping is a separate and prerequisite step of query reformulation (see Figure 3(a)). Ontology matching algorithms may be introduced to automatically determine corresponding entities between two ontologies [8, 36]. In this paper, an entity refers to a class or a property. We tested AgreementMaker [11], one of the top finishers in 2010 Ontology Alignment Evaluation Initiative (OAEI) [1]. The highest accuracy of AgreementMaker on all test sets is less than 42%. Inspection of these results revealed two dominant challenges, *missing mapping* and *ambiguous mapping*. We introduce a running example to illustrate the challenges. Figure 2 shows the target (T) and the source (S) ontologies. Figure 4(a) is a SPARQL query q which asks for the time of any course that is taught by Einstein.

- The missing mapping challenge: some entities do not have any mapping, such as the class *Schedule* and property *hasSchedule* in S. Ontology matching algorithms 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*.

- 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 simply not enough information in the ontologies alone to determine a correct mapping.

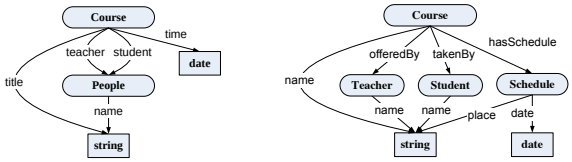
  An example of an ambiguous mapping considers that property *name* of class *People* in T can be mapped to *name* of either class *Teacher* or *Student* in S. There is no basis for preferring one mapping or another. However, if one considers the SPARQL query in Figure 4(a), clearly *Teacher* is preferred. A complementary query would prefer *Student*.

  Some ontology matching programs 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 query will be incorrect. Thus, only after the query is known, is it possible to disambiguate the mapping.



(a) Target Database (b) Source Database

Figure 1: Relational database examples about the domain of course. Arrows represent foreign keys.



(a) Target Ontology (T) (b) Source Ontology (S)

Figure 2: Ontology examples about the domain of course. Oval vertices represent classes, and rectangular vertices are datatypes. Edges represent object properties, or datatype properties.

Ambiguous mappings occur often in real world applications. In our experiments, two out of three test domains have ambiguity. Around 10% to 30% of queries involve ambiguous mappings in the test cases with ambiguity.

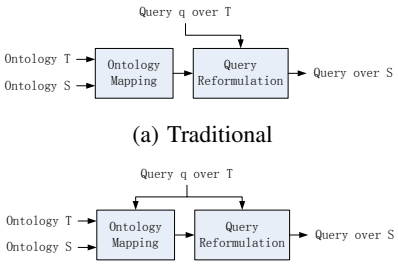
To resolve these challenges, we introduce *query-specific ontology mapping*. For each input query, the system determines a partial ontology mapping sufficient to reformulate the specific query. In effect, a query becomes a third argument to the ontology mapping algorithm (see Figure 3(b)). To resolve ambiguous mappings, the query is leveraged to provide context for choosing among competing mappings. Note that using the query as context requires no extra input from users or experts.

Centering on the subgraph components of ontologies and queries enables another departure from typical mapping systems. 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. The similarity is not dependent on the precise alignment of entities within paths, so the missing mapping challenge is resolved. 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 heuristics to efficiently find the mapping. The heuristic search algorithms guarantee to find an optimal solution.

In our running example, the path that contains *People* and *name* in query q also contains the property *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 in Figure 4(a) should be:

$$\{T{:}Course, T{:}teacher, T{:}People, T{:}name, string\}$$
$$= \{S{:}Course, S{:}offeredBy, S{:}Teacher, S{:}name, string\}$$
$$\{T{:}Course, T{:}time, date\}$$
$$= \{S{:}Course, S{:}hasSchedule, S{:}Schedule, S{:}date, date\}$$

Note that this mapping is specific to the query. If another query asks for the time of any course that is taken by *Einstein*, the mapped path should contain S:*Student* instead of S:*Teacher*.



(a) Traditional

(b) The proposed, QODI

Figure 3: Diagram of OBDI systems with traditional and the proposed ontology mapping components.

QODI is evaluated on real world test cases from three domains: Bibliography, Conference Organization and Life Science. The baselines include an ontology-based implementation of Clio, the state of the art data integration / exchange system, and ontology matching systems that generate correspondences between entities, such as AgreementMaker. QODI outperforms all baselines on all test cases.

2. RELATED WORK

As stated in a recent book of data integration, some aspects of data integration can be viewed as a knowledge representation problem, which uses ontologies as data models [15]. Recent review articles, one on schema matching [8] and the other on ontology matching [36], make note of the overlap between schema and ontology matching problems. As remarked by Shvaiko and Euzenat, " they often share similar matching solutions" [36].

Doan et. al. states that "relational schemas can be viewed as ontologies with restricted relationship types" [16]. This is exploited in our test sets, which includes ontologies translated from relational data sources by Ultrawrap [35]. Ultrawrap details the syntax and semantics of a translation from relational schemas to OWL ontologies. In addition to the readily apparent mapping of tables to classes and attributes to datatype properties, the translation considers integrity constraints to create object properties [42, 34].

Clio, the state-of-the-art semi-automatic data integration and exchange system has close similarities with QODI [19]. 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 [30]. Thus, part of the mapping process in Clio exploits an implicit graph representation of the relational schema. Recall that the translation from relational schemas to ontologies by Ultrawrap 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. The mapping result of Clio is independent from user queries. In the case of ambiguity, Clio may generate incorrect mappings in the first step, and propagate those errors to the final mapping. QODI dynamically generates mapping for each input query. The query provides context information to resolve ambiguous mappings. Based on our experiments, QODI's capability of disambiguation provides measurable performance improvement.

Contextual information has been exploited in schema mapping by Bohannon [9]. Context is mined from categorical attributes.

This research is limited to a static mapping without considering queries. Thus, it does not address the ambiguity problem.

Pay-as-you-go data integration systems are designed to resolve large-scale data integration problems [24, 13]. They are distinguished by accumulating partial mappings over time. Those frameworks are designed to scale up the data integration systems, instead of addressing ambiguity.

