IncMap: Pay as you go Matching of Relational Schemata to OWL Ontologies

Christoph Pinkel[1], Carsten Binnig[2], Evgeny Kharlamov[3], and Peter Haase[1]

[1] fluid Operations AG, D-69190 Walldorf, Germany,
[2] University of Mannheim, D-68131 Mannheim, Germany,
[3] University of Oxford, Oxford, UK

Abstract. Ontology Based Data Access (OBDA) enables access to relational data with a complex structure through ontologies as conceptual domain models. A key component of an OBDA system are mappings between the schematic elements in the ontology and their correspondences in the relational schema. Today, in existing OBDA systems these mappings typically need to be compiled by hand, which is a complex and labor intensive task. In this paper we address the problem of creating such mappings and present *IncMap*, a system that supports a semi-automatic approach for matching relational schemata and ontologies. Our approach is based on a novel matching technique that represents the schematic elements of an ontology and a relational schema in a unified way. *IncMap* is designed to work in a query-driven, pay as you go fashion and leverages partial, user-verified mappings to improve subsequent mapping suggestions. This effectively reduces the overall effort compared to compiling a mappings in one step. Moreover, *IncMap* can incorporate knowledge from user queries to enhance suggestion quality.

1 Introduction

Effective understanding of complex data is a crucial task for enterprises to support decision making and retain competitiveness on the market. This task is not trivial especially since the data volume and complexity keep growing fast in the light of Big Data [1]. While there are many techniques and tools for scalable data analytics today, there is little known on how to find the *right* data.

Today, enterprise information systems of large companies store petabytes of data distributed across multiple – typically relational – databases, each with hundreds or sometimes even thousands of tables (e.g., [2]). For example, an installation of an SAP ERP system comes with tens of thousands of tables [3]. Due to the complexity of data a typical scenario for data analyses today involves a domain expert who formulates an analytical request and an IT expert who has to understand the request, find the data relevant to it, and then translate the request into an executable query. In large enterprises this process may iterate several times between the domain and IT experts, the complexity of data and other factors, and may take up to several weeks.

Ontology-based data access (OBDA) [4] is an approach that has recently emerged to provide semantic access to complex structured relational data. The

core elements of an OBDA system are an *ontology*, describing the application domain, and a set of declarative *mappings*, relating the ontological schema elements (e.g., names of classes and properties) with the relational schema elements (e.g., names of table and attributes) of the underlying data sources. Using the ontology and the mappings, domain experts can access the data directly by formulating queries in terms defined in the ontology that reflects their vocabulary and conceptualization. Using query rewriting techniques, the end-user queries are then translated into queries over the underlying data sources.

Today, most approaches for ontology-based data access focus on the definition of mapping languages and the efficient translation of high-level user queries over an ontology into executable queries over relational data [4,5]. These approaches assume that a declarative mapping of the schema elements of the ontology to the relational elements is already given. So far, in real-world systems [6,7] that follow the ontology-based data access principle, the mappings have to be created manually. The costs for the manual creation of mappings constitute a significant entry barrier for applying OBDA in practice.

To overcome this limitation we propose a novel semi-automatic schema matching approach and a system called *IncMap* to support the creation of mappings directly from relational schemata to ontologies.

We focus on finding one-to-one (direct) correspondences of ontological and relational schema elements, while we also work on extensions for finding more complex correspondences. In order to compute mapping suggestions *IncMap* uses a relational schema, an OWL ontology, a set of user conjunctive queries over the ontology, and user feedback as basic input.

The matching approach of *IncMap* is inspired by the Similarity Flooding algorithm of Melnik et al. [8] that works well for schemata that follow the same modeling principles (e.g., same level of granularity). However, applying the Similarity Flooding algorithm naively for matching schema elements of a relational schema to an OWL ontology results in rather poor quality of the suggested correspondences as we show in our experiments. A major reason is the impedance mismatch between ontologies and relational schemata: While ontologies typically model high-level semantic information, relational schemata describe the syntactical structure on a very low level of granularity.

