Efficient and Effective On-line Learning-Based Instance Matching over Heterogeneous Data

Samur Araujo [#1], Duc Thanh Tran [*2], Arjen de Vries [#3]

[#] *Delft University of Technology, PO Box 5031, 2600 GA Delft, the Netherlands*
[1] s.f.cardosodearaujo@tudelft.nl
[3] a.p.devries@tudelft.nl

[*] *Karlsruher Institute of Technology, Germany*
[2] ducthanh.tran@kit.edu

***Abstract*—This document gives formatting instructions for authors preparing papers for publication in the Proceedings of an IEEE conference. The authors must follow the instructions given in the document for the papers to be published. You can use this document as both an instruction set and as a template into which you can type your own text.**

I. Introduction

Instance matching [1] refers to the problem of determining whether two descriptions are about the same real-world entity. Traditionally, research in this context was focused on the single-domain setting, where data come from the *same or similar datasets*. Basically, given the descriptions of entities available as records in databases, RDF descriptions on the Web, etc., the instance matching task breaks down to the core problems of (1) finding a suitable *instance representation* (i.e., selecting attributes and their values), (2) using this for evaluating different *similarity measures*, and (3) finally selecting the most similar ones according to a *threshold*.

For large datasets, the instance matching problem is typically solved in two steps, namely to find candidate matches first using a relatively simple but quick matching technique, and then to refine them using a more advanced but also more expensive technique. For the latter more sophisticated and *effective matching*, there are different techniques for learning the right combination of attributes, similarity measures and threshold to be used for computing and selecting the resulting matches [2], [3], [4]. Typically, the former, *candidate selection* resembles what is called *blocking* [5], [6], [7] in the single dataset settings (e.g. deduplication), which is described as the process of finding non-overlapping blocks of instances that share a similar key (a compact description of an instance), such that they can be compared between each other. For this type of key matching, indexes can be built to accelerate the process. In particular, fast *index lookups* can be performed to directly retrieve candidates from the index that share/overlap with the key of a given source instance.

In this work, we focus on the problem of candidate selection in the Web environment with *multiple heterogeneous datasets*, such as Linked Data. Here, heterogeneity particularly means that the datasets' schemas might vary. Existing techniques [6] assume instances are from the same or similar datasets such that the keys chose for one dataset can also be used to find candidates in the other datasets. This is however problematic in this setting because for a key such as one based on the values of the `name` attribute in the one dataset, there might not exist the `name` attribute, in the other datasets. Aiming to address this problem of heterogeneity, schema-agnostic candidate selection has been proposed recently [8]. It does not exploit attributes for building keys, but instead, simply treat instances as unstructured bags of tokens that can be extracted from the attribute values. That is, the key is simply composed of set of tokens that do not come with any attribute information. Instances are considered similar and form blocks when their keys overlap, i.e. they have some tokens in common. The drawback of the schema-agnostic approach is that it may build very ambiguous candidate sets, because a token may not be discriminative enough. For instance, the token "Paris", occurs in 7446 distinct instances at DBPedia[1] dataset.

We note that there exist only a few works [9], [8] that specifically address the problem of candidate selection over multiple heterogeneous datasets. To this end, this work provides the following contributions:

Effective Candidate Selection. Schema-agnostic candidate selection looses valuable attribute information and thus may yield too many candidates. On the other hand, using attributes to form keys requires schema matching[10] to find attributes in one dataset that correspond to (key) attributes in the other dataset [9]. In this work, we exploit attribute information for more effective candidate selection. However, we do not require attributes to complete match (e.g. "surname" and "family name") but employ a more

[1] http://dbpedia.org/

relaxed notion of *comparability* (e.g. we may map "drug-name" and "synomym"). In particular, even when they represent completely different attributes, some pairs of attributes among them might be more comparable than some other pairs. The comparability between attributes is then used to construct lookup queries on the target dataset. In order to obtain a high recall (i.e. retain correct candidates, true positives), which is the topmost goal in candidate selection, all comparable attributes have to be considered for constructing the queries. Among them, there exists a group that return the best set of candidates. We propose a *branch-and-bound based optimization framework* that iteratively search for this optimal query.

Efficient Candidate Selection. Using attribute-value pairs in the keys increases the selectivity of the resulting queries. However, since all comparable attributes are considered for achieving high recall, we obtain a large amount of candidate queries. To also optimize for efficiency, we incorporate the number of queries and query execution times into the branch-and-bound optimization so that it is geared not only towards queries that produce high quality results but also, towards executing a minimal number of time-efficient queries.

In the experiments, we show that compared to the schema-agnostic [8] and the schema-based [9] approaches, our approach yields superior results both in terms of efficiency and effectiveness in 96% of the cases. We compare our approaches on the context of instance matching itself, showing that it produces competitive results to other systems that approach the problem on off-line fashion.

Outline. This paper is organized as follows. After this introduction, we present the problem of candidate selection over multiple heterogeneous datasets, in the Section 2. In Section 3, we elaborate on our algorithm for building candidate selection queries. In Section 4, we present the algorithm to find candidate sets itself. Section 5 presents the experimental results, along with a comparative study against two known state-of-the-art approaches for candidate selection. Section 6 introduces the related work. Finally, Section 7 concludes this paper.

II. Related Work

Several *instance matching techniques* have been proposed to address both the efficiency and effectiveness of instance matching over single domain settings. Below we discuss state-of-the-art approaches, focusing on those that target the heterogeneous settings.

Blocking techniques [5] aims to make entity de-duplication more efficient by reducing the number of unnecessary comparisons between records. Based on a feature that is distinctive and can be processed efficiently (also called Blocking Key Value, BKV), instances are partitioned into blocks such that potentially similar instances (i.e. candidate results to be further refined) are placed in the same blocks. Examples of blocking techniques include the sorted neighbourhood approach and canopy clustering [11]. However, these techniques are focus on the single dataset settings, where the schema are homogeneous and the choice of a BKV is less problematic.