Ambiguity and uncertainty are related. Dong proposes probabilistic schema mapping, which lists all entity mapping candidates with probabilities, to detail uncertainty problem [17]. In QODI, the probability of entity mapping is captured by the similarity between entities. Our claim is that the correct mapping can be determined given the user query as constraints. Uncertainty in schema matching has also been studied for XML schemas [21].

A semi-automatic OBDI system, Karma, is recently built to map structured data sources to ontologies [25]. Similar as Karma, the combination of QODI and Ultrawrap is also capable for integrating structured sources. However, the mapping component of Karma does not consider the context from queries for disambiguation. Ontology based data access (OBDA) uses ontologies expressed in Description Logic as a conceptual view over data sources [10, 26]. Our method can also be used to generate mappings for OBDA with proper representation.

Schema matching and ontology matching have been well studied [7, 36, 18, 8]. Existing schema matching systems use schemas and instances for matching, such as iMap [14], Similarity Flooding [31], GLUE [16], Corpus-based matching [28], and Cupid [29]. Sorrentino et. al. proposes schema normalization for schema matching [37]. Multiple schema mappings are merged in [2]. Many ontology matching systems compete in the OAEI [1], such as AgreementMaker [11], RiMOM [27], and COMA++ [5]. Parundekar, Knoblock, and Ambite discover mapping between concepts through conjunctions and disjunctions of linked data [32]. PARIS align ontologies and instances at the same time [39]. Zhong et. al. considers matching between unbalanced ontologies [44]. In addition to using schemas and instances, SPHINX [6] and MWeaver [33] consider user-given examples for matching. In the ontology matching and schema matching world, most of systems focus on mapping between entities. In this paper, we formally define query-specific mapping to facilitate OBDI systems.

3. PROBLEM DEFINITION

The approach comprises decomposing a SPARQL query into a set of paths, then matching and scoring each of those paths with a path in the source ontology. The following section begins with graph definitions and culminates with the formal definition of the mapping problem, a *q-mapping*.

3.1 Basic Definition

DEFINITION 3.1 (ONTOLOGY). *An ontology is defined as a tuple $< C, P_H, P_O, P_D, D >$, where C, P_H, P_O, P_D, and D represent the finite sets of classes, class hierarchies, object properties, datatype properties, and datatypes, respectively.*

Classes represent concepts. Object properties are relationships between classes. The domain of an object property p is the class that p belongs to. The range of an object property p is the class that the value of p belongs to. Datatype properties are relationships between classes and datatypes. The domain of a datatype property p is the class that p belongs to. The range of a datatype property p is the type of the value of p. Defined by the direct mapping, a relational schema can be translated to an ontology by applying rules [35, 42]. The rules include tables to classes, the attributes that are foreign keys to object properties, and the attributes that are not foreign keys to datatype properties. See Figure 1 and 2 for relational databases and their corresponding ontologies.

An *ontology graph* is a representation of an ontology as a directed labeled graph (see Figure 2).

DEFINITION 3.2 (ONTOLOGY GRAPH). *An Ontology Graph, $G = (V, E)$, with vertex set V and edge set E, is a directed labeled graph, where $V = C \cup D$, $E = P_O \cup P_D$, and with the constraint that for each edge e, the initial vertex is its domain and the terminal vertex is its range.*

To simplify handling inheritance relationships, rather than coding the logic of inheritance into the path-related algorithms, the graph representation of an ontology is expanded by replicating properties. If the domains or ranges of a property have subclasses, new edges with the same label as the property are created for each of the subclasses.

Target and source ontologies are distinguished as T and S, respectively. We also use these notations to interchangeably denote ontologies or ontology graphs. The target ontology represents the federating data model.

DEFINITION 3.3 (SOURCE). *A source of a directed labeled graph G is a vertex with 0 in-degree. The set of all sources of G is denoted $SOURCE_G$.*

LEMMA 3.1. *The sources of an ontology graph only represent classes.*

The proof follows from the fact that vertices representing datatypes have non-zero in-degree.

DEFINITION 3.4 (SINK). *A sink of a directed labeled graph G is a vertex with 0 out-degree. The set of all sinks of G is denoted $SINK_G$.*

DEFINITION 3.5 (PATH). *A path in a directed labeled graph G is an ordered list of vertices and edges such that: 1) the first and last elements of the path are vertices; 2) for each vertex v that is neither the first nor last element in the list, both its previous element and next element are edges; 3) for each edge e in the list, its previous element is its initial vertex in the graph, and its next element is its terminal vertex.*

The length of a path p, denoted as $|p|$, is the sum of the number of vertices and edges in p. A path of graph G is also a subgraph of G, although the representation is different. We use path to interchangeably denote both a path and a corresponding subgraph.

DEFINITION 3.6 (SS-PATH). *A source-to-sink path or ss-path is a path from vertex v_1 to vertex v_2 in a directed labeled graph G, where v_1 is a source and v_2 is a sink.*

More than one ss-path may connect a source and a sink.

DEFINITION 3.7 (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$_{G,v_1,v_2}$).*

We also use SS-PATH-SET$_{G,v_1}$ to denote the set of all ss-paths from source v_1 to all sinks.

As an example, the SS-PATH-SET$_{T,Course,string}$ contains three ss-paths (see Figure 2).

$$\{\{Course, title, string\}$$
$$\{Course, teacher, People, name, string\}$$
$$\{Course, student, People, name, string\}\}$$

Algorithm 1 Generate a query graph.

Input: ontology T and SPARQL query q
Output: query graph Q

```
V = ∅, E = ∅
// Construct a map to record all rdf:type
M is a map from variables to classes
for all triple pattern t in q do
    s, p, o is the subject, predicate and object of t
    if p is rdf:type then
        Add s as key and o as value to M
    end if
end for
// Translate triple patterns into a subgraph of T
// Vertices are either given by rdf:type or inferred as the domains and ranges of predicates
// Edges are predicates
for all triple pattern t in q do
    s, p, o is the subject, predicate and object of t
    if p is not rdf:type then
        if M has key s then
            d is the value of s in M
        else
            d is a domain of p
        end if
        if M has key o then
            r is the value of o in M
        else
            r is a range of p
        end if
        Add d and r to V
        Add p to E, directing from d to r
    end if
end for
return Q = (V, E)
```

DEFINITION 3.8 (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.*

A *query graph* is a subgraph of an ontology graph that corresponds to a SPARQL query. The query graph of the SPARQL query in Figure 4(a) is shown in Figure 4(b).

