QODI: Query as Context in Automatic Data Integration

Aibo Tian, Juan F. Sequeda, and Daniel P. Miranker

Department of Computer Science, The University of Texas at Austin
Austin, Texas, USA
{atian,jsequeda,miranker}@cs.utexas.edu

Abstract. QODI is an automatic ontology-based data integration system (OBDI). 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. The mapping algorithm decomposes the query into a set of paths, and compares the set of paths with a similar decomposition of a source ontology.

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.

Keywords: QODI, Ontology-based data integration, Query-specific ontology mapping.

1 Introduction

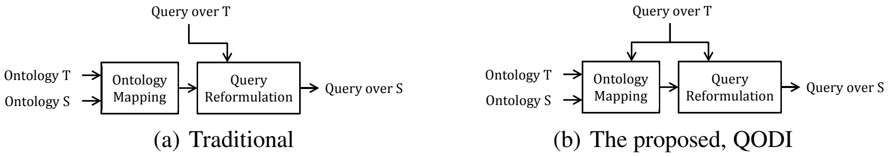
Web-wide integration of structured data is being enabled by the emerging Semantic Web protocols that specify uniform query interfaces to the databases included in the deep web [10]. These developments were recently boosted by W3C ratification of standards for publishing relational database content as RDF[1]. 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 systems (OBDI).

This paper details the mapping algorithms of Query-driven Ontology-based Data Integration (QODI). QODI is currently deployed as the mediator of a faceted search system over RNA databases[2]. 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, 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 [13,14].

[1] http://www.w3.org/2001/sw/wiki/RDB2RDF

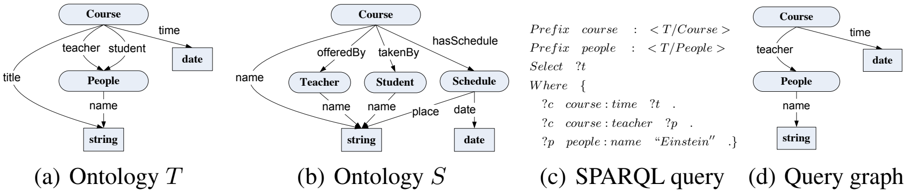
[2] http://ribs.csres.utexas.edu/ontoexplorer

H. Alani et al. (Eds.): ISWC 2013, Part I, LNCS 8218, pp. 624–639, 2013.



(a) Traditional (b) The proposed, QODI

Fig. 1. Diagram of OBDI systems with traditional and the proposed ontology mapping



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

(a) Ontology T (b) Ontology S (c) SPARQL query (d) Query graph

Fig. 2. Example ontologies and SPARQL query about the domain of course. Oval vertices represent classes, and rectangular vertices are datatypes. Edges represent object properties, or datatype properties. The SPARQL query asks for the time of any course taught by "Einstein".

In the typical organization of an OBDI system, ontology mapping is a separate and prerequisite step of query reformulation (see Figure 1(a)). Ontology matchers may be introduced to automatically determine corresponding entities [3,6]. In this paper, an entity refers to a class or a property. We tested AgreementMaker [5], one of the top finishers in 2010 Ontology Alignment Evaluation Initiative (OAEI) [1]. 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. Figure 2 shows a target ontology T, a source ontology S, and 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.

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 1(b)). 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 efficient 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:

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

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

2 Problem Definition

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

2.1 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 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 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 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 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 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 (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 2(c) is shown in Figure 2(d).

2.2 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 [13,14].

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 paper concerns ontologies that describe database content and queries that retrieve information from databases. Retrieving database data ultimately requires the rewriting of datatype properties.

2.3 Query-Specific Ontology Mapping

The following definitions define query-specific ontology mapping, which is the core problem of this paper. An ss-path correspondence records the mapping confidence between two ss-paths.

Definition 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 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 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 8 (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, and $\beta_{\Omega_{Q,\bar{G}}} = \max_{G \subseteq S} \beta_{\Omega_{Q,G}}$.*

3 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.

3.1 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 [5,12,6]. 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[3]). 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.