So far, only one unsupervised blocking technique has been explicitly designed to work in the heterogeneous Web setting, where to tackle the heterogeneity between schemas, schema attributes are ignored and the BKV is simply the set of all tokens that can be extracted from the instance data [8]. As it does not use attribute information in the BKV, we call it as schema-agnostic blocking. However, the limitation of this approach is known, it may yield too many candidates because it looses valuable attribute information, which would form more discriminative keys.

Candidate selection applies the same blocking principles to reduce the number of comparison on the task of instance matching. Analogue to blocks, similar instances are grouped together forming candidate sets; and blocking keys are called candidate selection schemes. Differently from the single dataset setting, in the multiple datasets setting, the alignment between candidate scheme from different sources is required, so that we can use information from the source scheme to select candidates based on the target scheme. Song et.al [9] focused on this setting and uses the attribute information, and its values, in the candidate schemes. Selected predicates are those with discriminativity and coverage above a threshold. However, the mapping of the candidate schemes is manually defined in their approach. ObjectCoref[12] applied a self-learning algorithm to solve instance matching itself. Although most of their principles can be used to generate candidate selection schemes, a direct translation of their algorithm to this task demands much more investigation. Differently from those approaches, we focus on generating both the candidate schemes and their alignments, in a unsupervised fashion.

Matching techniques are employed after candidate selection, or blocking, for disambiguating candidate matches, therefore, targeting the effectiveness of the match. Mainly, they are named *learning-based approaches* that can be further distinguished in terms of training data and degree of supervision, respectively (i.e. supervised, semi-supervised, unsupervised [13], [14], [15]). ObjectCoref[16] is a supervised approach that self-learns the discriminativeness of RDF properties. Then, matches are computed based on comparing values of a few discriminative properties. RIMON [17] is an unsupervised approach that firstly applies schema-based type of candidate selection to produce a set of candidate resources and then, uses a document-based

similarity metric (cosine similarity) for disambiguating candidate resources. Chaudhuri et.al [18] propose an supervised approach for learning the recording matching queries. Basically, it learns from a set of positive and negative matching, a matching scheme, including the similarity function and threshold. Then, these matching schemes can be used to select the correct matches from set of possible candidates. Following a similar idea, KnoFuss+GA [19] propose a online-learning system that can derive a matching scheme between heterogeneous sources.They do so by learning a fitness function for a genetic algorithm, which derives a suitable decision rule for a given matching task. Zhish.links [20] uses background knowledge to tune the matching scheme. Mainly, they manually define a string and geographic similarity measure to produce precise matches. As most of the approaches, they process the data locally, indexing it to speed up its access. Although, all those approaches proof to to be effective in the heterogeneous settings and some have target the efficiency part as well, none of them target instance matching over remote data endpoints, where efficiency is the main issue. The learning of matching scheme from the data over those remote endpoints are not straightforward. Differently from accessing the data off-line, in the remote scenario, the data have to be queried. The challenge is to efficiently integrate this process of querying those endpoints into the learning of the matching scheme, so that only the necessary and sufficient data to produce the correct matches are queried.

SERIMI [21] is the only unsupervised approach that tackles the problem of instance matching over remote endpoints. It learns the candidate selection scheme by querying the endpoints and applies two matchers to eliminate string level ambiguity (many instances with similar name on the same class) and class level ambiguity (many instances with the same name on different classes) on the candidate sets. However, it does not pay attention to executing efficiently the candidate selection scheme, only focusing on the effective part of the problem.

Differently, we target the efficiency and effectiveness of candidate selection over heterogeneous and remote data endpoints. We propose an ensemble of two matchers used for refined the candidate matches, and we show it efficiently produces competitive effective matches to those elaborated matching techniques.

III. Overview

In this section, we introduce the data model, discuss the problem, and finally, present a brief overview of existing solutions as well as our approach.

A. Data

We focus on RDF data and other types of data that can be modelled as graphs. Closely resembling the RDF data model, we employ a graph-structured data model where every dataset is conceived as a graph $G \in \mathbb{G}$ comprising a set of triples:

Definition 1 (Data Model): A dataset set is a graph G formed by a set of triples of the form (s, p, o) where $s \in U$ (called subject), $p \in U$ (predicate) and $o \in U \cup L$ (object). Here, U denotes the set of Uniform Resource Identifiers (URIs) and L the set of literals. Every literal $l \in L$ is a bag of tokens, $l = \{t_1, \ldots, t_i, \ldots, t_n\}$, drawn from the vocabulary V, i.e. $t_i \in V$.

With respect to this model, *instances* are resources that appear at the subject position of triples. An instance representation can be obtained from the data graph as follows:

Definition 2 (Instance Representation): The instance representation $IR : U \times \mathbb{G} \rightarrow \mathbb{G}$ is a function, which given an instance $s \in U$ and a graph $G \in \mathbb{G}$, maps s to a set of triples in which s appears as the subject, i.e. $IR(s) = \{(s, p, o) | (s, p, o) \in G, o \in L\}$.

Thus, an instance is basically represented through a set of *predicate-value* pairs, where values are bags of tokens. @TODO: add a graph to provide example, also, provide one example for instance representation We will use the terms instance and instance representation interchangeably from now on. Note that for the sake of presentation, only the outgoing edges (s, p, o) of an instance s are considered while incoming edges (triples where s appears as the object) can also be added to the representation of s.

B. Problem - Find Instance Matches and Match Candidates

Instance matching is about finding instances that refer to the same real-world object based on their representations extracted from the data. In this paper, we tackle the problem of *instance matching across multiple datasets*: given instances of a *source dataset* G_s, the problem is to find instances in others datasets, collectively referred to as the *target dataset* G_t, which represents the same real world object.