DEFINITION 3.9 (QUERY GRAPH). *Given a SPARQL query q over ontology T, a query graph (denoted as Q) is the subgraph of T generated by applying Algorithm 1.*

3.2 Assumptions

Basic assumptions are as follows:

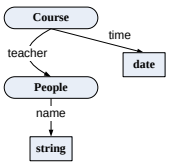
1. All object properties and datatype properties have domains and ranges. This assumption simplifies the construction of ontology graphs. For manually designed ontologies, high quality ones have domains and ranges. For the ontologies translated from relational schemas, the domains and ranges are created automatically.

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 in the query or is provided by rdf:type. Given these assumptions, there exists only one query graph for each query. For simplicity, we leave the relaxing of these assumptions for future work. If multiple query graphs are allowed, each of them can be mapped separately.

3. The sinks of a query graph only represent datatypes. This paper focuses on ontologies that describe database content. User queries are mostly issued for retrieving or transferring data, which are values of datatype properties. Retrieving database data ultimately requires the rewriting of datatype properties.

```
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 over T



(b) Query graph

Figure 4: An example of SPARQL query, which asks for the time of any course taught by *Einstein*. A question mark indicates that element is a variable.

3.3 Query-Specific Mapping

Given the target ontology T, the source ontology S, and a query q over T, our task is to find correspondences between the ss-paths in the query graph Q to the paths in the source ontology graph.

An ss-path correspondence records the mapping confidence between two ss-paths. We consider the relation of the two ss-paths as equivalence. The expressions of ss-path correspondence is in GLAV formalism [20].

DEFINITION 3.10 (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.*

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.

DEFINITION 3.11 (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 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 SOURCE$_{p_1}$ = SOURCE$_{p_2}$, the two corresponding ss-paths $p'_1, p'_2 \in$ GRAPH-SS-PATH-SET$_G$ also share a common source, SOURCE$_{p'_1}$ = SOURCE$_{p'_2}$, where $\pi_{p_1,p'_1} \in \Omega_{Q,G}$, $\pi_{p_2,p'_2} \in \Omega_{Q,G}$.*

This definition contains several constraints. First, all sinks of G are datatypes. Recall that the sinks of the query graph Q are also datatypes. Thus, the values of datatype properties in the input query can be translated to the values in the reformulated query. 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 3.12 (Q-MAPPING). *Given two ontology graphs T, S, and a SPARQL query q over T, the query-specific ontology mapping (denoted as 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;*
- *$\beta_{\Omega_{Q,\bar{G}}} = \max_{G \subseteq S} \beta_{\Omega_{Q,G}}$.*

4. QODI: MAPPING ALGORITHM

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

4.1 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 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 is a list of strings containing the labels of all entities except datatypes in an ss-path. Path labels are used to disambiguate uncertain mappings. Path length differences are penalized.

Similarity estimation of source classes and datatype properties has been well studied in prior work [11, 27, 14, 18]. Any existing method may be used for this component. We evaluated three matchers: Substring string similarity that measures the portion of the longest common substrings, SMOA string similarity [38], and AgreementMaker [11]. Substring similarity is a simple string distance measure, SMOA is a sophisticated string distance measure, and AgreementMaker is an ontology matching system that considers both labels and structures. 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 label may contain a different number of tokens, the frequency of a token is set to one over the number of tokens (after removing stop words) in a label. The path label similarity, S_L, between two ss-paths p and p' is computed as,

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

where $\mathbf{f}_i(p)$ is the ith 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 should not have large difference. We use an exponential function to penalize the path length difference. The penalty S_{LE} of two paths p and p' is defined as,

$$S_{LE}(p,p') = e^{-\eta \cdot | |p| - |p'| |} \quad (2)$$

where $|p|$ is the length of path p, and η is a non-negative real number. With the same length difference, a large η gives more penalty.

The ss-path similarity measure, S_{SS}, is defined as,

$$S_{SS}(p,p') = S_C(p,p')^{\frac{1}{n_p}} \cdot S_D(p,p') \cdot S_L(p,p') \cdot S_{LE}(p,p') \quad (3)$$

where S_C and S_D are similarity measures for source classes and datatype properties, which are provided by matchers. n_p is the number of ss-paths in the query graph that share the same source as 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. Note that an ss-path contains one source and one datatype property.

4.2 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 as,

$$\begin{aligned}
\bar{G} &= \arg\max_{\Omega_{Q,G} \in \mathcal{M}_Q} \beta_{\Omega_{Q,G}} \\
&= \arg\max_{\Omega_{Q,G} \in \mathcal{M}_Q} \{ \prod_{\pi_{p,p'} \in \Omega_{Q,G}} S_{SS}(p,p') \} \\
&= \arg_{G \subseteq S} \{ \prod_{c \in SOURCE(Q)} \{ \max_{c' \in SOURCE(G)} \{ \\
&\qquad \prod_{p \in \text{SS-PATH-SET}(Q,c)} \{ \max_{p' \in \text{SS-PATH-SET}(G,c')} S_{SS}(p,p') \} \} \} \} \quad (4)
\end{aligned}$$

q-mapping(T,S,q) is the set of ss-path correspondences $\Omega_{Q,\bar{G}}$ between Q and $\bar{G}$. Equation (4) specifies the problem of finding the optimal subgraph of S as the problem of finding a set of optimal ss-paths that shares a source in S for each set of ss-paths that shares a source in Q. This reflects the restriction in Definition 3.11 that, if two ss-paths in query graph Q share a source, the corresponding ss-paths in the match candidate also share a source. The maximization is a joint process at two levels. First, the similarity between ss-paths models a joint probability over all entities within the ss-paths. Second, all ss-paths with the same source are mapped as a group.

DEFINITION 4.1 (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.*

If a q-mapping is not a complete q-mapping, we can include the number of mapped ss-paths as an additional measure, and compute the confidence measure based only on mapped ss-paths. The mapping with the largest number of mapped ss-paths is used as the mapping. If multiple mappings have the same number of mapped ss-paths, the one with the highest confidence measure is used.

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

Algorithm 2 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 = ∅
for all sink s of G do
  q.enqueue(s)
end for
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
```

4.3 Solving the Maximization

Equation (4) 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 given 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 [22]. In this context, 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, which means the heuristic never overestimates the actual cost. 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, 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 rather than the sum of two functions. The search terminates when a path $\bar{p}$ containing a sink (datatype) is pulled from $\mathcal{P}$.