The contributions of the paper are the following:

- In Section 3, we propose a novel graph structure called *IncGraph* to represent schema elements from both ontologies and relational schemata in a unified way. Therefore, we devise algorithms to convert an ontology as well as a relational schema into their unified *IncGraph* representation. We also briefly discuss techniques to further improve *IncGraph*.
- In Section 4, we present our matching algorithm that we use for matching *IncGraph*s. Its most prominent feature is that *IncMap* can produce the mapping incrementally, query by query. While the original Similarity Flooding algorithm generates correspondences for all schema elements, *IncMap* supports a *pay as you go* matching strategy. For each query we produce only required mappings. *IncMap* leverages the structure of mappings from previ-

ous queries to improve suggestion quality. This effectively reduces the total effort for the user to verify mapping suggestions.
– Section 5 presents an experimental evaluation using different (real-world) relational schemata and ontologies. We see that even in the basic version of *IncMap*, the effort for creating a mapping is up to 20% less than using the Similarity Flooding algorithm in a naive way. In addition, the incremental version of *IncMap* can reduce the total effort by another $50\% - 70\%$.

2 Background

In this section we briefly introduce ontologies [9], relational schemata, and the Similarity Flooding algorithm [8].

Ontologies. An ontology $\mathcal{O}$ specifies a conceptualization of a domain in terms of classes and properties and consists of a set of axioms. Without explanation, ontologies in this paper are OWL ontologies and we will use the following OWL constructs: object and data properties P, and domains $\texttt{Domain}(P)$ and ranges $\texttt{Range}(P)$ of properties. We denote with $\texttt{Class}(\mathcal{O})$ and $\texttt{Property}(\mathcal{O})$ the sets of class and property names, respectively, occurring in the ontology $\mathcal{O}$. For a given ontology $\mathcal{O}$, with $C \in \texttt{Domain}(P)$ we denote the fact that one can derive from $\mathcal{O}$ that the class name C is a domain of the property P. Also, $C' \in \texttt{Range}(P)$ denotes the fact that C' is a range of P and it is derivable from $\mathcal{O}$.

Relational Schemata. A relational schema $\mathcal{R}$ defines a set of relations (tables) T, where each table defines a set of columns c. We also assume that a schema contains foreign keys k that define references between tables.

Similarity Flooding Algorithm. The Similarity Flooding algorithm matches a given schema $\mathcal{S}$ with a schema $\mathcal{S}'$. In the first step, directed labeled graphs $\mathcal{G}(\mathcal{S})$ and $\mathcal{G}(\mathcal{S}')$ are constructed from $\mathcal{S}$ and $\mathcal{S}'$, where the nodes represent the schema elements, and the edges with labels define relationships between the schema elements. There is no exact procedure to construct the graphs from the schemata given in [8]. Thus, the Similarity Flooding algorithm is open for any graph construction process. The second step in the algorithm is to merge $\mathcal{G}(\mathcal{S})$ and $\mathcal{G}(\mathcal{S}')$ into one graph, a so-called pairwise connectivity graph `PCG`. Intuitively, each node of the `PCG` is a pair of nodes, and represents a potential match between schema elements of $\mathcal{S}$ and $\mathcal{S}'$. Then, the `PCG` is enriched with inverse edges and edge weights (propagation coefficients), where the value of the weights is based on the number of outgoing edges with the same label from a given node. This graph is called the induced propagation graph `IPG`. The final step of the algorithm is a fix-point computation to propagate initial similarities by using the structural dependencies represented by the propagation coefficients. The fix-point computation termination is based either on threshold values or the number of iterations. The result is a ranked list of suggested mappings. We refer to [8] for further details.

Algorithm 1: *IncGraph* for constructing graphs from ontologies

INPUT : OWL ontology $\mathcal{O}$
OUTPUT: Graph $\mathcal{G} = (\mathcal{V}, \texttt{Lbl}_\mathcal{V}, \mathcal{E}, \texttt{Lbl}_\mathcal{E})$

Let $\mathcal{G} = (\mathcal{V}, \texttt{Lbl}_\mathcal{V}, \mathcal{E}, \texttt{Lbl}_\mathcal{E})$, $\mathcal{V} = \{n_\top\}$, $\texttt{Lbl}_\mathcal{V} = \{(n_\top, \top)\}$, $\mathcal{E} = \emptyset$, $\texttt{Lbl}_\mathcal{E} = \emptyset$;
foreach $C \in \texttt{Class}(\mathcal{O})$ **do** $\mathcal{V} := \mathcal{V} \cup \{n_C\}$ and $\texttt{Lbl}_\mathcal{E}(n_C) := C$
foreach $P \in \texttt{Property}(\mathcal{O})$ **do**
  $\mathcal{V} := \mathcal{V} \cup \{n_P\}$ and $\texttt{Lbl}_\mathcal{V}(n_P) := P$; Let $C \in \texttt{Domain}(P)$;
  if *P is an object property* **then**
    $\mathcal{E} := \mathcal{E} \cup \{(n_C, n_P)\}$ and $\texttt{Lbl}_\mathcal{V}((n_C, n_P)) :=$ `'ref'`;
    Let $C' \in \texttt{Range}(P)$;
    $\mathcal{E} := \mathcal{E} \cup \{(n_P, n_{C'})\}$ and $\texttt{Lbl}_\mathcal{V}((n_P, n_{C'})) :=$ `'ref'`;
  else if *P is a data property* **then**
    $\mathcal{E} := \mathcal{E} \cup \{(n_C, n_P)\}$ and $\texttt{Lbl}_\mathcal{E}((n_C, n_P)) :=$ `'value'`
return $\mathcal{G}$.