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

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 may not have a large difference. We use an exponential function to penalize the path length difference. The ss-path similarity measure, S_{SS}, is defined as,

$$S_{SS}(p,p') \;=\; S_C(p,p')^{\frac{1}{n_p}} \cdot S_D(p,p') \cdot S_L(p,p') \cdot e^{-\eta \,\cdot\, |\,|p|-|p'|\,|} \tag{2}$$

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 η 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.

3.2 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 $\bar{G}$.

$$\begin{aligned}\bar{G} &= \underset{\Omega_{Q,G}\in\mathcal{M}_Q}{\arg\max}\;\; \beta_{\Omega_{Q,G}} \\ &= \arg_{G\subseteq S}\,\{\prod_{c\in SOURCE_Q}\{\max_{c'\in SOURCE_G}\{ \\ &\qquad \prod_{p\in \text{SS-PATH-SET}_{Q,c,*}}\{\max_{p'\in \text{SS-PATH-SET}_{G,c',*}} S_{SS}(p,p')\}\}\}\}\end{aligned} \tag{3}$$

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

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 (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.

3.3 Solving the Maximization

Equation (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 [9]. 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, 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 $\mathcal{P}$.

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 each datatype property. Figure 3

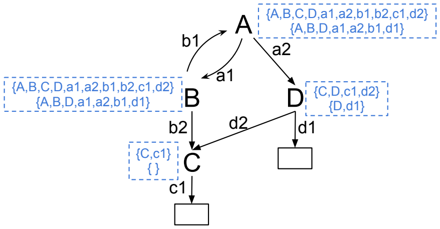


Fig. 3. An example ontology graph with reachable label sets. The dashed boxes around each non-sink vertex contain the reachable label sets through the two datatype properties $c1$ and $d1$. For example, the reachable label set from C through $c1$ is $\{C, c1\}$, and through $d1$ is empty.

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 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.

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:

$$h(p) = -\ S_C(r,p)^{\frac{1}{n_r}} \cdot S_D(r,p) \cdot \frac{\sum_{i=1}^{m} \min(\mathbf{f}_i(r), \mathbf{f}_i(p) + \bar{\mathbf{f}}_i(r,p,L_{x,e}))}{\sum_{i=1}^{m} \mathbf{f}_i(r) + \mathbf{f}_i(p) - \min(\mathbf{f}_i(r), \mathbf{f}_i(p))} \cdot e^{-\eta \ \cdot \ g(r,p,l_{x,e})} \tag{4}$$

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 ith 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:

$$\bar{\mathbf{f}}_i(r,p,L_{x,e}) = \begin{cases} \max(\mathbf{f}_i(r) - \mathbf{f}_i(p),\ 0) & \text{, if } x \neq e \text{ and string } i \in L_{x,e} \\ 0 & \text{, otherwise} \end{cases} \tag{5}$$

$$g(r,p,l_{x,e}) = \begin{cases} \max(|p| + l_{x,e} - 1 - |r|,\ 0) & \text{, if } x \neq e \\ |\ |p| - |r|\ | & \text{, if } x = e \end{cases} \tag{6}$$

Comparing (4) with (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 1 corresponds to the proof of admissibility of the heuristic in A* search.

Lemma 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 1. *When the search terminates, the path* $\bar{p}$ *is an optimal path.*

The proof of Theorem 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 $h(p)$ [9].

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 1.

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

3.4 Query Reformulation

The primary focus of this paper is mapping. Thus, we only 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 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. For the query in Figure 2, a path correspondence and the resulting translated triple patterns are:

$$\{\text{Course,teacher,People,name,string}\} = \{\text{Course,offeredBy,Teacher,name,string}\}$$

?c1 course:offeredBy ?c2 .   ?c2 teacher:name “Einstein” .

4 Experimental Setup

4.1 Test Sets

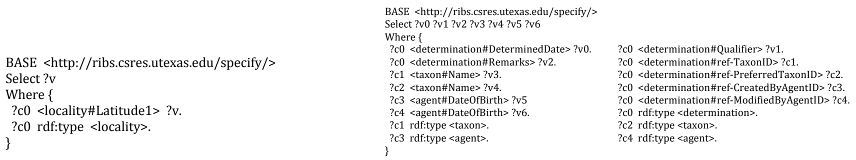
The test sets comprise three application domains: Life Science, Bibliography, and Conference Organization. The test cases include an ontology created by an international standards body, two ontologies created from direct mapping relational databases, and three ontologies used in OAEI [1].