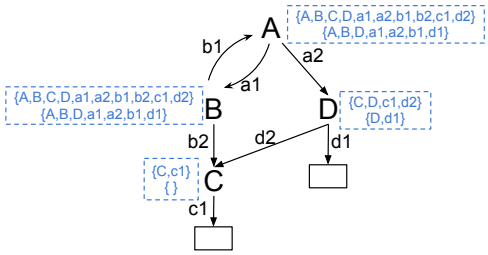


Figure 5: An example ontology graph with reachable label sets. The dashed boxes around each non-sink vertex contain the reachable label sets through the two datatype properties $c1$ and $d1$. For example, the reachable label set from C through $c1$ is $\{C, c1\}$, and through $d1$ is empty.

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 algorithm before discovering the similarity between datatype properties is low. To address this, $\mathcal{P}$ is initialized by a set of paths, such that each path 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 by the heuristic cost function.

The second technique is a preprocessing step that associates reachable label sets and shortest path lengths to each interior vertices of the ontology. 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 each datatype property. Figure 5 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 2. 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, a 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, and the reachable label sets and the lengths of shortest paths are stored for fast lookup.

We denote the ss-path in the query graph as q, 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 $l_{x,e}$. s_c and s_d are the source class similarity, and the datatype property similarity, respectively. The heuristic cost function h is defined as follows:

$$h(p) = -\ {s_c}^{\frac{1}{n_q}} \cdot s_d \cdot \frac{\sum_{i=1}^{m} \min(\mathbf{f}_i(q), \mathbf{f}_i(p) + \overline{\mathbf{f}}_i(q,p,L_{x,e}))}{\sum_{i=1}^{m} \mathbf{f}_i(q) + \mathbf{f}_i(p) - \min(\mathbf{f}_i(q), \mathbf{f}_i(p))} \cdot e^{-\eta \cdot g(q,p,l_{x,e})} \tag{5}$$

where n_q is the number of paths in the query graph that share the same source as q, and $\mathbf{f}_i(p)$ is the ith element of the feature vector of path p. $\overline{\mathbf{f}}_i(q,p,L_{x,e})$ and $g(q,p,l_{x,e})$ are defined as,

$$\overline{\mathbf{f}}_i(q,p,L_{x,e}) = \begin{cases} \max(\mathbf{f}_i(q) - \mathbf{f}_i(p),\ 0) & \text{, if } x \neq e \text{ and string } i \in L_{x,e} \\ 0 & \text{, otherwise} \end{cases} \tag{6}$$

$$g(q,p,l_{x,e}) = \begin{cases} \max(|p| + l_{x,e} - 1 - |q|,\ 0) & \text{, if } x \neq e \\ |\ |p| - |q|\ | & \text{, if } x = e \end{cases} \tag{7}$$

Let us denote the path as $\bar{p}$, when the search terminates. Based on the termination condition, $x = e$. Substituting x with e, $h(\bar{p}) = -S_{SS}(q,\bar{p})$, where S_{SS} is defined in (3). The following lemma and theorem prove that $\bar{p}$ is the best scoring path. Lemma 4.1 corresponds to the proof of admissibility of the heuristic in A* search, and Theorem 4.1 corresponds to the proof of correctness.

LEMMA 4.1. *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 the optimal path $\tilde{p}$. Per the definition of $\overline{\mathbf{f}}_i$, 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 forth 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. $|p| + l_{x,e} - 1 \geq |q|$. Then $g(q,p,l_{x,e}) = |p| + l_{x,e} - 1 - |q|$, and $g(q,\tilde{p},l_{e,e}) = |\tilde{p}| - |q|$. Thus, $g(q,p,l_{x,e}) \leq g(q,\tilde{p},l_{e,e})$.
2. $|p| + l_{x,e} - 1 < |q|$. Then $g(q,p,l_{x,e}) = 0$, and $g(q,\tilde{p},l_{e,e}) \geq 0$. Thus, $g(q,p,l_{x,e}) \leq g(q,\tilde{p},l_{e,e})$.

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

THEOREM 4.1. *When the search terminates, the path $\bar{p}$ is an optimal path.*

The proof of Theorem 4.1 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 $h(p)$ [22]. We omit further details.

If there is no path from the given vertex through any datatype property, the reachable label sets of the vertex are all empty. We do not need to run the search algorithm. Otherwise, Theorem 4.1 guarantees that the algorithm will find an optimal path. An optimal path has finite length. If a path p has infinite length, h is infinitesimal, because the forth factor of h is infinitesimal while the other factors are bounded by 1. Most real world queries do not have cycles, so we prune cyclic paths during the search to further reduce the computation. If a path contains a vertex more than once or has two vertices in the same class hierarchy, the path is removed. 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. It is not easy to extend the search algorithm in phase 1) to the second phase. A naive algorithm can search all classes, and apply the search algorithm in phase 1) with each of them as the source vertex. We exploit another heuristic algorithm to prune unlikely classes. 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 4.1.

