CogMap: A Cognitive Support Approach to Property and Instance Alignment

Jan Nößner, David Martin, Peter Z. Yeh, and Peter F. Patel-Schneider(✉)

AI Research Group, Nuance Communications, Sunnyvale, CA 94085, USA
jan.noessner@gmail.com,
{david.martin,peter.yeh,peter.patel-schneider}@nuance.com

Abstract. The iterative user interaction approach for data integration proposed by Falconer and Noy can be generalized to consider interactions between integration tools (generators) that generate potential schema mappings and users or analysis tools (analyzers) that select the best mapping. Each such selection then provides high-confidence guidance for the next iteration of the integration tool. We have implemented this generalized approach in CogMap, a matching system for both property and instance alignments between heterogeneous data. The generator in CogMap uses the instance alignment from the previous iteration to create high-quality property alignments and presents these alignments and their consequences to the analyzer. Our experiments show that multiple iterations as well as the interplay between instance and property alignment serve to improve the final alignments.

1 Introduction

In recent years, companies have spent more and more effort in building knowledge graphs based on light-weight ontologies, which incorporate data from multiple heterogeneous sources (which we will henceforth call "information stores"). A key challenge of these efforts is determining the best alignment of the schema of a new store to the ontology of the knowledge graph, while minimizing the "manual" analytical effort required of a human knowledge engineer.

Most of the current ontology alignment systems, such as those evaluated recently in the annual ontology alignment evaluation initiative [1], have several limitations. Most of these alignment algorithms solve one integration problem (deriving a mapping between two ontologies) using a fully-automated, "one-shot" approach. Thus, they are often not able to improve by iterating over previous alignments. Partly for this reason, the results of fully automated algorithms are often error prone [32] and cannot be reliably used for high-quality data integration.

Currently much information to be integrated is obtained from non-ontological sources such as relational databases or XML documents. Classical ontology alignment systems are often not able to process this data [1]. To address this need, systems like OntoDB [20] and standards like D2RQ [3] have emerged. However,

M. Arenas et al. (Eds.): ISWC 2015, Part I, LNCS 9366, pp. 269–285, 2015.
DOI: 10.1007/978-3-319-25007-6_16

these solutions do not include semi-automated alignment algorithms which take instance information into account.

Our approach, implemented in the CogMap system, follows a cognitively-inspired, iterative approach. With multiple iterations the system is able to improve over time, since it builds on the results of previous iterations (or, in the case of the first iteration, seed queries given by the user). At each iteration, the results are augmented with new information that has been verified by a user or automated verification capability.

CogMap uses instance information to perform property alignment. While most state-of-the-art schema alignment algorithms do not take instance information into account, focusing exclusively on the alignment of classes and properties and mainly considering their labels or structural information [7,29], using instance matching has attended more and more attention over the last years [16].

CogMap explores instances by not only focusing on data properties but also taking object properties into account. In the case of databases, it follows foreign keys; with RDF information stores it explores sub-tags. To the best of our knowledge, there exists no other approach which explores the space of potential mappings between information stores as we do.

CogMap is not restricted to the alignment of information based on formal ontologies. It also supports relational databases and XML documents, which can serve either as the source or the target of an alignment. In addition, CogMap allows support for other data formats to be added in a modular fashion.

2 Related Work

Many schema alignment systems have been developed in ontology matching. The development of these systems has largely been driven by the available benchmark datasets of the ontology alignment evaluation initiative. An overview of the current systems and their evaluation is given by Grau *et al.* [14]. The most important datasets, however, cover only a small problem space.

Although most ontology matching systems ignore instances, there exists a strand of literature which combines schema alignment and instance alignment [16]. Bilke and Naumann [2] developed an approach that first aligns instances and uses this information for schema alignment. Their evaluation is based on artificially populated data whereas we employ real-world data information stores like Freebase and DBpedia. Bilke *et al.* [1,26], Thor *et al.* [34], Gal [12], and Leme *et al.* [24] use instances to align schema and resolve conflicts. Another fully automated system that integrates both schema and instance alignment is Paris [33]. Its algorithms are, however, resource intensive, in some cases taking days to produce a solution. In contrast, the CogMap algorithms are much less resource intensive and can be run on a typical desktop computer. Wang *et al.* investigates the problem of having only a few non-overlapping instances by approaching the mapping problem as a classification problem. However, this approach is limited to mapping concepts and ignores properties. Duan *et al.* [5] use hashing techniques to speed up instance-based matching. Nunes *et al.* [30]

present an instance-based algorithm for complex data property matching. A prominent example is the system RiMOM, which dynamically combines several alignment strategies including instance alignment [25]. Due to its recent excellent achievements at the ontology alignment evaluation initiative, we chose this system for our evaluation.

To the best to our knowledge, none of these approaches is exploring object properties with an iterative cognitive support approach. QuickMig [4] is a migration tool for database systems which follows a semi-automated approach. However, it considers only exact value matches and their results are not used to improve the ongoing iterations.

A smaller number of systems utilize learning. A prominent example is SILK [17–19] which learns expressive linking rules by using genetic programming. However, its target user is a technical expert who can, e.g., analyse complex matching trees while CogMap focuses on domain experts by hiding technical complexity. LIMES [27] focuses on runtime improvements by using the triangle inequality. However, it does not allow a user-centric iterative approach. Furthermore, neither system is able to map data properties to object properties, which is required by the real-world datasets we examined. (See the algorithm section for details.)