Algorithm 2: *IncGraph* for constructing graphs from relational schemata

INPUT : Relational Schema $\mathcal{R}$
OUTPUT: Graph $\mathcal{G} = (\mathcal{V}, \texttt{Lbl}_\mathcal{V}, \mathcal{E}, \texttt{Lbl}_\mathcal{E})$

Let $\mathcal{V} = \emptyset$, $\texttt{Lbl}_\mathcal{V} = \emptyset$, $\mathcal{E} = \emptyset$, $\texttt{Lbl}_\mathcal{E} = \emptyset$;
foreach *table T in $\mathcal{R}$* **do**
  $\mathcal{V} := \mathcal{V} \cup \{n_T\}$ and $\texttt{Lbl}_\mathcal{V}(n_T) := T$;
  foreach *column c in $\mathcal{R}$* **do**
    $\mathcal{V} := \mathcal{V} \cup \{n_c\}$ and $\texttt{Lbl}_\mathcal{V}(n_c) := c$;
    $\mathcal{E} := \mathcal{E} \cup \{(n_T, n_c)\}$ and $\texttt{Lbl}_\mathcal{E}((n_T, n_c)) :=$ `'value'`
    if *c has a foreign key k to some table T'* **then**
      $\mathcal{V} := \mathcal{V} \cup \{n_k\}$ and $\texttt{Lbl}_\mathcal{V}(n_k) := k$;
      $\mathcal{E} := \mathcal{E} \cup \{(n_T, n_k)\}$ and $\texttt{Lbl}_\mathcal{E}((n_T, n_k)) :=$ `'ref'`
      $\mathcal{E} := \mathcal{E} \cup \{(n_k, n_{T'})\}$ and $\texttt{Lbl}_\mathcal{E}((n_k, n_{T'})) :=$ `'ref'`
return $\mathcal{G}$.

3 The *IncGraph* Model

In this section, we describe the *IncGraph* model used by *IncMap* to represent schema elements of an OWL ontology $\mathcal{O}$ and a relational schema $\mathcal{R}$ in a unified way.

An *IncGraph* model is defined as directed labeled graph $\mathcal{G} = (\mathcal{V}, \texttt{Lbl}_\mathcal{V}, \mathcal{E}, \texttt{Lbl}_\mathcal{E})$. It can be used as input by the original Similarity Flooding algorithm (Section 2) or *IncMap*. $\mathcal{V}$ represents a set of vertices, $\mathcal{E}$ a set of directed edges, $\texttt{Lbl}_\mathcal{V}$ a set of labels for vertices (i.e., one label for each vertex) and $\texttt{Lbl}_\mathcal{E}$ a set of labels for edges (i.e., one label for each edge). A label $l_\mathcal{V} \in \texttt{Lbl}_\mathcal{V}$ represents a name of a schema element whereas a label $l_\mathcal{E} \in \texttt{Lbl}_\mathcal{E}$ is either "ref" representing a so called `ref`-edge or "value" representing a so called `val`-edge.

3.1 *IncGraph* Construction

The goal of the procedures for the basic construction is to incorporate explicit schema information from $\mathcal{O}$ and $\mathcal{R}$ into the *IncGraph* model. Incorporating implicit schema information is discussed in the next section.