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

Algorithm 3 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
```

5. QODI: QUERY REFORMULATION

A central challenge in query reformulation, missing mapping, is resolved prior to the reformulation step in QODI. The missing mapping problem in QODI 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. In traditional data integration systems with entity-to-entity mapping, the query reformulation algorithm must include explicit postprocessing steps, e.g. the *Generalize-With-Join* operator [4], when lacking subgraph isomorphism.

Instead, QODI's query reformulation algorithm exploits the fact that a q-mapping contains ss-path correspondences that cover the

query graph. Pseudo-code for the reformulation algorithm is given in Algorithm 3. Informally, the algorithm traverses the mapped ss-paths in the correspondences, and generates a triple pattern for each graph edge. The URI of each edge in the ss-path is translated as the 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, since the primary focus of this paper is mapping.

6. EXPERIMENTAL SETUP

The objectives of the evaluation are threefold: to determine how well the method resolves ambiguity, how capable the method is of generating complete q-mappings for a SPARQL query, and finally, how accurate the mappings are.

6.1 Test Sets

The test sets comprise one pair of ontologies from each of three domains: Bibliography, Conference Organization and Life Science. The test cases include three ontologies used in OAEI [1], two ontologies created from direct mapping relational databases, and one ontology created by a standards organization.

The Bibliography domain comprises the UMBC ontology from the OAEI benchmark track, and an ontology that models DBLP, generated from the direct mapping of a SQL schema defined for a DBLP metadata database through Ultrawrap [35] . Class hierarchies are manually added. The Conference domain consists of the two ontologies from the OAEI conference track, SIGKDD and SOFSEM. 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). The Specify ontology was created from direct mapping the SQL schema of the database in the Specify biological collections software package[2]. Specify is used in over 200 locations. We have made the ensemble test suite available on our website[3].

Sets of test queries are also required, and were created as follows. First, groundtruth mappings were manually generated, containing all correct ss-path mappings between each pair of ontologies. Subsequently, a computer program systematically generated two kinds of SPARQL queries for each ontology. 1) A *PathOnly* query has a query graph consisting of only one ss-path in the groundtruth mappings. 2) A *ClassAll* query has a query graph consisting of the set of all ss-paths that share a source in the groundtruth mappings. All queries with only one ss-path are removed from ClassAll queries.

The primary motivation for PathOnly queries is to test the mapping systems' ability to generate correct path mappings. ClassAll queries are complex compared to PathOnly queries. For example, one ClassAll query for Life Science contains 14 ss-paths in the query graph. If any of these ss-paths is incorrectly mapped, the mapping fails. Since the queries were generated from groundtruth mappings, they have complete q-mappings.

Figure 6 shows examples of PathOnly and ClassAll queries generated for the DBLP ontology.

6.2 Metrics

One key metric is to measure ambiguity. An accurate measure of ambiguity requires a manual alignment of ontologies anticipating a wide range of applications. Since a perfect measure would have to take into account all possible application scenarios, it is difficult for a manual measure to assure perfect accuracy.

[2]http://specifysoftware.org/

[3]http://www.cs.utexas.edu/~atian/page/dataset.html

```
Select ?v
Where {
 ?c0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY_PROCEEDINGS>.
 ?c0 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY#url> ?c1.
 ?c1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY_URL>.
 ?c1 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY_URL#url> ?v.
}
```

(a) PathOnly SPARQL query