Challenges. Compared to the single dataset setting, this problem entails additional challenges especially when the data is heterogeneous not only at the data but also schema level. In this regard, *data-level heterogeneity* means that for the same property, instances referring to the same object may have different values, or different syntactical representations of the same value. @TODO: For instance, the values "Michael Jackson" or "Jackson, Michael" can be both used as the value of the property name of an instance representing Michael Jackson. (1) *Schema-level heterogeneity* arises when there is only little or no schema overlaps between the datasets. That is, instances referring to the same object may be represented by different predicates, or different representations of the same predicates. Finding different representations of the same predicate

is part of a problem also known as schema matching. Another challenge in this multiple dataset setting is (2) *efficiency*: for instance matching, a scheme is needed to determine how to compare two given instances; we will show that especially with schema heterogeneity, finding the best scheme for matching across heterogeneous datasets involves a greater search space and more feedback information (training data). Searching through all possible solutions and obtaining feedback information to evaluate them is expensive especially when we consider the online learning of instance matching schemes that requires access to data from remote endpoints. @TODO: discuss in intro that we need schemes for individual instances, and motivates and explain the concept of online learning of instance schemes: involves live access to remote endpoints that host fresh version of the datasets.

Computing Instance Matches. More precisely, an *instance matching scheme* is a (weighted) combination of similarity function predicates, $\sum_i w_i \sim (p_i) > \alpha$. Every $\sim (p_i)$ is a function, which given two instances s_i and s_j, returns the similarity between these instances based on their similarity on the values of the predicate p_i. The scheme computes the overall similarity between s_i and s_j by combining the similarities obtain for individual predicates and determines them as a *match* when its exceeds the threshold α. Clearly, such a scheme focus on data-level heterogeneity. Given s_i and s_j are in the source and target datasets, respectively, and these datasets vary in schema, an extended scheme of the form $\sum_i w_i \sim (\langle p^s_m, p^t_n \rangle_i) > \alpha$ is needed to capture that the values of the source predicate p^s_m shall be compared with values of the target predicate p^t_n. In other words, when the predicates in the source and target are not the same, comparable pairs of predicates have to be found and incorporated into the scheme. In fact, this problem entails the subproblems of (A) finding the pair of comparable predicates $\langle p^s_m, p^t_n \rangle$ (schema matching) as well as (B) choosing and (C) weighting them, and determining the (D) similarity functions $\sim$ (e.g. Jaccard distance) and (E) thresholds α.

Computing Instance Match Candidates. Instead of solving all these problems, some instance matching solutions focus on the blocking step, which aims to quickly select candidates matches (hence also referred to as the *candidate selection* step). Instead of using a combination of similarity function predicates, candidate selection simply employs a (conjunction of) blocking key(s), i.e. $\bigwedge \sim (\langle p^s_m, p^t_n \rangle_i)$ (called *candidate selection scheme*), where $\sim$ is a binary function that returns whether the two values of p^s_m and p^t_n match or not. Here, the predicates p^s_m and p^t_n constitute the pair of comparable blocking keys while their values are called *blocking key values* (BKV). Usually, the similarity function is based on exact value matching or value overlap. That is, two instances form a *candidate match* if their blocking key values are the same or overlap on some tokens. Mostly, $\sim$ is defined manually such that candidate selection amounts to the problem of (A) finding comparable predicate pairs and (B) choosing the most selective ones as blocking key pairs. Often, candidate selection is performed as a preprocessing step, producing results that are further refined by a more effective instance matcher that also tunes the weights and threshold to obtain better results.

C. Existing Solutions

State-of-the-art matching systems are based on supervised learning, leveraging training data as feedbacks to evaluate candidate schemes [18]. Basically, optimal schemes found are those which maximize the coverage of positive examples while avoiding negative examples. They are geared towards homogeneous datasets, focusing on the learning of schemes of the type $\sum_i w_i \sim (p_i) > \alpha$ as discussed above. It has been shown that in fact, the underlying learning strategies can also be used to obtain the extended schemes $\sum_i w_i \sim (\langle p^s_m, p^t_n \rangle_i) > \alpha$. Instead of all combinations of individual predicates, the search space would have to include all combinations of all possible pairs of predicates.

Since obtaining representative training data across datasets is difficult, recent approaches that specifically target heterogeneous data derive schemes directly from the data. However, unsupervised approaches of this kind focus on the more simpler problem, namely the learning of the candidate selection schemes $\bigwedge \sim (\langle p^s_m, p^t_n \rangle_i)$. For instance, Song and Heflin [9] assume precomputed schema mappings such that the comparable predicate pairs are known. They propose to choose them based on their coverage and discriminability; two metrics derived from the data basically reflecting the number of instances a given predicate can be applied to and how well it distinguishes them. Based on manually defined coverage and discriminability threshold, the best pairs of comparable blocking keys are selected.

As an alternative, a schema-agnostic approach [22] has been proposed for candidate selection. It does not use predicates for matching but treat instances simply as bags of value tokens. Instances form matches when they have some value tokens in common. Therefore, instances which share the same token (in any predicate) are placed in one candidate set. This approach does not require any effort for learning the scheme and is particularly suited when there is a lack of schema overlap such that only few or no comparable predicates exist. The problem with this is that the candidate sets produced are highly redundant because instances are often placed in multiple candidate sets. Consequently, this work employs much more additional processing to further refine these candidate sets.

D. Existing Solutions vs. Our Solution

In this work, we tackle the problem of instance matching. However, we focus on the problem of learning the candidate selection scheme and simply use the resulting scheme in combination with an existing matcher to refine candidate results.

It has been shown that the use of schema knowledge improves precision but considerably decreases the coverage of correct matches [22] (recall). In this work, we propose a supervised learning strategy to incorporate predicate information into the candidate selection scheme that considerably improves both measures compared to this previous work []. @TODO: what have to be cited here? what previous work do you mean?

The main difference between this and both the existing supervised and unsupervised learning strategies lies in the granularity of the learned scheme. Instead of using one scheme for all instances, our solution may yield different schemes for different source instances. This separate treatment of instances is introduced to specifically deal with the heterogeneity problem: for instance, two distinct drugs on Sider dataset, named Alpraxolan and Morphine, have two different schemes for mapping to the same drugs on Drugbank Dataset. Alpraxolan uses the scheme $\langle$`sider:label`, `drugbank:drugname`$\rangle$, while Morphine uses the scheme $\langle$`sider:name`, `drugbank:synomym`$\rangle$, Using these more fine-grained schemes, we show that the quality of results produced by our approach is superior than those produced by existing supervised [18] and unsupervised approaches [9].