Algorithm 1 creates an *IncGraph* model $\mathcal{G}$ for a given ontology $\mathcal{O}$. The algorithm constructs a vertex n_C for each class name $C \in \texttt{Class}(\mathcal{O})$ and a vertex

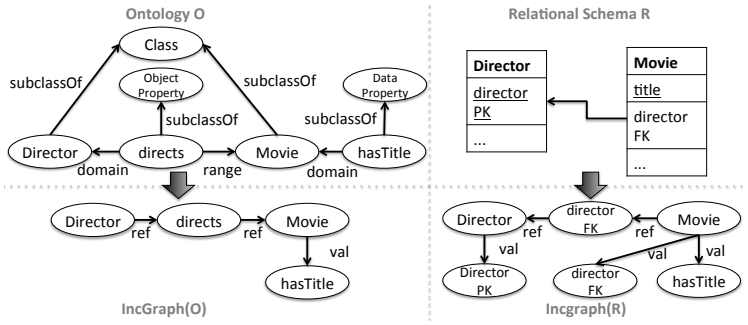


Fig. 1. *IncGraph* Construction Example

n_P for each property name $P \in \texttt{Property}(\mathcal{O})$ using the names of these ontology elements as label in $\texttt{Lbl}_\mathcal{V}$. Directed edges in the *IncGraph* model are created for each domain and range definition in $\mathcal{O}$. The labels $\texttt{Lbl}_\mathcal{E}$ for edges are either "ref" in case of an object property or "value" in case of a data property. For a domain definition in $\mathcal{O}$ the direction of the edge in $\mathcal{G}$ is from the node n_C representing the domain of P to the node n_P representing the property P. For a range definition the direction of the edge in $\mathcal{G}$ is from the node n_P representing object property to the node $n_{C'}$ representing the range of P (i.e., another class). If an object property in $\mathcal{O}$ has no range (respectively, domain) definition, then a directed labeled edge to a node $n_\top$ is added to explicitly model the most general range (respectively, domain), i.e., a top-level concept $\top$ like `Thing`.

Algorithm 2 creates a *IncGraph* model $\mathcal{G}$ for a given relational schema $\mathcal{R}$: The algorithm constructs a vertex n_T for each table and a vertex n_c for each column using the names of these schema elements as labels $\texttt{Lbl}_\mathcal{V}$. Directed edges with the label "value" are created from a node n_T representing a table to a node n_c representing a columns of that table. For columns with a foreign key k an additional node n_k is created. Moreover, two directed edges with the label "ref" are added, which represent a path from node n_T to a node $n_{T'}$ representing the referenced table via node n_k.

Figure 1 shows the result of applying these two algorithms to the ontology $\mathcal{O}$ and the relational schema $\mathcal{R}$ in this figure. Both $\mathcal{O}$ and $\mathcal{R}$ describe the same entities *Directors* and *Movies* using different schema elements. The resulting *IncGraph* models of $\mathcal{O}$ and $\mathcal{R}$ represent the schema structure in a unified way.

3.2 *IncGraph* Annotations

IncGraph is designed to represent both relational schemata and ontologies in a structurally similar fashion because matching approaches such as ours work best when the graph representations on both the source and target side are as similar as possible. However, even in *IncGraph* structural differences remain due to the impedance mismatch and different design patterns used in ontologies and relational schemata, respectively.

We consider this issue by supporting *annotations* in *IncGraph*. Annotations basically are additional `ref`-edges in either the source or target model that can be designed to bridge structural gaps for different design patterns or levels of

granularity. For instance, *shortcut edges* in the relational *IncGraph* model could represent a multi-hop join over a chain of relationship relations. Annotations can be constructed by plug-ins during *IncGraph* construction.

We plan to evaluate the opportunities of different kinds of annotations in future work.

4 The *IncMap* System

In this section, we present our matching approach and system called *IncMap*. *IncMap* takes a source and target *IncGraph* as input, i.e., the *IncGraph*s produced for a relational schema and for an ontology as described in Section 3.

4.1 Overview of *IncMap*

In its basic version, *IncMap* applies the original Similarity Flooding algorithm (with minor adaptions) and thus creates initial mapping suggestions for the *IncGraph* of an ontology $\mathcal{O}$ and a relational schema $\mathcal{R}$. In its extended version, *IncMap* activates inactive `ref`-edges before executing the Similarity Flooding algorithm to achieve better mapping suggestions.

Another extension is the incremental version of *IncMap*. In this version the initial mapping suggestions are re-ranked by *IncMap* in a semi-automatic approach by including user feedback. Re-ranking works iteratively in a query-driven fashion thus increasing the quality of the suggested mappings. In each iteration, *IncMap* applies a version of the Similarity Flooding algorithm (as described before). However, in addition between each iteration user feedback is incorporated.