```
Select ?v0 ?v1 ?v2 ?v3
Where {
 ?c0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY>.
 ?c0 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY#title> ?v0.
 ?c0 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY#type> ?v1.
 ?c0 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY#url> ?c1.
 ?c1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY_URL>.
 ?c1 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY_URL#url> ?v2.
 ?c0 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_ENTITY#author> ?c2.
 ?c2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://ribs.csres.utexas.edu/onto/dblp/DBLP_AUTHOR>.
 ?c2 <http://ribs.csres.utexas.edu/onto/dblp/DBLP_AUTHOR#author> ?v3.
}
```

(b) ClassAll SPARQL query

Figure 6: Example SPARQL queries in the query set

We define an approximate measure of ambiguity. There are two aspects to the approximation. First, only mapping between datatype properties is considered as the source of ambiguity. Second, two datatype properties are considered as mapped if a matcher, as opposed to a human labeler, assigns them the highest similarity.

DEFINITION 6.1 (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 Ω 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 $S_D(p_t, p_s)$.*

Note that the definition of datatype ambiguous q-mapping depends on the similarity measure of a matcher.

$ambiguous\_rate$ is used as the metric to measure the proportion of queries that have datatype ambiguous q-mappings.

$$ambiguous\_rate = \frac{\#\text{ queries with datatype ambiguous q-mapping}}{\#\text{ queries}} \quad (8)$$

where # represents *the number of*.

The remaining assessments concern the proportion of queries for which a system is able to generate complete q-mappings independent of correctness, and subsequently, the proportion of queries that are correctly mapped. The metrics are reminiscent of recall and precision used in ontology matching and information retrieval.

$valid\_rate$ is the proportion of queries with complete q-mappings generated.

$$valid\_rate = \frac{\#\text{ queries with complete q-mappings generated}}{\#\text{ queries}} \quad (9)$$

For measuring the accuracy of mapping systems, we should consider the case that a query is correctly mapped, and also the case that part of a query is correctly mapped. Thus, we define both a query-based precision measure, $query\_precision$, and a path-based precision measure, $path\_precision$.

$$query\_precision = \frac{\#\text{ correctly mapped queries}}{\#\text{ queries}} \quad (10)$$

$$path\_precision = \frac{\sum_q \text{percentage of correctly mapped ss-paths in } q}{\#\text{ queries}} \quad (11)$$

	B↑	B↓	C↑	C↓	L↑	L↓
ambiguous_rate	0.242	0.113	0	0	0.333	0.111
Substring	0.267	0.000	-	-	0.333	0.250
Clio_Substring	0.667	0.000	-	-	0.417	**0.500**
QODI_Substring	**0.933**	0.000	-	-	**0.500**	**0.500**

Table 1: The ambiguous_rate and query_precision with Substring as matcher. Row 2 contains the ambiguous_rates of different test sets. Rows 3, 4, and 5 contain the query_precisions of different methods that only count the queries with datatype ambiguous q-mapping in PathOnly query sets. 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. L↑ uses Darwin Core and L↓ uses Specify as the target ontology for the Life Science 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.

	B↑	B↓	C↑	C↓	L↑	L↓
ambiguous_rate	0.177	0	0	0	0.194	0
SMOA	0.364	-	-	-	0.143	-
Clio_SMOA	0.364	-	-	-	0.429	-
QODI_SMOA	**0.636**	-	-	-	**0.714**	-

Table 2: The ambiguous_rate and query_precision with SMOA as matcher. See caption of Table 1 for details.

where an individual query in the test set is denoted q.

6.3 Baselines

We compare QODI against two kinds of baselines: ontology matching systems, and methods from an existing mapping system for data integration.

In ontology matching systems, 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 [19]. Clio generates mappings between attributes in relational databases, and finds associations between those mappings through foreign key constraints. The identification of foreign key constraints to associate tables is similar to the direct mapping implemented by Ultrawrap [35] that generated the ontologies used in our test cases. Thus, we are able to implement baselines similar to 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. The only difference between Clio baselines and QODI is that these baselines do not consider the labels inside a path.

We use three matchers for the proposed method and all baselines. 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 [38]. The third is AgreementMaker configured as detailed in OAEI 2010 conference track [11].

	B↑	B↓	C↑	C↓	L↑	L↓
ambiguous_rate	0	0	0	0	0.139	0
AgreementMaker	-	-	-	-	0.000	-
Clio_AgreementMaker	-	-	-	-	0.000	-
QODI_AgreementMaker	-	-	-	-	**0.200**	-