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[4]. 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 [14]. Class hierarchies are manually added. DBLP ontology has 17 classes, and 51 properties. The Conference domain consists of the two ontologies from OAEI, SIGKDD and SOFSEM. We have made the test suite available on our website[5].

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. 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 concept, and a ClassAll query asks for all values of all attributes of a concept. For ontology T in Figure 2, a PathOnly query could ask for names of all students taking courses, and a ClassAll query could ask for titles, time, and names of all students of all courses. Figure 4 shows examples of real PathOnly and ClassAll queries generated for the Specify ontology, as well as the meaning of both queries.

[4] http://specifysoftware.org/

[5] http://ribs.csres.utexas.edu/qodi



```
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#Qualifier> ?v1.
 ?c0  <determination#Remarks> ?v2.              ?c0  <determination#ref-TaxonID> ?c1.
 ?c1  <taxon#Name> ?v3.                         ?c0  <determination#ref-PreferredTaxonID> ?c2.
 ?c2  <taxon#Name> ?v4.                         ?c0  <determination#ref-CreatedByAgentID> ?c3.
 ?c3  <agent#DateOfBirth> ?v5                   ?c0  <determination#ref-ModifiedByAgentID> ?c4.
 ?c4  <agent#DateOfBirth> ?v6.                  ?c0  rdf:type <determination>.
 ?c1  rdf:type <taxon>.                         ?c2  rdf:type <taxon>.
 ?c3  rdf:type <agent>.                         ?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.

Fig. 4. Real SPARQL queries generated for Specify ontology in the experiments

4.2 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 [7]. 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 [15]. The third is AgreementMaker configured as detailed in OAEI 2010 conference track [5].

4.3 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 9 (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.*

$$valid\_rate = \frac{\#\text{ queries with complete q-mappings generated}}{\#\text{ queries}} \tag{7}$$

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.

$$query\_precision = \frac{\#\text{ correctly mapped queries}}{\#\text{ queries}} \tag{8}$$

$$path\_precision = \frac{\sum_q \text{percentage of correctly mapped ss-paths in } q}{\#\text{ queries}} \tag{9}$$

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 10 (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 $S_D(p_t, p_s)$.*

$ambiguous\_rate$ is a measure of the proportion of queries that have datatype ambiguous q-mappings.

$$ambiguous\_rate = \frac{\#\text{ queries with datatype ambiguous q-mapping}}{\#\text{ queries}} \tag{10}$$

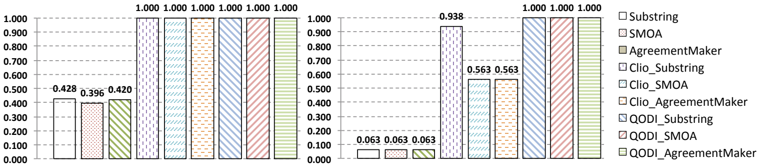
5 Experimental Results

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 5.3 discusses the accuracy using different η. Due to the space limit, only part of the results are reporting. Please refer to the technical report for all results [16].

5.1 Valid_rate

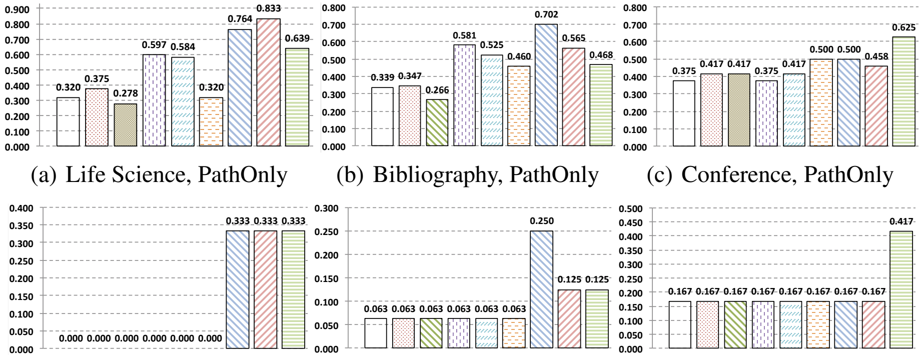
Figure 5 shows the valid_rate for Bibliography test set. Conference and Life Science test sets have similar numbers, and not reported here. The three methods of QODI achieve 100% valid_rate for all test sets. This is because QODI does not determine any entity mapping beforehand. Each path correspondence is assigned a confidence, and the mapped paths has the highest confidence.

The Clio baselines are able to generate complete mappings for the PathOnly query set but not all ClassAll queries. For some ClassAll queries, Clio cannot find a complete q-mapping if the mapped entities are incorrectly selected from ambiguous mappings.