Recently, the ontology alignment evaluation initiative initiated an interactive track which simulates interactive matching [31], where a human expert is involved to validate mappings found by the matching system. The client was modified to allow interactive matchers to ask an oracle, which emulates a perfect user. The interactive matcher can present a correspondence to the oracle, which then tells the user whether the correspondence is right or wrong. However, the initiative uses a dataset which does not contain any instance data and thus is not suitable for evaluating our approach. The two most successful participating systems 2014 were AML[11] with respect to gained f-measure due to the interactive approach and LogMap [22] with respect to efficiency (number of interactions required). We have included both systems in our evaluation.

Tools have been developed to support the alignment of databases to ontologies. One example is OnTop (ontop.inf.unibz.it), which provides a Protégé plug-in to facilitate the creation of integration rules. OnTop focuses on fast execution of already existing data integration rules, but not on the (semi-)automated construction of them. Furthermore, its target ontology is assumed to be small and to contain only schema information but no instance information. There have also been attempts to build graphical tools for supporting the user in data integration. Karma [23], for example, loads data from different information stores and uses instance information for schema alignment. However, its approach is different from our algorithm. Karma learns the general structure of fields based on previous alignments made whereas CogMap operates on instance information. Two disadvantages of Karma's approach are that it generally assumes that fields (e.g., ids) have similar structures in different datasets and its algorithms require a large amount of training data.

3 The Cognitive Support Approach

Researchers in ontology and schema matching have recently recognized the need for various types of cognitive support in aligning complex conceptual models [8,10]. Most approaches are based on advanced visualization of the models to be integrated and the mappings created by the user [13]. While the appropriate use of visualizations is known to be a key aspect for successful manual data integration, visualizations quickly reach their limits in the presence of very complex or very large models.

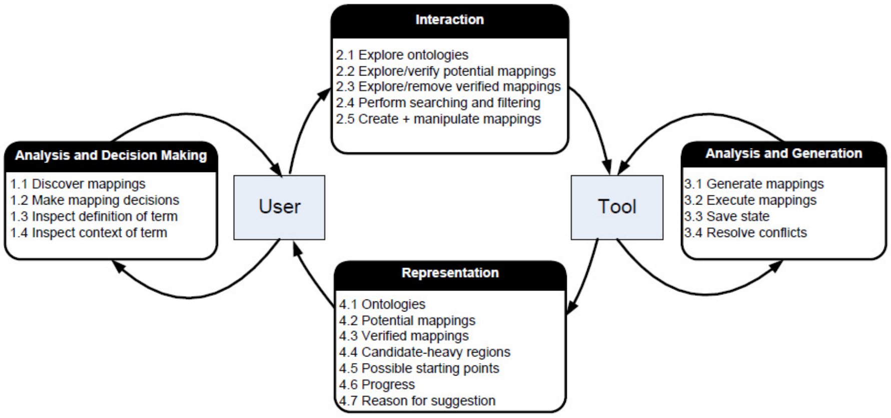
Interaction
2.1 Explore ontologies
2.2 Explore/verify potential mappings
2.3 Explore/remove verified mappings
2.4 Perform searching and filtering
2.5 Create + manipulate mappings
Analysis and Decision Making
1.1 Discover mappings
1.2 Make mapping decisions
1.3 Inspect definition of term
1.4 Inspect context of term
User
Tool
Analysis and Generation
3.1 Generate mappings
3.2 Execute mappings
3.3 Save state
3.4 Resolve conflicts
Representation
4.1 Ontologies
4.2 Potential mappings
4.3 Verified mappings
4.4 Candidate-heavy regions
4.5 Possible starting points
4.6 Progress
4.7 Reason for suggestion

Fig. 1. The cognitive support model for data integration by Falconer [9].

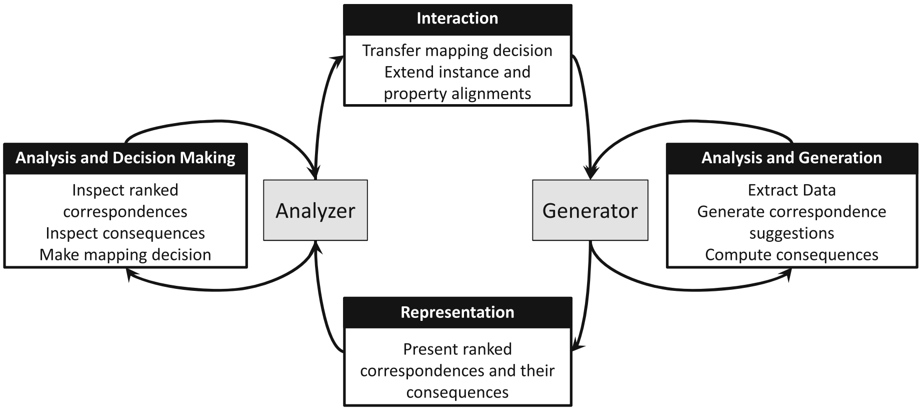
Interaction
Transfer mapping decision
Extend instance and property alignments
Analysis and Decision Making
Inspect ranked correspondences
Inspect consequences
Make mapping decision
Analyzer
Generator
Analysis and Generation
Extract Data
Generate correspondence suggestions
Compute consequences
Representation
Present ranked correspondences and their consequences

Fig. 2. Modified cognitive support Model as implemented in CogMap.

As a result, recent work has tried to go beyond pure visualization support to include cognitively efficient interaction strategies to support the user [9]. Falconer [8] proposed an interactive strategy for data integration where the integration task is distributed between the user and the tool (Figure 1). The MappingAssistant [32] project used a modified cognitive support model for data integration, focusing on detecting and correcting incorrect data integration rules.