Another aspect that has been neglected so far is time efficiency. More precisely, works on finding the scheme as discussed above focus on the quality of matches. On the other hand, there are works on executing similarity joins and building blocking indexes, focusing on how to process the schemes efficiently. In other words, more emphasis is put on the efficiency of execution and less on the efficiency of learning the scheme. The efficiency of learning became crucial when data is accessed over remote endpoints, where minimize the data access is necessary to be make any solution feasible.

Further, how well a scheme can be optimized for time efficiency depends on the nature of the scheme itself. Some schemes are inherently expensive, requiring a large amount of data to be loaded and to be joined. Further, schemes that produce the same result quality may vary in terms of runtime efficiency. Thus, optimized execution performance cannot be achieved independent of learning.@TODO: I did not understand this statement. learning of what? what is the relation with optimization?

In this work, we consider the entire process of learning and execution. We consider time as an additional optimization criteria such that optimal schemes are those, which (a) can be learned quickly, (b) can be executed efficiently and (c) yield high quality candidates. We show that this holistic optimization of time efficiency leads to faster execution. In fact, to achieve comparable quality results, the entire process of learning and execution is faster compared to the unsupervised approach, which requires almost no time in learning. We also compare the performance results for the entire process with the supervised approach, which requires training data to be locally available (i.e. offline learning instead of online learning over remote endpoints). Despite the overhead of retrieving data over endpoints, we show that our approach yields competitive performance.

E. Our Solution

To tackle this instance specific schema, we do not consider the solution for instance matching as weight of predicates, but rather, we consider it as a query problem for every instance. Therefore, a candidate set for an instance is the results of query over a target dataset.

We pose the problem of candidate selection as a optimization problem, where, for each source instance s, the goal is to find a template query that selects all positive matches for s, avoiding negative matches. Without lost of generality, we can assume that every source instance maps to a target instance (a 1-to-1 mapping), consequently, the optimal query for this problem would retrieve a candidate sets with one element. For now, we consider that an oracle can decide if the match is correct or not. Due the heterogeneity of the data, there is no unique template query that works for all instances. Therefore, to be effective, we need to find a template query for each specific source instance. As the number of those queries may be large, it is time prohibitive to evaluate all to find the optimal one; specially when we have to query a remote endpoint. Therefore, for every instance, we need to be time efficient by evaluating only the queries that has the highest chance to be optimal.

Basically, we tackle this problem in two stages. First, we build all possible effective template queries by sampling the source and target data. In this process we learn the most discriminative comparable predicates and each pair becomes one clause template query. Then, for each source instance, we use a branch-and-bound optimization algorithm that searches for those queries that have the highest chance to retrieve a optimal candidate set, through tree-structured space compose of all found template queries.

We start the process with a initial set of template queries. Aiming to approximate our solution to the optimal solution, we approach this problem in an iterative fashion, where at each iteration we make use of a set of policies to decide whether or not evaluate a query. Mainly, those policies are based on queries results obtained on previous iterations, and their aim

is to select the queries that have the highest chance to be optimal for next iterations, therefore this process minimizes the number of queries performed. Generally, the most selective queries are selected, which are more efficient and effective, because they select less elements (and it takes less time); and they select less incorrect matches. To be effective, at each iteration we refine the template queries, building highly selective template queries with respect to our optimal criteria (maximize positive and minimize negative matches).

Basically, this refining process adds another clause in the template queries, called class clauses. To build class clauses, we apply a matcher over the already generated candidate sets obtaining positive and negative matches that are input to an algorithm that output a set of class clauses, which are those predicate/value pairs that select only the positive matches. The matcher can be any approach that uses a more complex similarity measure to select the correct matches (positive examples) among the possible candidates.

This algorithm aims to find a best path of queries, representing a minimal set of time-efficient queries that produce high quality results.

IV. Learning Template Queries From Data

In this section we describe how we construct the template queries in our approach. Basically, we want to avoid queries composed by too many clauses, because they are too selective, and they end up missing some positive matches. This intuition was also identified by ObjectCoref[12]; and we empirically proved that in fact two clauses is enough to efficiently generate high recall, with acceptable precision. Therefore, in this work, we investigate template queries that are composed at most of two clauses; namely, an attribute clause and a class clause.

The attribute clause helps on find candidates based on the assumption that positive matches share a similar token on a pair of highly discriminative predicates. The class clause help on disambiguating candidates that share the same token but belong to different classes (class level ambiguity). We only build template queries that contains an attribute clause, or an attribute clause and an class clause.

Sampling is key point when learning from data, and almost impossible when the data is accessed remotely. We propose an adapted sampling method for dealing with the scalability issue on the heterogeneous setting. In this work, we assume that the source instances to be matched belong to the same class (e.g. country, people, drugs, etc.). Low cardinality classes are merged and a subset is chosen from high cardinality classes. This assumption also implies that the target matches for those source instances will also belong to a limited set of target classes. Therefore, to cover the necessary data to learn the predicates, much less data need to be mined, because both source and target instances are less diverse than to consider the whole dataset.

Given that the source instances S were given, we select 5% of S to determine the source discriminative predicates. We query the target endpoint using an exact boolean query over the tokenized value of those source predicates; obtaining the target data that were used to compute the target discriminative predicates. We empirically proved that this approach was efficiently and effective for our problem. However, note that the discussion on different sampling techniques is out of the scope of this paper.

A. Finding Attribute Clause

Let U_A be the set of all attributes of a dataset G, the candidate selection schema of G is a subset of attributes, i.e. $U_A^* \subseteq U_A$. A alignment between a source schema U_A^{*s} and target scheme U_A^{*t} is denoted by U_A^{*st}.

An attribute clause is defined as: $\langle p_t, o_s, \sim \rangle^A = \{\langle s_t, p_t, o_t \rangle | \langle s_t, p_t, o_t \rangle \in G_t \wedge o_s \sim o_t \wedge \langle p_s, p_t \rangle \in U_A^{*st}\}$, where $\sim$ is one of the four type of similarity function: EXACT, LIKE, AND and OR @TODO: explain this semantics. To build the attribute queries we first need to generate U_A^{*st}. To generate U_A^{*s} and U_A^{*t}, we use a similar approach propose by Song. et.al; based on the discriminability and coverage of the predicates. High discriminative predicates and with high coverage are selected. To produce the aligned schemas U_A^{*st}, known schema matching technique for heterogeneous data were applied.