The idea of user feedback is that the user confirms those mapping suggestions of the previous iteration, which are required to answer a given user query over ontology $\mathcal{O}$. Confirmed suggestions are used as input for the next iteration to produce better suggestions for follow-up queries. This is in contrast to many other existing approaches (including the original Similarity Flooding algorithm) that return a mapping for the complete source and target schema only once.

IncMap is designed as a framework and provides different knobs to control which extensions to use and within each extension which concrete variants to choose (e.g., to select a concrete strategy for activating inactive edges). The goal of this section is to present *IncMap* with all its variants and to show their benefits for different real-world data sets in our experimental evaluation in Section 5. A major avenue of future work is to apply optimization algorithms to find the best configurations of *IncMap* for a given ontology $\mathcal{O}$ and schema $\mathcal{R}$ automatically by searching the configuration space based on the knobs presented before.

4.2 Basic Matching in *IncMap*

As already mentioned, in the basic version of *IncMap*, we simply apply the Similarity Flooding algorithm for the two *IncGraph*s produced for a relational schema $\mathcal{R}$ and for an ontology $\mathcal{O}$ similar to the process as described in Section 2.

As a first step, *IncMap* generates the `PCG` (i.e., a combined graph which pairs similar nodes of both input *IncGraph*s) using an initial lexical matching, which

supports interchangeable matchers as one knob for configuration. One difference is the handling of inactive `ref`-edges in the input *IncGraph*s. For inactive `ref`-edges, which are not handled in the original Similarity Flooding, we apply the following rule when building the `PCG`: if an edge in the `PCG` refers to at least one inactive `ref`-edge in one of the *IncGraph* models, it also becomes inactive in the `PCG`.

In addition, other than in the original Similarity Flooding approach, where propagation coefficients for the `IPG` are ultimately determined during graph construction, our propagation coefficients can be calculated several times when the graph changes with the activation and deactivation of edges. Also, propagation coefficients in *IncMap* are modular and can be changed. In particular, a new weighting formula supported by *IncMap* considers the similarity scores on both ends of an edge in the `IPG`. The intuition behind this is that a higher score indicates better chances of the match being correct. Thus, an edge between two matches with relatively high scores is more relevant for the structure than an edge between one isolated well-scored match and another with a poor score. For calculating the weight $w(e)$ of a directed edge $e = (n_1, n_2)$ from n_1 to n_2 in the `IPG` where l is the label of the edge, we currently use two alternatives:

- *Original Weight as in [8]*: $w(e) = 1/out_l$ where out_l is the number of edges connected to node n_1 with the same label l
- *Normalized Similarity Product*: $w(e) = (score(n_1) * score(n_2))/out_l$.

4.3 Extended *IncMap*: Iterative User Feedback

Query-driven incremental mappings allow to leverage necessary user feedback after each iteration to improve the quality of mapping suggestions in subsequent iterations. One of the reasons why we have chosen Similarity Flooding as a basis for *IncMap* is the fact that user feedback can be integrated by adopting the initial match scores in an `IPG` before the fix-point computation starts.

Though the possibility of an incremental approach has been mentioned already in the Similarity Flooding paper [8], it so far has not been implemented and evaluated. Also, while it is simple to see *where* user feedback could be incorporated in the `IPG`, it is far less trivial to decide *which* feedback should be employed and *how* exactly it should be integrated in the graph. In this paper we focus on leveraging only the most important kind of user feedback, i.e., the previous confirmation and rejection of suggested mappings. We have devised three alternative methods how to add this kind of feedback into the graph.

First, as a confirmed match corresponds to a certain score of 1.0, while a rejected match corresponds to a score of 0.0, we could simply re-run the fix-point computation with adjusted initial scores of confirmed and/or rejected matches. We consequently name this first method *Initializer*. However, there is a clear risk that the influence of such a simple initialization on the resulting mapping is too small as scores tend to change rapidly during the first steps of the fix-point computation.

To tackle this potential problem, our second method guarantees maximum influence of feedback throughout the fix-point computation. Instead of just initializing a confirmed or rejected match with their final score once, we could repeat the initialization at the end of each step of the fix-point computation after

normalization. This way, nodes with definite user feedback influence their neighborhood with their full score during each step of the computation. We therefore call this method *Self-Confidence Nodes*. However, as scores generally decrease in most parts of the graph during the fix-point computation and high scores become more important for the ranking of matches in later fix-point computation steps, this method implies the risk of over-influencing parts of the graph. For example, one confirmed match in a partially incorrect graph neighborhood would almost certainly move all of its neighbors to the top of their respective suggestion lists.