In our implementation of the cognitive support model, which we call CogMap, we go one step further and allow the "user" to be either a human user or an intelligent automated agent. Thus, in our implementation of the cognitive support model (Figure 2), we distinguish between an *analyzer* and a *generator*, instead of a user and a tool.[1]

In each iteration, CogMap extracts data based on the results of previous iterations (or, in the first iteration, based on given seed queries), generates property correspondence suggestions, and computes the consequences for the top suggestions. These consequences are the instances that would be aligned if the analyzer selects (verifies) this property correspondence. Next, CogMap sends the ranked correspondences and their consequences to the analyzer. The analyzer then inspects this information and selects a correspondence. The selected property correspondence, and the resulting instance alignment, are added to the evolving results sets, which allows the system to improve its suggestions in subsequent iterations. The algorithm terminates when no properties remain to generate new correspondences, or no correspondence is selected by the analyzer.

CogMap is designed to cope with many different types of data stores, in many different formats. We currently have implemented support for RDF accessed via SPARQL, relational databases, and general XML based files. This list is easily extended by implementing our `Connector` interface.

4 Algorithm

The primary focus of the CogMap algorithm (Algorithm 1) is to construct property correspondences and instance alignments. An alignment (or mapping) consists of a set of correspondences. According to Euzenat *et al.* [6], a correspondence is a 4-tuple $\langle e_s, e_t, r, c\rangle$, where e_s and e_t are source and target entities, r is a semantic relation, and c is a confidence value (usually, $c \in [0, 1]$). Like most ontology alignment systems [1], we focus on equivalent relations $\langle e_s, e_t, \equiv, c\rangle$.

The algorithm can be split into three phases. The *data extraction* (lines 4-6) and *data exploration* (lines 13-16) phases are only executed in the first iteration ($i = 0$). The *alignment generation and selection* (lines 7-12) phase is repeated until no more correspondences are found. This phase includes the decision making of the analyzer. The following subsections will explain the phases in more depth.

[1] As stated, the analyzer can be a human. As this paper is about the effectiveness of the overall approach, we only use simple agents. Sophisticated automated agents or humans can utilize world knowledge or judgements to select bettter alignments instead of just picking the highest-scoring ones.

4.1 Data Extraction and Exploration

We adapt the terms data property, object property, and instances from the semantic web literature, extending them to databases and XML documents in an obvious fashion. For example, instances include database rows and XML nodes.

In the *data extraction* phase, we extract all data property names and their corresponding values for M instances into a source table T_s and a target table T_t (line 5). The left block (2nd column) of Table 1 illustrates the general form of T_s and T_t after extraction.

In the *data exploration* phase, we explore the search space by following object properties. In other words, for each object property *op* of an instance i, we examine the object which is the value of that property. Then, for each data

Algorithm 1 High-level algorithm of CogMap.

```
Input: $S_s, S_t$: Seed queries for source and target
Input: $M$: number of extracted instances of each information store (default: 5000)
Input: $k$: number of suggestions (default: 5)
Output: $\mathcal{X}, \mathcal{Y}$: Set of user-verified property correspondences and instance correspondences

getAlignments
 1: $\mathcal{X}, \mathcal{Y} \leftarrow \emptyset$
 2: $i \leftarrow 0$
 3: repeat
        ▷ Data Extraction
 4:     if i=0 then
 5:         $T_s, T_t \leftarrow$ Extract $M$ instances and their data properties and values based on
            seeds $S_s$ and $S_t$.
 6:     end if
        ▷ Alignment Generation and Selection
 7:     $X_i \leftarrow$ Compute top-$k$ property correspondence suggestions based on $T_s$, $T_t$ and
        $\mathcal{Y}$ (if not empty).
 8:     for every $x \in X_i$ do
 9:         $Y_x \leftarrow$ Compute instance alignment consequences for $x$ based on $T_s$, $T_t$ and $\mathcal{Y}$
            (if not empty).
10:     end for
11:     Analyzer selects the optimal $x \in X_i$ based on $X_i$ and $\{Y_x | x \in X_i\}$.
12:     add $x$ to $\mathcal{X}$, $\mathcal{Y} \leftarrow Y_x$.
        ▷ Data Exploration
13:     if i=0 then
14:         $I_s, I_t \leftarrow$ Extract source and target instance sets from instance alignment $\mathcal{Y}$.
15:         $T_s, T_t \leftarrow$ Extend tables by following the object-property assertions of $I_s$ and
            $I_t$.
16:     end if
17:     $i \leftarrow i + 1$
18: until No more suggestions found
```

Table 1. General form of source and target table. Initially, the direct data properties and the corresponding data are imported (left block). Second, and subsequent, steps further explore the data by including object properties (right blocks). (*dp*=data property, *op*= object property, i = instance, and v = value).

				op_a			op_b
	dp_1	$\cdots$	dp_n	$dp_{1,a}$	$\cdots$	$dp_{n,a}$	
i_1	$v_{1,1}$	$\cdots$	$v_{1,n}$	$v_{1,1,a}$	$\cdots$	$v_{1,n,a}$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
i_m	$v_{m,1}$	$\cdots$	$v_{m,n}$	$v_{m,1,a}$	$\cdots$	$v_{m,n,a}$	$\cdots$

property of that object, we add its value to the row for i. Thus, the right blocks of Table 1 ($op_a, op_b, \cdots$) are added during this phase. The reason this exploration happens at the end of the first iteration ($i = 0$, line 13-16) is that there may exist many object properties to follow. This often leads to a large amount of data. Thus, the idea is to restrict the exploration to the smaller instance sets I_s and I_t. These instance sets are extracted from the first instance alignment $\mathcal{Y}$ (line 15) such that $I_s = \{e_s | \langle e_s, e_t, \equiv, c\rangle \in \mathcal{Y}\}$ and $I_t = \{e_t | \langle e_s, e_t, \equiv, c\rangle \in \mathcal{Y}\}$. Then, CogMap only follows the object properties for the instance sets I_s and I_t which are usually much smaller.