B. Finding Class Clauses

An class clause is defined as: $\langle p_t, o_t \rangle^C = \{\langle s_t, p_t, o_t \rangle | \langle s_t, p_t, o_t \rangle \in G_t\}$. Assuming that a list of positive and negative matches are available, to build class clauses, we use a set-cover based algorithm [23] to quickly retrieve a list of target predicate/value pairs that occur in all positive matches but in none negative ones. The positive and negative matches are obtained during the candidate selection process that we will detail further. Then, each predicate/value pair found, become an individual class clause.

C. Composing Template Queries

We now defined a template query.

Definition 3 (Template Query): A template query is conjunction of clauses, defined as $(\bigwedge_i^n \langle p_i, o_i, \sim \rangle^A \bigwedge_t^m \langle p_t, o_t \rangle^C)$, where $n \geq 0$ and $m \geq 0$. In this work, we only consider template queries with at least an attribute clause, and a maximum of one class clause. The similarity $\sim$ defined the type of the query, refer to as *query type* from now on.

Definition 4 (Candidate Set): A candidate set of an instance s in G_s is a instantiation of a template query where we have at least an attribute clause $\langle p_t, k, \sim \rangle^A$ where $\langle s, p_s, k \rangle \in IR(s)$

For instance, for an template query formed by the attribute query $\langle$`rdf:label`, "`eosinophilic pneumonia`", $OR\rangle$, can be expressed in SPARQL as:

```
SELECT DISTINCT ?s
WHERE {?s rdf:label ‘‘eosinophilic pneumonia’’ }

SELECT DISTINCT ?s  WHERE {?s rdf:label ?o FILTER
regex(?0,‘‘eosinophilic pneumonia’’) }

SELECT DISTINCT ?s  WHERE {?s rdf:label ?o FILTER
regex(?o,‘‘eosinophilic’’)&&regex(?o,‘‘pneumonia’’)}

SELECT DISTINCT ?s  WHERE {?s rdf:label ?o FILTER
regex(?o,‘‘eosinophilic’’)||regex(?o,‘‘pneumonia’’)}
```

V. Branch-and-Bound Optimization

Suppose you have the following source data represented as triples:

```
<sider:12312> <label> "Morphine"
<sider:12312> <type> <sider:Drug>
<sider:43434> <title> "Eosinophilic Pneumonia"
<sider:43434> <type> <sider:Drug>
```

And target data:

```
<drugbank:DB00295> <drugname> "Morphine Sulphate"
<drugbank:DB00295> <synonym> "Morphine"
<drugbank:DB00295> <type> <drugbank:Drug>
<drugbank:DB00494> <drugname> "Eosinophilic Pneumonia"
<drugbank:DB00494> <type> <drugbank:Drug>
```

For this example, each source predicate in U_A^{*s} =`label`, `title` match all predicates in U_A^{*t} ={`name` and `synonym`}; therefore, there are four possible predicate alignment that could be exploited to find the match between the source and target instances, namely: U_A^{*st} ={$\langle label, drugname\rangle$, $\langle label, synonym\rangle$,$\langle title, drugname\rangle$, $\langle title, synonym\rangle$}. Those pairs in U_A^{*st} form four template queries as defined before. As any search optimization space, this set of possible choices (queries) forms a tree-shaped search space where each node represents a query and each level of the tree represents an instance. Then, the decision to be taken, it is to select a best query at each level of the tree. Fig.1 depicts this search space.

Given n source instances, k different query types and k comparable predicate pairs in U_A^{*st}, a naive approach would perform all $n \times k \times q$ queries to find the optimal candidate set for each instance. We show how this number of queries can be reduced through our branch-and-bound based optimization that aims to execute only a few effective queries for every source instance.

A. Search-based Optimization

The problem start by learning the schema alignments U_A^{*st} from the data. Then , from this alignment, a initial set of template queries containing only attribute queries is created. Our initial search space is composed of $n \times k \times q$ queries. We conceive an iterative process where source instances are processed one by one, resulting in a tree-shaped search space where the tree nodes correspond to queries and each level of the tree represents an instance A path on this tree from the root to a leave indicates the queries selected for their respective instances. We will use node and query as synonyms from now on.

The challenge is to select for each source instance the one fast query (few fast queries) needed to produce all and only the correct candidates. Thus, a query can be characterized by two dimensions, namely its *execution time* and the *optimality of its results*. As the optimality is not known, we propose a *cardinality-based heuristic*: a candidate selection query is more optimal when it yields less candidates (i.e. a result set with low cardinality). Thus, the most optimal query is the one that yields exactly one candidate while queries with no results or too many results are less optimal. While this is a rather aggressive heuristic that is exclusively focused on reducing the number of candidates (not taking into account whether they are correct or not), we show it performs well in the experiments.

The goal of the optimization is to avoid executing all queries, while selecting the optimal queries requires knowing the cardinality and execution time, which can only be obtained after executing the query. To avoid query execution, we use information acquired during the iterative process to estimate these. In particular, we note that for every instance, there are $k \times q$ query templates, which are initiated with the key value representation of that instance. For different instances, the queries instantiating a particular type vary only w.r.t. the key value component. Thus, for each query template, we recorded the execution time and cardinality observed for instances processed in previous iterations. Then, we compute an average for each of these dimensions and use it as the estimate for the instance in the current iteration.

B. Best-First Search With Branch-and-Bound Pruning

Based on these two dimensions, we use best-first search with branch-and-bound pruning [24] to execute only the best queries in the search space. The overall procedure is presented Alg. 3, which has three main components. It has a *bounding policy*, which decides when to stop the whole process. The *branching policy* determines the visit order of nodes within every level and when to move to a next level, based on cardinality and time estimates. Further, a classifier helps the bounding policy to decide whether to skip certain queries or not.