Finally, with our third method, we attempt to balance the effects of the previous two methods. We therefore do not change a confirmed match directly but include an additional node in `IPG` that can indirectly influence the match score during the fix-point computation. We name this method *Influence Nodes*. By keeping the scores of those additional influence nodes invariant we ensure permanent influence throughout all steps of the fix-point computation. Yet, the influence node only indirectly affects the neighborhood of confirmed nodes through the same propagation mechanism that generally distributes scores through the graph.

5 Experimental Evaluation

The main goal of *IncMap* is to reduce the human effort for constructing mappings between existing relational database schemata and ontologies. Mapping suggestions are intended to be used only after they have been validated by a user. Thus, there are two relevant evaluation measures: first, the percentage of the mappings in the reference mappings that *can be represented* by *IncMap*. We specify this percentage for all reference mappings when introducing them. Certain complex mappings (e.g., mappings performing data transformations) cannot be represented by *IncMap*. These complex mappings are rare in all real-world reference mappings we used in this paper. The second and most important measure is the *amount of work* that a user needs to invest to transform a set of mapping suggestions into the correct (intended) mappings. As the latter is the most crucial aspect, we evaluate our approach by measuring the work time required to transform our suggestions into the existing reference mappings.

5.1 Relational Schemata and Ontologies

To show the general viability of our approach, we evaluate *IncMap* in two scenarios with fairly different schematic properties. In addition to showing the key benefits of the approach under different conditions, this also demonstrates how the impact of modular parameters varies for different scenarios.

IMDB and Movie Ontology. As a first scenario, we evaluate a mapping from the schema of well known movie database IMDB[4] to the Movie Ontology [10]. With 27 foreign keys connecting 21 tables in the relational schema and 27 explicitly modeled object properties of 21 classes in the ontology, this scenario is average

[4] http://www.imdb.com

in size and structural complexity. The reference mappings we use to derive correspondences for this scenario[5] has been made available by the -ontop- team [11]. A set of example queries is provided together with these reference mappings. We use these to construct annotations for user queries as well as to structure our incremental, query-by-query experiments. We extract a total of 73 potential correspondences from this mapping, 65 of which can be represented by *IncMap* as mapping suggestions. This corresponds to 89% of the mappings that could be represented in *IncMap*.

MusicBrainz and Music Ontology. The second scenario is a mapping from the MusicBrainz database[6] to the Music Ontology [12]. The relational schema contains 271 foreign keys connecting 149 tables, while the ontology contains 169 explicitly modeled object properties and 100 classes, making the scenario both larger and more densely connected than the previous one. Here we use R2RML reference mappings that have been developed in the project EUCLID.[7] As there were no example queries provided with the mapping in this case, we use example queries provided by the Music Ontology for user query annotations and to structure the incremental experiment runs.

For these reference mappings, two out of 48 correspondences cannot be represented as mapping suggestions by *IncMap* as they require data transformations. This corresponds to 95.8% of the mappings that could be represented in *IncMap*.

5.2 Work Time Cost Model

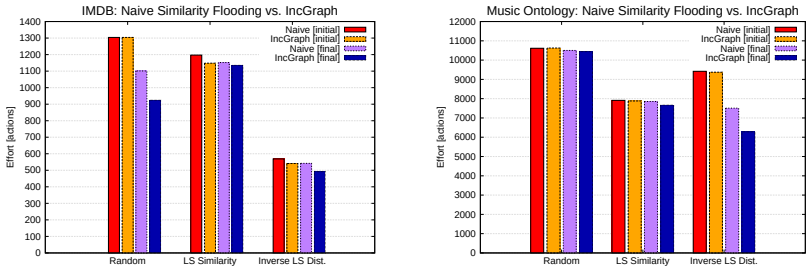
We evaluate our algorithms w.r.t. reducing *work time* (human effort). As the user feedback process always needs to transform mapping suggestions generated by *IncMap* into the correct mappings (i.e. to achieve a precision and recall of 100%), the involved effort is the one distinctive quality measure. To this end, we have devised a simple and straightforward work time cost model as follows: we assume that users validate mappings one by one, either accepting or rejecting them. We further assume that each validation, on average, takes a user the same amount of time $t_{validate}$. The costs for finding the correct correspondence for any concept in this case is identical with the rank of the correct mapping suggestion in the ranked list of mapping suggestions for the concept times $t_{validate}$.