(a) Bibliography, PathOnly (b) Bibliography, ClassAll (c) legend

Fig. 5. valid_rate for Bibliography test set. Higher number means better performance.



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

(d) Life Science, ClassAll (e) Bibliography, ClassAll (f) Conference, ClassAll

Fig. 6. query_precision for different test sets. Refer to Figure 5(c) for legend.

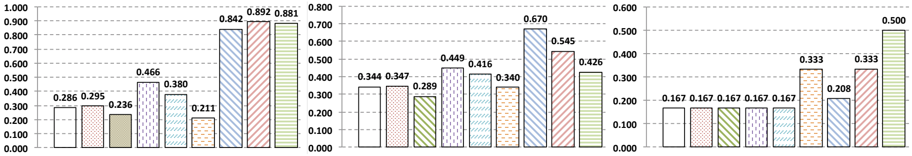
The comparison between QODI and Clio shows that disambiguation is important even for generating complete q-mappings regardless of correctness.

The ontology matching 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 between ontology matching baselines and Clio baselines demonstrates the importance of the missing mapping challenge.

5.2 Precision

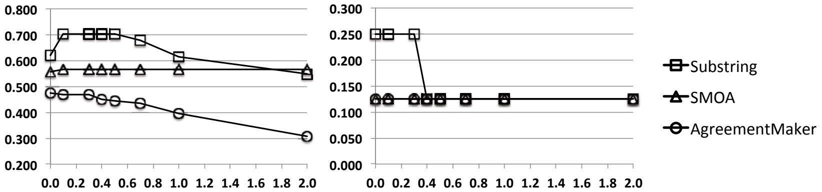
Figure 6 and 7 show the precisions of all methods. For all test sets, at least one QODI method dominates all baselines in terms of both precision measures. For ClassAll query sets, there are big gaps between QODI and all baselines. QODI is the only system that achieves non-zero query_precision for the 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 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.



(a) Life Science, ClassAll (b) Bibliography, ClassAll (c) Conference, ClassAll

Fig. 7. path_precision with ClassAll query sets for different test sets. Refer to Figure 5(c) for legend. path_precision for PathOnly query sets is the same as query_precision.



(a) Bibliography, PathOnly (b) Bibliography, ClassAll (c) legend

Fig. 8. query_precision of Bibliography test set when using different η (horizontal axis)

5.3 Parameter Tuning

Figure 8 shows the query_precision of Bibliography test set using different path length penalty parameter η. The results for Conference and Life Science sets are not reported due to space limit. 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 (2). Thus only the paths with the same lengths are considered as similar, ignoring the labels of the paths.

5.4 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 as shown in Table 1.

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

Table 1. The ambiguous_rate (row 2) and query_precision of queries with datatype ambiguous q-mapping (row 3, 4, 5) using SMOA 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.

	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**	-	-	-

Bibliography with UMBC as target ontology, with rates 0.242 and 0.177. We only report the query_precision using SMOA as matcher in Table 1. Other matchers show similar results. 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.

6 Related Work

Ontology matching has been well studied [2,6,3,8]. Many ontology matching systems compete in the OAEI [1], such as AgreementMaker [5] and RiMOM [12]. In the ontology matching world, most of systems focus on mapping between entities. In this paper, we define query-specific mapping for OBDI systems.

Clio, the state-of-the-art semi-automatic data integration and exchange system has close similarities with QODI [7]. Schema mapping in Clio is done in 2-steps: finding initial mappings between attributes; and associating mappings by logical inference through referential constraints. A semi-automatic OBDI system, Karma, is recently built to map structured data sources to ontologies [11]. For both Clio and Karma, the mapping is generated based on schemas alone. Neither system uses context from queries for resolving ambiguous mappings. Ontology based data access (OBDA) uses ontologies expressed in Description Logic as a conceptual view over data sources [4]. The mapping generated by QODI may be used for OBDA with proper representation.