Branching. It is a breadth-first search procedure that processes queries associated with the source instance of the current level before moving to the next instance in the next level. In every level, this search is guided by the branching policy, which always selects the node that is best w.r.t. both dimensions. More precisely, it chooses the one with lowest cardinality and among

those not distinguishable in terms of that, it chooses the one that requires less time (based on the estimated values discussed before) . According to the cardinality-based heuristic, this policy indicates to stop and to move to the next level when a node with cardinality 1 is visited (because there are no better queries than this). However, this cardinality-based heuristic can only be applied to query nodes constructed for the same key pair. Queries constructed for different key pairs may be useful to retrieve candidates of different types (e.g. the query constructed from `rdf:label` retrieves candidates of type `Ingredient` while the query constructed from `drugs:drugname` retrieve `Drugs`). In other words, we need to find optimal queries specific to a key pair. Accordingly, when a query with cardinality 1 has been found, the search moves to the queries constructed for a different key pair. Once the optimal queries have been found for all key pairs, or when all queries have been processed, the search moves on to the next level. We use only the instances retrieved for the one with lowest cardinality, among all queries processed for a given key pair. Over all key pairs, we aggregate the results retrieved for their optimal query to obtain all candidates for one source instance.

Bounding. While the branching policy determines the next node to evaluate (and to stop processing one level when the optimal query is found) the *bounding policy* decides when to stop the whole process. The search should terminate when there exist no or too few matches for the source instances. This can be decided when through many levels, empty candidates sets are obtained as results. In fact, this kind of early termination could also be applied to the search within every level: when too many nodes within the same levels yield empty result, we can move on to the next level. As shown in Alg. 3, the bounding strategy is controlled by the parameter γ. In the experiments in this paper, early termination through bounding is not used because we know in advance that matches exist between the datasets.

Classifier Prediction. While the branching policy can help to reduce the number of nodes per level, it requires processing all nodes until a query with cardinality 1 is found. This strategy can be further optimized as we observe that given a particular preceding node, the current node to be chosen by the branching strategy mostly has higher cardinality, hence does not need to be executed. To exploit this, we train a classifier, which given a node, predicts if the current node has lower cardinality or not. The process actually include three phases, as illustrated in Fig. **??**. The branch-and-bound strategy is used only in the second and third phase; the predictor only in the third phase. Hence, as evident in Fig. **??**, lesser nodes are executed only in this *Predicting* phase, while there is no or only little reduction of nodes in the first two phases.

Algorithm 1 CandidateSelection(G, G'). Find candidates for instances in G.

```
sourcekeys ← FindCandidateSchema(G)
targetkey ← FindCandidateSchema(G')
keypairs ← AligningSchemas(sourcekeys, G, targekeys, G')
nodes ← BuildNodes(keypairs)
learning ← true
for all b in targetkeys do
  predictor[b] ← NayveBayesClassifier.new()
end for
for all i in G do
  if  i ≥ γ and candidates = ∅  then
    return null //Satisfied bounding policy
  end if
  for all node in nodes do
    b ← node.targetkey
    if cost[b] = 1 then
      next //Satisfied branching policy
    end if
    if learning or predictor[b].predict(node) then
      cost[b] = node.evaluate(i)
      processed[b] ← processsed[b] + node
      if learning then
        predictor[b].AddExample(node) //Learning Phase
      end if
    end if
  end for
  if i ≤ β then
    SortNodesByElapseTime(nodes) //Sort nodes
  end if
  if test error converges then
    learning ← false
  end if
  if i = α then
    examples ← matcher(candidates)
    nodes ← updateNodes(examples)
    learning ← true
  end if
  candidates[i] ← AggregateMinimalCandidatesSet(processed)
end for
return  candidates[i]
```

In the *Sorting* phase, nodes are sorted by their average evaluation time. Average time and this order are computed after $\beta\%$ of the instances have been processed. This order is kept to train the classifier and also used during the Prediction phase (so that branching continues with the query that requires lesser time, given they all equal in terms of cardinality). The β parameter can be set to be small to obtain average time simply after a few instances (we set it to 1% in the experiment).

In the *Learning* phase, the predictor is learned as a Naive Bayes classifier [25]. It predicts whether the current node to be chosen by the branching strategy indeed has lower cardinality or not. As features, we use the type of the current query, the type of the previous query and a boolean value indicating if the previous query has cardinality greater than zero (i.e. whether it was executed before). Recall that the branching policy applies to queries constructed for every key pair and it moves on to next queries when it found an optimal one. Similar to that, a classifier is learned for every set of nodes that share the same target key and accordingly, is only used for queries that have been constructed from this target key. The Learning phase stops soon after three iterations have the same test error. When the Learning phase terminates, both the branching policy and the predictor are applied to choose and to skip

query nodes, respectively. The effect of branching with and without the predictor is shown in Fig. **??**.

In the *Updating* phase, new template queries are generated with a class clause. The class queries are computed after $\alpha\%$ of the instances have been processed. In this work, we fix α to 5% of the instances. Then, the candidates obtained so far are input to a matcher that outputs positive and negative examples. As a matcher, the class-based disambiguator [**?**] is used in this work, because we assumed the source instances belong to a same class. Those generated examples are input to the algorithm that computes the class clauses. For each class clause found a new query is created by adding it to the original queries. For q queries and n class clauses, then $n \times q$ new queries are generated in the end. After this process, the classifier is retrained using the Leaning procedure described before.

VI. Evaluation On Candidate Selection

In this section, we discuss how we evaluate Sonda on the context of candidate selection and discuss about the results. Our system Sonda was implemented in Ruby and the queries were implemented as SPARQL queries issues over alive SPARQL endpoints. Sonda is available for download at GitHub as a command line tool [2], as well as all the results that we obtained.

A. Datasets

We evaluated our framework using the datasets and ground truth published by the instance-matching benchmark of the Ontology Alignment Evaluation Initiative (OAEI) [26]. We used the datasets provided in 2010 and 2011. We used the life science (LS) collection (which includes Sider, Drugbank, Dailymed TCM, and Diseasome) and the Person-Restaurant (PR) from the 2010 collection. We excluded LinkedCT from our experiments due to known quality problem in the ground truth. We used all datasets from the 2011 collection.