For RDF repositories we utilize SPARQL queries to access data. We do not rely on the completeness of domain and range restrictions for extracting properties, since they are often poorly defined (e.g., in DBpedia). Instead we take the distinct set of all properties of the relevant instances (those retrieved by S for extraction or those identified as values of object properties for expansion) as the relevant data properties. For relational data extraction, we just add the limit M to the seed SQL query S, execute the query, and store the result in table T. For exploration, we follow the foreign keys according to the definitions in the database schema. For XML files, we extract every attribute and every direct child node that has a primitive value from the initial XPath expression S. We store both attribute and child node values as data properties in Table 1. For the exploration phase, we inspect the children x of all non-primitive nodes. From these nodes, we again store the values of every attribute and every direct child node that has primitive values.

For some properties, a given instance may have multiple values. For simplicity, we consider only single values in this presentation. In practice, we have found the concatenation of multiple values to be effective. More sophisticated strategies will be developed in future work. On the other hand, there may exist properties and/or instances which have almost no assertions, especially in large RDF knowledge bases and XML documents. To ensure the effectiveness of the approach, those assertions might need to be ignored. To cope with that issue, CogMap has an optional parameter ϕ to filter instances and data properties with sparse value assertions.

4.2 Alignment Generation and Selection

The goal of CogMap is to establish instance correspondences and correspondences between data and object properties. In doing so, the space of ontology elements (in RDF stores) or schema elements (in relational and XML stores) that CogMap considers is constrained by the seed information S. In addition, instance alignments are constrained by the domains and ranges of property alignments. For example, if we align a property $e_s =$`id` to a property $e_t =$`movieName`, then the resulting instance alignment is bounded by $S_s =$`movie` and $S_t =$`film` as domains. Thus, there is no need to consider possible correspondences involving other instances in the information stores.

In addition to the standard data-property to data-property, object-property to object-property, and instance to instance correspondences, we also support object- and data-property to data-property correspondences (Example: $e_t =$ `film/country/./name/` and $e_s =$ `movie/language`).

Line 7 of Algorithm 1 first computes the top-k property correspondence suggestions X_i. In the first iteration ($i = 0$), CogMap uses all available instance data since no instance alignment exists yet ($\mathcal{Y} = \emptyset$). For computational reasons,

Table 2. Selected implemented components.

	Aggregators
Unions or joins of sets of correspondences. Average, maximum, or multiples of confidence values if correspondences share the same source entity e_s and target entity e_t.	
Name	Description **Filters**
TopKFilter	Returns the top-k correspondences with the highest confidence value c.
OneToOneFilter	Returns a functional one-to-one alignment. We implemented a greedy strategy. First, it orders the correspondences in descending order. Then, it traverses through the list and drops all correspondences whose entities e_s or e_t have been already matched.
Name	Description **Property Matchers**
PropertyNameMatcher	Matches properties according to their name.
ValueLengthMatcher	Matches properties p_1 and p_2 with close average value
DistinctValueMatcher	length / close percentages of distinct row entries l_1 and l_2. The similarity is computed with $Min(l_1, l_2)/(Max(l_1, l_2)$.
InstanceBasedMatcher	If instance alignments $\mathcal{Y} = \emptyset$, we align properties by concatenating all values of all instances for each property and compute the string similarity. If $\mathcal{Y} \neq \emptyset$, we compute the property similarities for every instance pair in $\mathcal{Y}$ separately and average over the results.
	Instance Matchers
Align instances by concatenating every value for every property and computing their string similarity. If a specific property p is given, consider only values of that property. If an instance alignment $\mathcal{Y}$ is given, traverse through that alignment and update the similarities based on the string values of all the given property value(s).	

we use an implicit cutoff at this initial stage. In the following iterations, we can improve the suggestions by considering the instance correspondences from the previous iteration and comparing the property values only for the instance pairs in $\mathcal{Y}$. In these iterations, we do not apply any threshold but rank the correspondences at the end.

Then, we compute the consequences Y_x for the top-k suggestions X_i (line 8-10). That is, for each of those suggested property correspondences, we compute the instance alignment that will result if this correspondence is selected by the analyzer. Initially ($i = 0$), all source instances are compared against all target instances. In following iterations the instance alignment $\mathcal{Y}$ from the previous iteration is used to compute the new alignment. The threshold applied for the instance alignment equals the confidence value c of $\langle e_1, e_2, \equiv, c\rangle \in X_i$.

The value of k is relatively unimportant here. As long as a correct correspondence is in the top-k suggestions, the results of the approach will not be significantly affected. We have found that $k = 5$ is generally adequate to achieve this condition, and results in a reasonable load on human analysts. In an automated setting it would be easy to use a larger k, which might produce slightly better results at the price of of somewhat longer run times (to score the extra suggestions).

Finally, the analyzer selects the optimal $x \in X_i$ based on the suggestions X_i and the consequences $\{Y_x | \; x \in X_i\}$. As noted above, this selection can either be made by a human user or by an automated selection function that takes the confidence value and the suggestions into account. In this paper, we use only a simple automated agent that selects the best-scoring alignment. Employing humans or more-sophisticated agents would presumably produce better results, but then any advantage of the approach might only come from the intelligence in the human or agent—using a simple agent means that the benefits come from the overall alignment philosophy. (We plan to address elsewhere the user interface issues associated with supporting selections by a human.) After selection of x, we update the seed property alignment $\mathcal{X}$ and the seed instance alignment $\mathcal{Y}$ for the next iteration.

CogMap supports many different components to match instances and properties. Every `Filter`, `Aggregator`, and `Matcher` is a component. Each component has an `execute()` method, which returns a set of alignments.