Table 3: The ambiguous_rate and query_precision with AgreementMaker as matcher. See caption of Table 1 for details.

7. EXPERIMENTAL RESULTS

Given a pair of ontologies, O_1 and O_2, the experiments are conducted on two directions of mappings: from O_1 (target) to O_2 (source), and from O_2 (target) to O_1 (source). The results for the two mapping directions are shown separately for ambiguous_rate to distinguish the differences. For valid_rate, query_precision and path_precision, the average results of both mapping directions are reported. We set $\eta = 0.3$ based on the tuning on the Bibliography test set with PathOnly queries using Substring as matcher. Section 7.4 discusses the accuracy using different η.

7.1 Ambiguity

Two questions concern ambiguity. How ambiguous are our test sets? What is the accuracy of a method on queries with ambiguous mappings? For the first question, we measure the ambiguous_rate. For the second question, we measure the query_precision of PathOnly queries with ambiguous mappings. Since each PathOnly query corresponds to one path mapping in the groundtruth, the query_precision shows the capability of disambiguation.

Per Definition 6.1 and Equation (8), ambiguous_rate is related to matchers. The results using each matcher are reported separately in Table 1, 2, and 3.

Two out of three test sets, Bibliography and Life Science, have non-zero ambiguity. The ambiguous_rate measured with different matchers are different, but they share similarities on the two test sets. All three matchers assert that L↑ has ambiguity. The rates range from 0.139 to 0.333. Two matchers, Substring and SMOA, agree on the ambiguity of B↑. The rates are 0.242 and 0.177. For both L↑ and B↑, QODI achieves the highest query_precision on the queries with datatype ambiguous q-mappings. Especially for B↑, QODI achieves 0.933 and 0.636 query_precision comparing to 0.667 and 0.364 from Clio. This shows that QODI is capable of disambiguation.

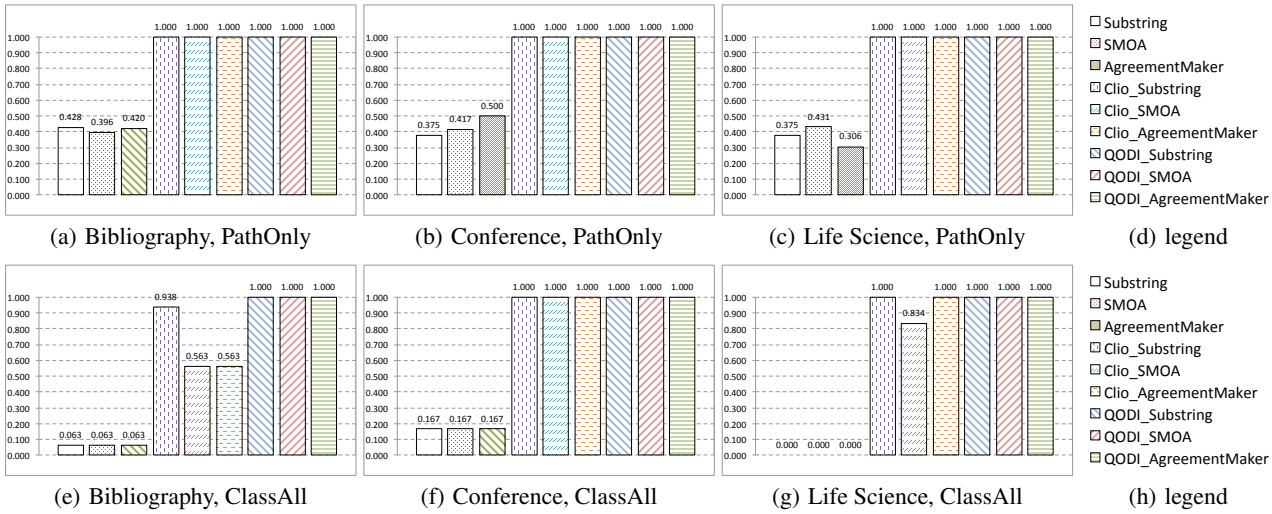
7.2 Valid_rate

Figure 7 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 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 as shown in Table 1, 2, and 3. For some ClassAll queries, Clio cannot find a complete q-mapping if the mapped entities are incorrectly selected from ambiguous mappings.

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

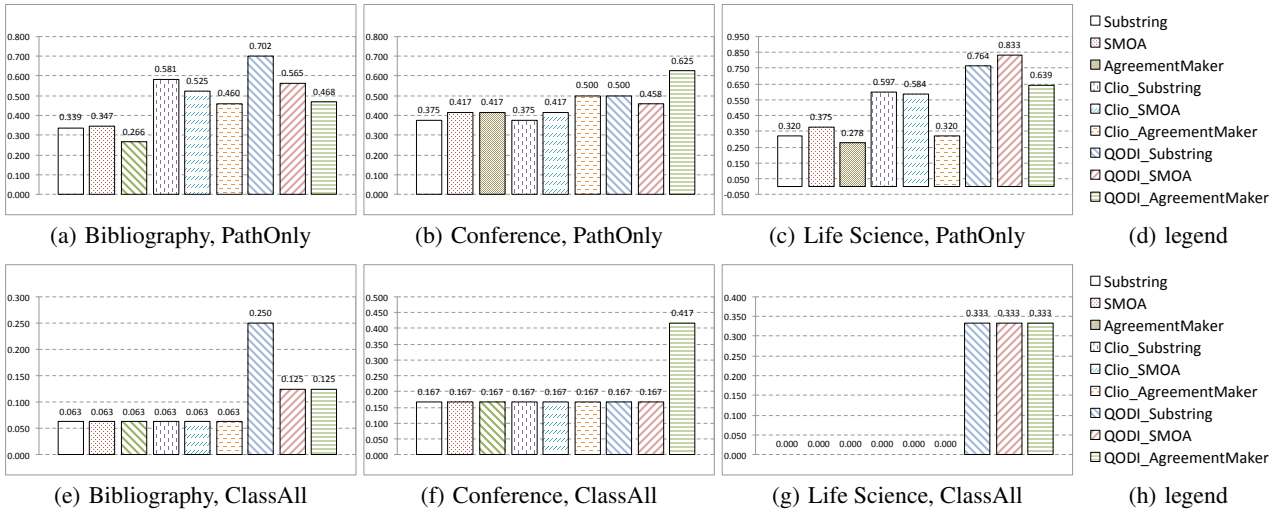
The generic mapping baselines are able to generate complete q-mappings for less than 50% of PathOnly queries, but barely generate complete q-mappings for ClassAll queries. The big gap be-



(a) Bibliography, PathOnly (b) Conference, PathOnly (c) Life Science, PathOnly (d) legend

(e) Bibliography, ClassAll (f) Conference, ClassAll (g) Life Science, ClassAll (h) legend

Figure 7: Valid_rate for different test sets. Higher number means better performance.



(a) Bibliography, PathOnly (b) Conference, PathOnly (c) Life Science, PathOnly (d) legend

(e) Bibliography, ClassAll (f) Conference, ClassAll (g) Life Science, ClassAll (h) legend

Figure 8: Query_precision for different test sets. Higher number means better performance.

tween generic mappings and Clio baselines demonstrates the importance of the missing mapping challenge.

7.3 Precision

Figure 8 and 9 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 the ClassAll query set, each query has more than one path that shares a source. On one hand, more paths 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 baselines to generic mapping baselines, for all test sets and all measures, at least one Clio baseline dominates or performs as well as generic mapping baselines.

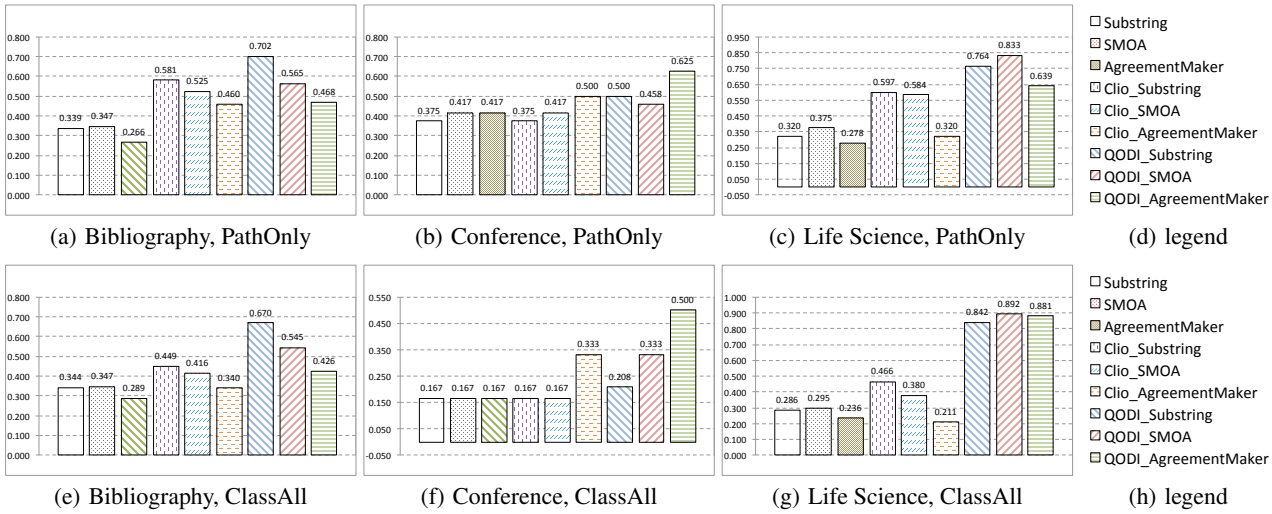
7.4 Parameter Tuning

Figure 10 shows the query_precision using different path length penalty parameter η. With the same length difference, a large η 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 η is too large, the query_precision can be decreased. With large η, the penalty of length dominates the similarities of source classes, datatype properties, and path labels in (3). 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 η.

8. DISCUSSION

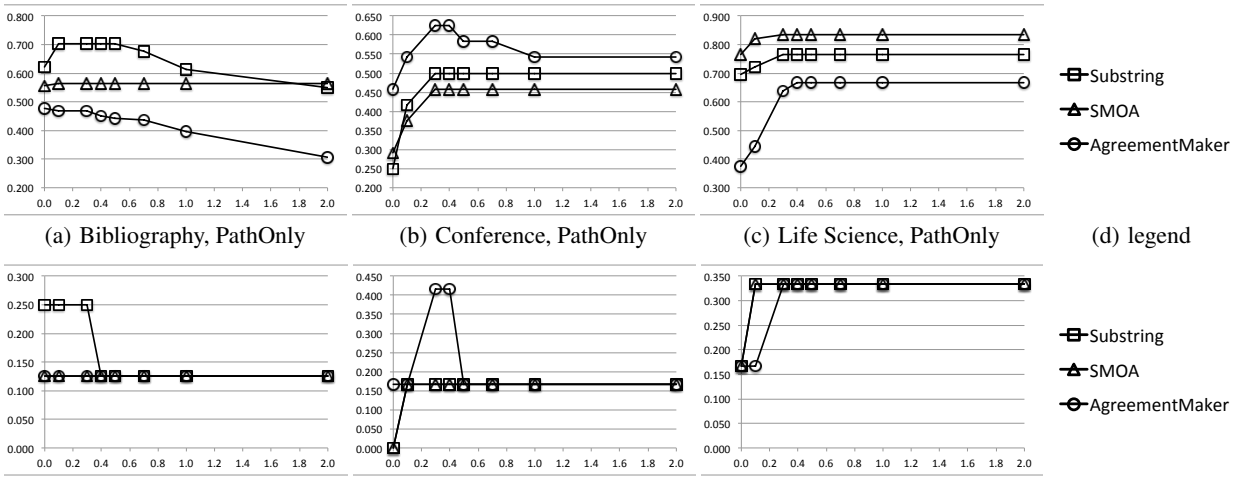
This work identifies ambiguous mappings as sources of errors in automatic data integration. In our experiments, two out of three test domains have ambiguity. Around 10% to 30% of queries involve ambiguous mappings in the test sets with ambiguity.



(a) Bibliography, PathOnly (b) Conference, PathOnly (c) Life Science, PathOnly (d) legend

(e) Bibliography, ClassAll (f) Conference, ClassAll (g) Life Science, ClassAll (h) legend

Figure 9: Path_precision for different test sets. Higher number means better performance.



(a) Bibliography, PathOnly (b) Conference, PathOnly (c) Life Science, PathOnly (d) legend

(e) Bibliography, ClassAll (f) Conference, ClassAll (g) Life Science, ClassAll (h) legend