7 Conclusions and Future Work

In this paper, we introduce query-specific ontology mapping, and implement an OBDI system, QODI. Departing from existing ontology matchers, QODI generates path correspondences, instead of entity correspondences, to facilitate query reformulation. The correspondences are discovered by heuristic search algorithm. A query is used as an input to the mapping to provide context for disambiguation and also reduce the mapping complexity.

Future work consists of at least three possible directions. First, the fundamental organization of QODI admits integration of user interaction for refinement. Second, path mappings can be accumulated over time as in pay-as-you-go systems. Third, new similarity measures and the relaxing of the basic assumptions can be explored.

Acknowledgements. This research is funded in part by the National Science Foundation grant IIS-1018554. Juan Sequeda was funded by the NSF Graduate Research Fellowship. We thank James Beach and Rodney Spears for providing a copy of the Specify database and sharing their knowledge of biodiversity informatics.

References

1. Ontology Alignment Evaluation Initiative, `http://oaei.ontologymatching.org/`
2. Bellahsene, Z., Bonifati, A., Rahm, E. (eds.): Schema Matching and Mapping. Springer (2011)
3. Bernstein, P., Madhavan, J., Rahm, E.: Generic schema matching, ten years later. In: Proc. VLDB, vol. 4(11) (2011)
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.: The MASTRO system for ontology-based data access. In: Semantic Web, vol. 2(1), pp. 43–53 (2011)
5. Cruz, I., Antonelli, F., Stroe, C.: Agreementmaker: efficient matching for large real-world schemas and ontologies. In: Proc. VLDB, vol. 2(2), pp. 1586–1589 (2009)
6. Euzenat, J., Shvaiko, P.: Ontology matching. Springer-Verlag New York Inc. (2007)
7. Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema mapping creation and data exchange. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Mylopoulos Festschrift. LNCS, vol. 5600, pp. 198–236. Springer, Heidelberg (2009)
8. Giunchiglia, F., Yatskevich, M., Shvaiko, P.: Semantic matching: Algorithms and implementation. In: Spaccapietra, S., et al. (eds.) Journal on Data Semantics IX. LNCS, vol. 4601, pp. 1–38. Springer, Heidelberg (2007)
9. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Systems Science and Cybernetics 4(2), 100–107 (1968)
10. Heath, T., Bizer, C.: Linked data: Evolving the web into a global data space. Synthesis Lectures on the Semantic Web: Theory and Technology 1(1), 1–136 (2011)
11. Knoblock, C.A., et al.: Semi-automatically mapping structured sources into the semantic web. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 375–390. Springer, Heidelberg (2012)
12. Li, J., Tang, J., Li, Y., Luo, Q.: Rimom: A dynamic multistrategy ontology alignment framework. IEEE Trans. Knowl. Data Eng. 21(8), 1218–1232 (2009)
13. Sequeda, J.F., Arenas, M., Miranker, D.P.: On directly mapping relational databases to RDF and OWL. In: Proc. WWW, Lyon, France, pp. 649–658 (2012)
14. Sequeda, J.F., Miranker, D.P.: Ultrawrap: SPARQL execution on relational data. Technical Report TR-12-10, University of Texas at Austin (2012)
15. Stoilos, G., Stamou, G., Kollias, S.D.: A string metric for ontology alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 624–637. Springer, Heidelberg (2005)
16. Tian, A., Sequeda, J.F., Miranker, D.P.: QODI: Query as context in automatic data integration. Technical Report TR-12-15, University of Texas at Austin (2012)