B. Evaluation metrics and alternative approaches

We used standard metrics, namely Reduction Ratio (RR), Pair-wise Completeness (PP) and F1. Basically, high RR means that the candidate selection algorithm helps to focus on a smaller number of candidates, while high PP means that it could preserve more of the correct candidates. Because the number of all possible candidates is large in this scenario, we use a normalized version of RR. In particular, these metrics are computed as follows:

$$\mathrm{PC} = \frac{\#\text{Correctly Computed Candidates}}{\#\text{Ground Truth Candidates}}$$

$$\mathrm{RR} = \frac{\#\text{Instances with Non-Empty Candidate Sets}}{\#\text{All Computed Candidates}}$$

[2]https://github.com/samuraraujo/ICDE2013

$$\mathrm{F1} = \frac{2 * \mathrm{RR} * \mathrm{PC}}{\mathrm{RR} + \mathrm{PC}}$$

Beside these metrics, we also count the average number of queries evaluated per instance as well as overall time for accomplishing the task of leaning the candidate selection scheme and finding(searching) the candidate sets.

For comparison, we implemented the *S-agnostic* [8] and *S-based* [9] approaches as discussed in Sec. 2. S-based uses only an OR query and it does not feature the branch-and-bound optimization. It requires key pairs, which are generated as in Sec. 3.1. Further, S-based applies a similarity function on the keys to further prune incorrect candidates after that have been retrieved using the OR queries. For comparison purposes, we apply this strategy to all approaches, using the same similarity function. Sonda uses four types of queries for each key pair, and employs the proposed branch-and-bound optimization to select best queries.

C. Querying Candidates

We implemented the queries in our algorithm as SPARQL queries (as discussed before) and directly query a SPARQL endpoint to obtain results (limit to 30 instances per query). For that, we loaded all datasets into the OpenLink Virtuoso Universal Server (Version 6.1.5.3127), except for DBPedia, which we queried its on-line sparql endpoint. We use the default S-P-O index created by this server, and created an inverted index for literal values using the following commands:

```
DB.DBA.RDF_OBJ_FT_RULE_ADD
(null, null, 'index_local');
DB.DBA.VT_INC_INDEX_DB_DBA_RDF_OBJ ();
```

We use the specific Virtuoso SPARQL implementation to have access to the index, and we limited all query results to 30 instances. This avoids the queries to retrieve too many data for non-discriminative queries.

D. Candidate Selection Results

Table 1 shows the results. Comparing all approaches over all the 16 datasets pairs that we evaluated, Sonda achieves the best F1 score in all cases (in 96% of the cases, to be precise), except for the NYT-Geonames pair, where S-based has best F1 result (due to high RR).

Overall Sonda ($Sonda_A$ and $Sonda_C$) achieves a considerable improvement in both RR and PC than the other approaches, in the Freebase, DBPedia and Geonames cases. Mainly, due to the high level of ambiguity on those target datasets, those approaches performed worse because they considered only OR queries (that are too inclusive and penalize RR), or do not use comparable keys as in the cases of S-agnostic (penalizing PC).

Class Queries. The RR in $Sonda_C$ was considerably higher than in $Sonda_A$ for a few cases (DBPedia, Freebase and Geonames). This occurred due to class level

ambiguity (many instances with the same name but from different classes) in those datasets. It indicates that the class clauses indeed make the queries more precise because it select exactly the correct class of target instances. However, a little increase on searching time can be observed in $Sonda_C$, mainly due to a higher number of queries that were considered in this version.

Time Performance and F1-Measure. Regarding the learning time performance, in average it takes less than 10% of the overall searching time. Although, a more complex learning process (that produces better comparable keys) takes more time, it leads to a better F1.

In general, the best F1 can only be achieved by using an complex learning process and an efficient execution process. For instance, S-based used the same set of comparable keys than Sonda, but it achieved the worse searching time performance in all cases. Even though Sonda has more queries to evaluate (because it uses different types of queries with different time performances), it is faster than S-based, because its branch-and-bound algorithm helps to select the ones that require less evaluation time (those that are more efficient than the ones used by the other approaches). In average, S-agnostic has the best learning and searching time performance, but the worse F1.

Concluding, the best F1 is associated to learn effective comparable keys, which take more time to compute, and to learn efficient queries execution plan, which can efficiently execute the most effective queries.

Query Type Effectiveness. As S-based and S-agnostic considered more inclusive queries (OR queries) than Sonda, we can conclude that, although necessary, more inclusive queries leads to worse F1, mainly because they decrease the RR, and as it was used a limit on queries, it also decrease the PC.

Predictor and Branching Policy Efficiency. We can see that in all cases, Sonda achieves a considerable reduction in the number of queries evaluated per instance. In some cases (e.g., Dailymed-Sider, Diseasome-Sider), it performed nearly one query per instance. Notice that we can not compare the predictor on S-based and S-agnostic, because in both cases, the number of queries executed per instance is fixed. The purpose here is to show that Sonda's predictor and branching policy was very efficient, selecting only a few queries per instance, as well as, very effective, selecting queries that produce near optimal PC, in many cases.

VII. Evaluation On Instance Matching

In this section, we discuss how we evaluate Sonda on the context of instances matching. Mainly, we apply two matcher overs the candidates sets produced by Sonda. Then, we compared the results of the match with state-of-the-art instance matching approaches that participate on the OAEI 2010 and 2011 challenge. Also, the ExampleDriven approach was evaluated over the same datasets. As evaluation metrics, we used standard precision, recall and F1 metrics.

A. Instance Matching Results

In this section, we compared the Sonda F1-measure (between precision and recall) with the other alternatives approaches that were evaluated over the same datasets.

VIII. Conclusions

References

[1] A. Ferrara, A. Nikolov, and F. Scharffe, "Data linking for the semantic web," *Int. J. Semantic Web Inf. Syst.*, vol. 7, no. 3, pp. 46–76, 2011.

[2] D. Song and J. Heflin, "Domain-independent entity coreference in rdf graphs," in *CIKM*, 2010, pp. 1821–1824.