As *IncMap* is interactive by design and would propose the user one mapping suggestion after another, this model closely corresponds to end user reality. We are aware that this process represents a simplification of mapping reality where users may compile some of the mappings by other means for various reasons. Nevertheless, this happens in the same way for any suggestion system and therefore does not impact the validity of our model for the purpose of comparison.

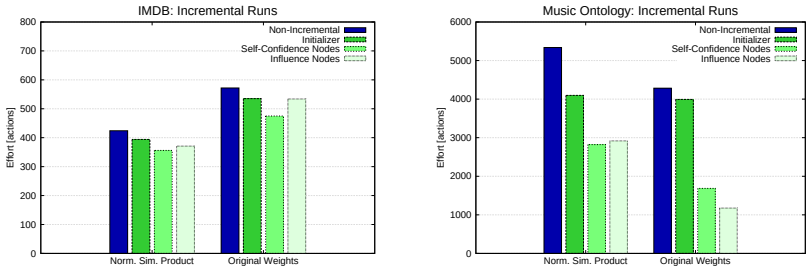
5.3 Experimental Evaluation

Experiment 1 – Naive vs. IncGraph*.* In our first experiment we compare the effort required to correct the mapping suggestions when the schema and ontol-

[5] https://babbage.inf.unibz.it/trac/obdapublic/wiki/Example_MovieOntology
[6] http://musicbrainz.org/doc/MusicBrainz_Database
[7] http://euclid-project.eu



(a) Naive vs. *IncGraph*



(b) Incremental Evaluation

Fig. 2. Experimental Evaluation

ogy are represented naively, or as *IncGraph*s. Additionally, we vary the lexical matcher used for the initial mapping between randomly assigned scores (minimal base line), Levenshtein similarity and inverse Levenshtein distance. Figure 2(a) shows that *IncGraph* in all cases works better than the naive approach. As *IncMap* reliably improves the mapping for all configurations, it also underlines the ability of *IncMap* to operate in a stable manner with different initial matchers.

Experiment 2 – Incremental Mapping Generation. Finally, we evaluated the best previous configurations incrementally, i.e., leveraging partial mappings. Figure 2(b) illustrates the effects on the total effort. We show total effort for all three incremental methods, for different propagation coefficients. Most significantly, incremental evaluation reduces the overall effort by up to $50\% - 70\%$. More specifically, Self-Confidence Nodes and Influence nodes work much better than the naive Initializer approach.

6 Related Work

Many existing mapping systems rely on two-step mapping procedures: They employ lexical similarity of terms together with structural similarity of the structures ([13,14,15] or [16,17] for surveys). A very few of them rely on variations of Similarity Flooding to perform the latter task. However, to the best of our knowledge, all of these approaches focus on ontology-to-ontology rather than relational schema-to-ontology mappings. RiMOM [18] performs a multi-strategy mapping discovery between ontologies and performs mappings using a variant of the Similarity Flooding algorithm, while it relies on structural similarities of ontologies derived from sub-class and sub-property relationships, rather than connectivity

of classes via properties as we do in order to get a better alignment of relational schemata and ontologies. In Yamm++ [19] the authors used Similarity Flooding and exploit both sub-class and sub-property relationships, and domain and ranges of ontologies, while they did it in a naive way which, as our experimental results showed, does not give good results for relational schemata-to-ontology mappings. Moreover, they use Similarity Flooding to obtain new mappings on top of the ones obtained via linguistic similarities, while we do not derive new mappings but refine the ranking over the linguistically derived ones. There are works on semi-automatic discovery of relational schema-to-ontology mappings, but they use approaches different from ours: For example, [20] transforms relational schemata and ontologies into directed labeled graphs respectively and reuse COMA [21] for essentially syntactic graph matching. Ronto [22] uses a combination of syntactic strategies to discover mappings by distinguishing the types of entities in relational schemata. The authors of [23] exploit structure of ontologies and relational schemata by calculating the confidence measures between virtual documents corresponding to them via the TF/IDF model. All these approaches do not incorporate implicit schema information and do not support an incremental mapping construction in the pay as you go fashion as *IncMap* does. Finally, [24] describes an approach to derive complex correspondences for a relational schema-to-ontology mapping using simple correspondences as input. This work is orthogonal to the approach presented in this paper.