The components are organized as a tree. The `Matchers` form the leaves. They take a source table T_s, a target table T_t, a set of previously verified property alignments $\mathcal{X}$ and a set of instance alignments $\mathcal{Y}$ from the previous iteration as input. An `Aggregator` executes every component in the list `cs` and aggregate the results. It might, for example, just take the maximum confidence value c of all correspondences with equal entities e_s and e_t. A `Filter` reduces the size of the alignment of its component after executing it. A simple filter might, for example, only return the correspondences for which confidence values c are above a certain threshold.

Table 2 lists a selection of implemented components and a short explanation of their functionality. CogMap incorporates mechanisms to deal with different

Table 3. Benchmark Statistics.

	Benchmark (1)		Benchmark (2)	
	DBpedia People	EPG Cast	Freebase Film	Fandango Movie
Format	RDF	RDB	RDF	XML
Data Properties	6	18	233	26
Object Properties	0	10	234	14
Instances	1,045,474	6,857	247,608	100,959

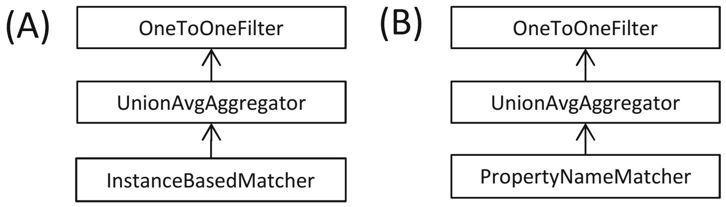

Fig. 3. Experiment Configurations.

date and number formats, which are omitted here for brevity, and easy interfaces to facilitate new component development. Figure 3 provides example trees built from these components.

5 Evaluation

We have selected two natural alignment tasks using real-world data, assessed the performance of CogMap on them benchmarks, and compared its performance with that of AML, LogMap, and RiMOM.

5.1 Benchmarks

The first benchmark aligns all people from DBpedia in FOAF format (wiki.dbpedia.org/Downloads39#persondata) with cast information of all programs playing on TV in the U.S. over a two week window from a commercial Electronic Program Guide (EPG) database. The second benchmark aligns Freebase films (www.freebase.com/film/film) with movie data from Fandango (www.fandango.com). Table 3 provides details on the number of instances and properties of each benchmark.

We designed and selected these benchmarks because existing state-of-the-art ontology alignment benchmarks, e.g., in the Ontology Alignment Evaluation Intiative campaigns[2], lack sufficient instance data, which are required by

[2] See oaei.ontologymatching.org

CogMap. Because of the size of these benchmarks, it was not possible to prepare in advance an official gold-standard. Instead, a human judge was employed to grade the correctness of the alignment results, which we discuss below.

5.2 Experiment Setup

We used the following experiment setup to answer three key questions:

- What is the impact of implementing a cognitive support model for alignment?
- What is the impact of using instance data for alignment?
- How general is our solution?

We first setup our solution, CogMap, using configuration (A) in Figure 3. A description of each component used in configuration (A) can be found in Table 2. CogMap analyzes the property correspondences, and selects the one with the highest confidence value to iterate on (see lines 11 and 12 of Algorithm 1).

We then created two variants of CogMap to answer the first two experimental questions above. We first created a variant—called InstMap—by ablating the cognitive support model used by CogMap. InstMap still uses instance data but does not iterate on the results to further improve alignment.

We also created a second variant—called Baseline—by ablating both the cognitive support model and the use of instance data. Baseline performs alignment using a property name matcher, but the other configuration components are the same (see configuration (B) in Figure 3).

Moreover, we selected three state-of-the-art ontology alignment systems [15] to compare CogMap against, in order to assess its practical impact. The three systems are AML [11], LogMap [22], and RiMOM [25]. AML is focused on computational efficiency and designed to handle very large ontologies. It is the leading system in the conference and anatomy tracks of the 2014 ontology alignment evaluation, in terms of f-measure. LogMap provides a scalable logical ontology alignment framework. RiMOM automatically combines multiple alignment strategies with the goal of finding the optimal alignment results. We selected these systems because they are the most established systems in the 2014 ontology alignment evaluation, and an executable version is available to the public.

Finally, we applied all systems above to both Benchmarks 1 and 2 to assess their generality, and hence answer the third experimental question. Unless otherwise noted, we set the number of instances to use from each benchmark to $M = 5000$, and the fraction of non-null values required for each property to $\phi = 0.1$. We also converted each benchmark into the RDF OWL syntax because many of the ontology matching systems compared cannot directly consume databases or XML files.

All experiments were run on a desktop PC with 4GB of RAM and an Intel i5 duo-core processor. We used Fast-Join [37] as the underlying matching algorithm for instances. Fast-Join combines both token-based similarity (Jaccard,

Table 4. Benchmark 1 results.

	Baseline	AML	LogMap	RiMOM	InstMap	CogMap
nDCG@3	0.38	0.76	0.38	0.38	0.76	1.00
nDCG@6	0.35	0.51	0.25	0.48	0.74	0.89
P@3	0.33	0.67	0.33	0.33	0.67	1.00
P@6	0.33	0.33	0.17	0.50	0.67	0.83
Runtime in sec	0.4	2.7	9.1	5.8	1.9	3.0

Cosine, or Dice) and string edit distance. Moreover, it is currently the fastest matching algorithm (see [21]), by implementing efficient pruning and hashing techniques, with soundness and completeness guarantees. This efficiency is required because of our large benchmarks, which make it infeasible to compare every source instance with every target instance.