Figure 10: Query_precision (vertical axis) when using different η (horizontal axis).

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.

9. REFERENCES

[1] Ontology Alignment Evaluation Initiative. `oaei.ontologymatching.org/`.

[2] B. Alexe, M. Hernández, L. Popa, and W.-C. Tan. Mapmerge: correlating independent schema mappings. *The VLDB Journal*, 21(2):191–211, 2012.

[3] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. Direct mapping of relational data to RDF. W3C Recommendation.

[4] Y. Arens, C. Knoblock, and W. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2):99–130, 1996.

[5] D. Aumueller, H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *Proc. SIGMOD*, pages 906–908. ACM, 2005.

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

[7] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Springer, 2011.

[8] P. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *Proc. VLDB*, 4(11), 2011.

[9] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting context into schema matching. In *Proc. VLDB*, pages 307–318. VLDB Endowment, 2006.

[10] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. Savo. The mastro system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.

[11] I. Cruz, F. Antonelli, and C. Stroe. Agreementmaker: efficient matching for large real-world schemas and ontologies. *Proc. VLDB*, 2(2):1586–1589, 2009.

[12] S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to rdf mapping language. W3C Recommendation.

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

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

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

[16] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12(4):303–319, 2003.

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

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

[19] 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.

[20] M. Friedman, A. Levy, T. Millstein, et al. Navigational plans for data integration. In *Proc. AAAI*, pages 67–73, 1999.

[21] J. Gong, R. Cheng, and D. W. Cheung. Efficient management of uncertainty in xml schema matching. *The VLDB Journal*, 21(3):385–409, 2012.

[22] P. Hart, N. 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.

[23] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.

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

[25] 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.

[26] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to ontology-based data access. In *Proc. IJCAI*, pages 2656–2661, 2011.

[27] 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.

[28] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *Proc. ICDE*, pages 57–68. IEEE, 2005.

[29] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proc. VLDB*, pages 49–58, 2001.

[30] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Systems (TODS)*, 4(4):455–469, 1979.

[31] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. ICDE*, pages 117–128. IEEE, 2002.

[32] R. Parundekar, C. A. Knoblock, and J. L. Ambite. Discovering concept coverings in ontologies of linked data sources. In *Proc. ISWC*, pages 427–443. Springer-Verlag, 2012.

[33] L. Qian, M. J. Cafarella, and H. Jagadish. Sample-driven schema mapping. In *Proc. SIGMOD*, pages 73–84, 2012.

[34] J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to rdf and owl. In *Proc. WWW, Lyon, France*, pages 649–658, 2012.

[35] J. F. Sequeda and D. P. Miranker. Ultrawrap: Sparql execution on relational data. Technical Report TR-12-10, University of Texas at Austin, 2012.

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

[37] S. Sorrentino, S. Bergamaschi, M. Gawinecki, and L. Po. Schema normalization for improving schema matching. In *Conceptual Modeling-ER*, pages 280–293. Springer, 2009.

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

[39] F. M. Suchanek, S. Abiteboul, and P. Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *Proc. VLDB*, 5(3):157–168, 2011.

[40] A. Tian, J. F. Sequeda, and D. P. Miranker. On ambiguity and query-specific ontology mapping. In *Proc. ISWC Workshop on Ontology Matching, poster*, Boston, US, November 2012.

[41] A. Tian, J. F. Sequeda, and D. P. Miranker. Queries, the missing link in automatic data integration. In *Proc. ISWC Posters and Demonstrations Track*, Boston, US, November 2012.

[42] S. Tirmizi, J. Sequeda, and D. Miranker. Translating sql applications to the semantic web. In *Database and Expert Systems Applications*, pages 450–464. Springer, 2008.

[43] H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information-a survey of existing approaches. In *IJCAI-01 workshop: ontologies and information sharing*, volume 2001, pages 108–117, 2001.

[44] Q. Zhong, H. Li, J. Li, G. Xie, J. Tang, L. Zhou, and Y. Pan. A gauss function based approach for unbalanced ontology matching. In *Proc. SIGMOD*, pages 669–680. ACM, 2009.