7 Conclusions and Outlook

We presented *IncMap*, a novel semi-automatic matching approach for generating relational schema-to-ontology mappings. Our approach is based on a novel unified graph model called *IncGraph* for ontologies and relational schemata. *IncMap* implements a semi-automatic matching approach to derive mappings from *IncGraph*s using both lexical and structural similarities between ontologies and relational schemata. In order to find structural similarities *IncMap* exploits both explicit and implicit schema information. Moreover, *IncMap* allows to incorporate user queries and user feedback in an incremental way, thus, enabling a pay as you go fashion of the mapping generation. Our experiments with *IncMap* on different real-world relational schemata and ontologies showed that the effort for creating a mapping with *IncMap* is up to 20% less than using the Similarity Flooding algorithm in a naive way. The incremental version of *IncMap* reduces the total effort of mapping creation by another $50\% - 70\%$. As future work we plan to follow three lines: (1) add more implicit schema information (annotations) to the *IncGraph*s, (2) support more complex mappings in *IncMap*, and (3) devise a search strategy over the configuration space to auto-tune *IncMap*.

8 Acknowledgements

This work was supported by the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, the Optique project.

References

1. Beyer, M.A., Lapkin, A., Gall, N., Feinberg, D., Sribar, V.T.: 'Big Data' is Only the Beginning of Extreme Information Management. Gartner rep. G00211490 (2011)
2. Crompton, J.: Keynote talk at the W3C Workshop on Sem. Web in Oil & Gas Industry (2008) http://www.w3.org/2008/12/ogws-slides/Crompton.pdf.
3. SAP HANA Help: http://help.sap.com/hana/html/sql_export.html (2013)
4. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking Data to Ontologies. J. Data Semantics **10** (2008) 133–173
5. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The Combined Approach to Ontology-Based Data Access. In: IJCAI. (2011) 2656–2661
6. Hepp, M., Wechselberger, A.: OntoNaviERP: Ontology-Supported Navigation in ERP Software Documentation. In: International Semantic Web Conference. (2008)
7. Blunschi, L., Jossen, C., Kossmann, D., Mori, M., Stockinger, K.: SODA: Generating SQL for Business Users. PVLDB **5**(10) (2012) 932–943
8. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In: ICDE, IEEE Computer Society (2002)
9. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (2012) W3C Rec.
10. Bouza, A.: MO – The Movie Ontology, http://www.movieontology.org (2010)
11. Rodriguez-Muro, M., Calvanese, D.: High Performance Query Answering over DL-Lite Ontologies. In: KR. (2012)
12. Raimond, Y., Giasson, F., (eds): Music Ontology, www.musicontology.com (2012)
13. Jiménez-Ruiz, E., Grau, B.C.: LogMap: Logic-Based and Scalable Ontology Matching. In: International Semantic Web Conference (1). (2011) 273–288
14. Lambrix, P., Tan, H.: SAMBO – A system for aligning and merging biomedical ontologies. J. Web Sem. **4**(3) (2006) 196–206
15. Fagin, R., Haas, L.M., Hernández, M.A., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema Mapping Creation and Data Exchange. In: Conceptual Modeling: Foundations and Applications. (2009) 198–236
16. Shvaiko, P., Euzenat, J.: Ontology Matching: State of the Art and Future Challenges. IEEE Trans. Knowl. Data Eng. **25**(1) (2013) 158–176
17. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. In: VLDB J. (2001) 334–350
18. Li, J., Tang, J., Li, Y., Luo, Q.: RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. IEEE Trans. Knowl. Data Eng. (2009) 1218–1232
19. Ngo, D., Bellahsene, Z.: YAM++: A Multi-strategy Based Approach for Ontology Matching Task. In: EKAW. (2012) 421–425
20. Dragut, E.C., Lawrence, R.: Composing Mappings Between Schemas Using a Reference Ontology. In: CoopIS/DOA/ODBASE (1). (2004) 783–800
21. Do, H.H., Rahm, E.: COMA – A System for Flexible Combination of Schema Matching Approaches. In: VLDB. (2002) 610–621
22. Papapanagiotou, P., Katsiouli, P., Tsetsos, V., Anagnostopoulos, C., Hadjiefthymiades, S.: Ronto: Relational to Ontology Schema Matching. In: AIS SIGSEMIS BULLETIN. (2006) 32–34
23. Hu, W., Qu, Y.: Discovering Simple Mappings Between Relational Database Schemas and Ontologies. In: ISWC/ASWC. (2007) 225–238
24. An, Y., Borgida, A., Mylopoulos, J.: Inferring Complex Semantic Mappings Between Relational Tables and Ontologies from Simple Correspondences. In: OTM Conferences (2). (2005)