The output of each system was graded by a human judge familiar with the data sources in each benchmark[3] using the metrics of *Precision at n (P@n)* and the *normalized (logarithmic) Discounted Cumulative Gain at n (nDCG@n)* [38] where n denotes that the top-n results. Precision P is defined as:

$$P = \frac{|\text{correct correspondences}|}{|\text{retrieved correspondences}|}$$

and $nDCG$ is defined as:

$$nDCG = \frac{rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log_2 i}}{(1 + \sum_{i=2}^{n} \frac{1}{log_2 i})}$$

where rel_i is 1 if the correspondence at position i is correct and 0 else. $nDCG@n$ gives more weight to correct correspondences that are ranked higher.

5.3 Results and Discussions

Tables 4 and 5 show the results for benchmarks 1 and 2, respectively. From these results, we observed that CogMap outperformed InstMap in most cases. We attribute this improvement to the only difference between the two systems: CogMap uses a cognitive support model while InstMap does not. Hence, the use of a cognitive support model has a positive impact on alignment results.

We also observed that InstMap outperformed Baseline in all cases. We attribute this improvement to the only difference between the two systems: the use of instance data. For example, Baseline could not correctly align the following data properties in benchmark 1 by matching just the names of these properties.

```
first_name ⇔ givenName
last_name ⇔ surName
full_name ⇔ name
```

[3] Determining the correctness of the correspondences produced by each system was simple for the human judge. We thus believe that the use of a human judge in this manner did not introduce any biases and did not affect the comparison.

Table 5. Benchmark 2 results.

	Baseline	AML	LogMap	RiMOM	InstMap	CogMap
nDCG@3	0.38	0.76	0.38	0.38	1.00	1.00
nDCG@6	0.25	0.60	0.49	0.38	0.90	1.00
nDCG@9	0.20	0.68	0.39	0.30	0.79	1.00
nDCG@12	0.17	0.68	0.38	0.25	0.69	0.85
P@3	0.33	0.67	0.33	0.33	1.00	1.00
P@6	0.17	0.50	0.50	0.33	0.83	1.00
P@9	0.11	0.67	0.33	0.22	0.67	1.00
P@12	0.08	0.67	0.33	0.17	0.50	0.75
Runtime in sec	2.6	7.0	21.7	33.2	20.5	29.1

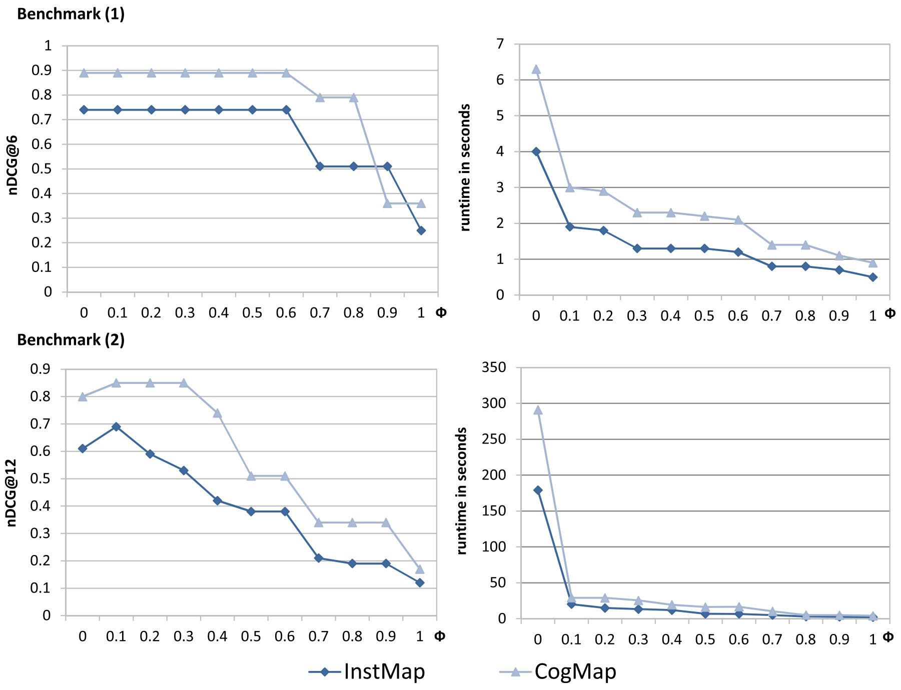

Fig. 4. Results for varying ϕ (number of non-null values) for CogMap and InstMap for both benchmarks. For high ϕ, *nDGC* and runtime decrease because fewer alignment candidates remain.

However, InstMap correctly found these alignments because of the overlap between the instances of these properties. Hence, these results show that the use of instance data also has a positive impact on performance.

Finally, we observed that CogMap out performed all three state-of-the-art ontology matching systems compared, i.e. AML, LogMap, and RiMOM. We attribute this improvement to the following factors:

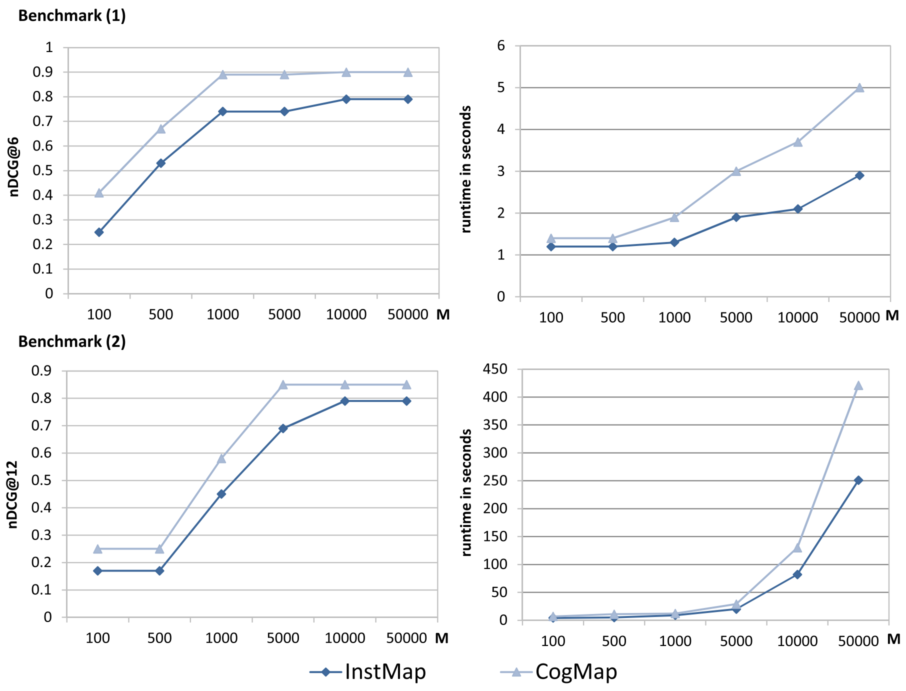

Fig. 5. Results for varying M (number of instances) for CogMap and InstMap for both benchmarks. The more instances included, the higher the overlap and hence better results ($nDCG$).

- CogMap uses instance data for alignment.
- CogMap uses an iterative cognitive model for alignment.
- CogMap can ignore rarely used properties by using the ϕ parameter.

Given the different characteristics of these two benchmark, the results above suggest the general utility of an alignment system like CogMap that combines a cognitive support model with the use of instance data. Moreover, the additional computation does not contribute to a significant increase in runtime. Across both benchmarks, CogMap had comparable (or better) runtime than the other state-of-the-art systems compared.

Figures 4 and 5 show the impact of varying ϕ (the fraction of non-null values required for each property) and M (the number of instances used) for CogMap and InstMap on both benchmarks. These results demonstrate the relative robustness of CogMap to these parameter settings compared to InstMap, and further demonstrate the positive impact of using a cognitive support model. For example, we observed on both benchmarks that the performance of CogMap only became negatively impacted for larger values of ϕ, which was in contrast to InstMap. Similarly, the performance of CogMap increased at a faster rate compared to InstMap as M was increased, and plateaued sooner than InstMap.

6 Conclusion and Future Work

This paper presents a cognitive based approach for aligning properties by taking instance information into account. The approach is implemented in the system CogMap which iteratively suggests property correspondences and their consequences in terms of instance alignments. In each round, the system is able to improve these alignments based on the user verifications of the previous round. Experiments show that the cognitive based approach outperforms both a baseline approach and the purely instance-based approach.

Currently, the system is restricted to aligning instances and properties. In future work, we will enable class alignments and complex matchings [36]. These complex matchings will be described using the R2RML standard (www.w3.org/TR/r2rml).

We will extend exploration of the knowledge sources. First, we will integrate object properties that are more than one hop away. This will require efficient pruning techniques to avoid an intolerable blowup of both data size and processing requirements. Second, we will use the organization of the knowledge structure (ontologies and schemas, when they are specified) to widen the search space by, e.g., exploring the data of the superclasses.

The knowledge structures will also help to improve the alignment itself by including ideas from [28,29]). Additionally, tree structure learning algorithms, inspired by [35], will be used to learn the optimal composition of matching trees.

Finally, we plan to explore the possibility of integration into Karma [23], which we believe would provide a suitable graphical user interface.

References

1. Bilke, A., Bleiholder, J., Naumann, F., Böhm, C., Draba, K., Weis, M.: Automatic data fusion with hummer. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 1251–1254. VLDB Endowment (2005)
2. Bilke, A., Naumann, F.: Schema matching using duplicates. In: Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, pp. 69–80. IEEE (2005)
3. Bizer, C., Seaborne, A.: D2RQ-treating non-RDF databases as virtual RDF graphs. In: Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), vol. 2004 (2004)
4. Drumm, C., Schmitt, M., Do, H.-H., Rahm, E.: Quickmig: automatic schema matching for data migration projects. In: Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, pp. 107–116. ACM (2007)
5. Duan, S., Fokoue, A., Hassanzadeh, O., Kementsietsidis, A., Srinivas, K., Ward, M.J.: Instance-based matching of large ontologies using locality-sensitive hashing. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 49–64. Springer, Heidelberg (2012)
6. Euzenat, J., Shvaiko, P., et al.: Ontology matching, vol. 18. Springer (2007)
7. Euzenat, J., Valtchev, P., et al.: Similarity-based ontology alignment in owl-lite. In: ECAI, vol. 16, p. 333 (2004)

8. Falconer, S.: Cognitive support for semi-automatic ontology mapping. Ph.D. Thesis, University of Victoria (2009)
9. Falconer, S., Noy, N.: Interactive techniques to support ontology matching. Schema Matching and Mapping, pp. 29–51 (2011)
10. Falconer, S.M., Storey, M.-A.D.: A cognitive support framework for ontology mapping. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 114–127. Springer, Heidelberg (2007)
11. Faria, D., Pesquita, C., Santos, E., Palmonari, M., Cruz, I.F., Couto, F.M.: The agreementmakerlight ontology matching system. In: Meersman, R., Panetto, H., Dillon, T., Eder, J., Bellahsene, Z., Ritter, N., De Leenheer, P., Dou, D. (eds.) ODBASE 2013. LNCS, vol. 8185, pp. 527–541. Springer, Heidelberg (2013)
12. Gal, A.: Interpreting similarity measures: bridging the gap between schema matching and data integration. In: IEEE 24th International Conference on Data Engineering Workshop, ICDEW 2008, pp. 278–285. IEEE (2008)
13. Granitzer, M., Sabol, V., Onn, K.W., Lukose, D., Tochtermann, K.: Ontology alignment - a survey with focus on visually supported semi-automatic techniques. Future Internet **2**(3), 238–258 (2010)
14. Grau, B.C., Dragisic, Z., Eckert, K., Euzenat, J., Ferrara, A., Granada, R. Ivanova, V., Jiménez-Ruiz, E., Kempf, A.O., Lambrix, P., et al.: Results of the ontology alignment evaluation initiative 2013. In: Proc. 8th ISWC Workshop on Ontology Matching (OM), pp. 61–100 (2013)
15. Grau, B.C., Dragisic, Z., Eckert, K., Euzenat, J., Ferrara, A., Granada, R., Ivanova, V., Jiménez-Ruiz, E., Kempf, A.O., Lambrix, P., et al.: Results of the ontology alignment evaluation initiative 2014. In: Proc. 8th ISWC Workshop on Ontology Matching (OM), pp. 61–100 (2013)
16. Isaac, A., Van Der Meij, L., Schlobach, S.: An empirical study of instance-based ontology matching. The Semantic Web. LNCS, vol. 4825, pp. 253–266. Springer, Heidelberg (2007)
17. Isele, R., Bizer, C.: Learning linkage rules using genetic programming. In: Proceedings of the Sixth International Workshop on Ontology Matching, pp. 13–24 (2011)
18. Isele, R., Bizer, C.: Learning expressive linkage rules using genetic programming. Proceedings of the VLDB Endowment **5**(11), 1638–1649 (2012)
19. Isele, R., Bizer, C.: Active learning of expressive linkage rules using genetic programming. Web Semantics: Science, Services and Agents on the World Wide Web **23**, 2–15 (2013)
20. Jean, S., Dehainsala, H., Xuan, D.N., Pierra, G., Bellatreche, L., Aït-ameur, Y.: OntoDB: it is time to embed your domain ontology in your database. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 1119–1122. Springer, Heidelberg (2007)
21. Jiang, Y., Li, G., Feng, J., Li, W.-S.: String similarity joins: An experimental evaluation. Proceedings of the VLDB Endowment **7**(8) (2014)
22. Jiménez-Ruiz, E., Cuenca Grau, B.: LogMap: logic-based and scalable ontology matching. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 273–288. Springer, Heidelberg (2011)

23. Knoblock, C.A., Szekely, P., Ambite, J.L., Goel, A., Gupta, S., Lerman, K., Muslea, M., Taheriyan, M., Mallick, P.: 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)
24. Leme, L.A.P.P., Casanova, M.A., Breitman, K.K., Furtado, A.L.: Instance-based OWL schema matching. In: Filipe, J., Cordeiro, J. (eds.) Enterprise Information Systems. LNBIP, vol. 24, pp. 14–26. Springer, Heidelberg (2009)
25. Li, J., Tang, J., Li, Y., Luo, Q.: Rimom: A dynamic multistrategy ontology alignment framework. IEEE Transactions on Knowledge and Data Engineering **21**(8), 1218–1232 (2009)
26. Naumann, F., Bilke, A., Bleiholder, J., Weis, M.: Data fusion in three steps: Resolving schema, tuple, and value inconsistencies. IEEE Data Eng. Bull. **29**(2), 21–31 (2006)
27. Ngomo, A.-C.N., Auer, S.: Limes-a time-efficient approach for large-scale link discovery on the web of data. Integration **15**, 3 (2011)
28. Niepert, M., Noessner, J., Meilicke, C., Stuckenschmidt, H.: Probabilistic-logical web data integration. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 504–533. Springer, Heidelberg (2011)
29. Noessner, J., Niepert, M., Meilicke, C., Stuckenschmidt, H.: Leveraging terminological structure for object reconciliation. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 334–348. Springer, Heidelberg (2010)
30. Pereira Nunes, B., Mera, A., Casanova, M.A., Fetahu, B., P. Paes Leme, L.A., Dietze, S.: Complex matching of RDF datatype properties. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013, Part I. LNCS, vol. 8055, pp. 195–208. Springer, Heidelberg (2013)
31. Paulheim, H., Hertling, S., Ritze, D.: Towards evaluating interactive ontology matching tools. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 31–45. Springer, Heidelberg (2013)
32. Stuckenschmidt, H., Noessner, J., Fallahi, F.: User-centric data integration with the mappingassistant. In: Cordeiro, J., Maciaszek, L.A., Filipe, J. (eds.) ICEIS 2012. LNBIP, vol. 141, pp. 323–339. Springer, Heidelberg (2013)
33. Suchanek, F.M., Abiteboul, S., Senellart, P.: Paris: Probabilistic alignment of relations, instances, and schema. Proceedings of the VLDB Endowment **5**(3), 157–168 (2011)
34. Thor, A., Kirsten, T., Rahm, E.: Instance-based matching of hierarchical ontologies. In: BTW, vol. 103, pp. 436–448 (2007)
35. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk-a link discovery framework for the web of data. LDOW **538** (2009)
36. Walshe, B., Brennan, R., O'Sullivan, D.: A comparison of complex correspondence detection techniques. In: OM (2012)
37. Wang, J., Li, G., Fe, J.: Fast-join: an efficient method for fuzzy token matching based string similarity join. In: 2011 IEEE 27th International Conference on Data Engineering (ICDE), pp. 458–469. IEEE (2011)
38. Wang, Y., Liwei, W., Li, Y., He, D., Chen, W., Liu, T.-Y.: A theoretical analysis of NDCG ranking measures. In: 26th Annual Conference on Learning Theory (2013)