[3] P. Cudré-Mauroux, P. Haghani, M. Jost, K. Aberer, and H. de Meer, "idmesh: graph-based disambiguation of linked data," in *WWW*, 2009, pp. 591–600.

[4] A. Nikolov, V. S. Uren, E. Motta, and A. N. D. Roeck, "Refining instance coreferencing results using belief propagation," in *ASWC*, 2008, pp. 405–419.

[5] M. A. Hernández and S. J. Stolfo, "The merge/purge problem for large databases," pp. 127–138, 1995.

[6] M. Michelson and C. A. Knoblock, "Learning blocking schemes for record linkage," in *AAAI*, 2006, pp. 440–445.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 1–16, 2007.

[8] G. Papadakis and W. Nejdl, "Efficient entity resolution methods for heterogeneous information spaces," in *ICDE Workshops*, 2011, pp. 304–307.

[9] D. Song and J. Heflin, "Automatically generating data linkages using a domain-independent candidate selection approach," in *International Semantic Web Conference (1)*, 2011, pp. 649–664.

[10] F. Scharffe and J. Euzenat, "Linked data meets ontology matching - enhancing data linking through ontology alignments," in *KEOD*, 2011, pp. 279–284.

[11] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *KDD*, 2000, pp. 169–178.

[12] W. Hu, J. Chen, and Y. Qu, "A self-training approach for resolving object coreference on the semantic web," in *WWW*, 2011, pp. 87–96.

[13] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov, "Discovering and maintaining links on the web of data," in *International Semantic Web Conference*, 2009, pp. 650–665.

[14] D. Song and J. Heflin, "Automatically generating data linkages using a domain-independent candidate selection approach," in *International Semantic Web Conference (1)*, 2011, pp. 649–664.

[15] X. Niu, X. Sun, H. Wang, S. Rong, G. Qi, and Y. Yu, "Zhishi.me: weaving chinese linking open data," in *Proceedings of the 10th international conference on The semantic web - Volume Part II*, ser. ISWC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 205–220. [Online]. Available: http://dl.acm.org/citation.cfm?id=2063076.2063091

[16] W. Hu, Y. Qu, and X. Sun, "Bootstrapping object coreferencing on the semantic web," *J. Comput. Sci. Technol.*, vol. 26, no. 4, pp. 663–675, 2011.

[17] J. Li, J. Tang, Y. Li, and Q. Luo, "Rimom: A dynamic multistrategy ontology alignment framework," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 8, pp. 1218–1232, 2009.

[18] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik, "Example-driven design of efficient record matching queries," in *VLDB*, 2007, pp. 327–338.

[19] A. Nikolov, M. d'Aquin, and E. Motta, "Unsupervised learning of link discovery configuration," in *ESWC*, 2012, pp. 119–133.

[20] X. Niu, S. Rong, Y. Zhang, and H. Wang, "Zhishi.links results for oaei 2011," in *OM*, 2011.

[21] S. Araujo, T. Tran, A. P. de Vries, J. Hidders, and D. Schwabe, “Serimi: Class-based disambiguation for effective instance matching over heterogeneous web data,” 2012.
[22] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser, “Efficient entity resolution for large heterogeneous information spaces,” in *WSDM*, 2011, pp. 535–544.
[23] R. D. Carr, S. Doddi, G. Konjevod, and M. V. Marathe, “On the red-blue set cover problem,” in *SODA*, 2000, pp. 345–353.
[24] R. Dechter and J. Pearl, “Generalized best-first search strategies and the optimality of a*,” *J. ACM*, vol. 32, no. 3, pp. 505–536, 1985.
[25] D. J. Hand and K. Yu, “Idiot’s Bayes—Not So Stupid After All?” *International Statistical Review*, vol. 69, no. 3, pp. 385–398, 2001. [Online]. Available: http://dx.doi.org/10.1111/j.1751-5823.2001.tb00465.x
[26] J. Euzenat, C. Meilicke, H. Stuckenschmidt, P. Shvaiko, and C. T. dos Santos, “Ontology alignment evaluation initiative: Six years of experience,” *J. Data Semantics*, vol. 15, pp. 158–192, 2011.

TABLE I

Results of the three systems over all pairs of datasets. Queries denote the total number of queries given to the system. Queries/Instance(Q/I) denotes the amount of queries evaluated per instance.

Dataset Pairs	Systems	Queries	Q/I	Learning(s)	Search(s)	RR(%)	PC(%)	F1(%)
Restaurant1-Restaurant2	SondaA	10	1.76	6.78	9.33	99.12	98.23	98.67
	SondaC	10	1.77	6.75	10.18	99.12	98.23	98.67
	S-based	2	2.0	3.67	21.85	89.52	97.35	93.27
	S-agnostic	3	3.0	0.74	18.3	90.98	97.35	94.06
Person11-Person12	SondaA	15	1.38	8.9	38.53	99.57	93.54	96.46
	SondaC	15	1.42	8.7	47.62	99.57	93.54	96.46
	S-based	3	3.0	5.84	93.15	31.87	100.0	48.34
	S-agnostic	4	4.0	4.45	69.63	30.25	100.0	46.45
Person21-Person22	SondaA	25	13.51	2.9	43.77	22.88	96.61	36.99
	SondaC	25	13.33	2.75	41.51	22.88	96.61	36.99
	S-based	5	5.0	1.42	18.31	22.01	100.0	36.08
	S-agnostic	4	4.0	0.57	11.06	21.67	100.0	35.62
Sider-Tcm	SondaA	10	1.75	7.96	14.44	86.22	98.83	92.1
	SondaC	10	1.6	7.24	13.14	87.11	98.83	92.6
	S-based	2	2.0	4.1	22.88	77.93	96.49	86.23
	S-agnostic	3	3.0	1.29	24.07	77.93	96.49	86.23
Sider-Dailymed	SondaA	25	2.47	81.06	175.73	30.18	67.61	41.73
	SondaC	75	5.43	83.09	268.66	33.94	68.89	45.48
	S-based	5	5.0	75.28	530.33	23.25	84.94	36.5
	S-agnostic	3	3.0	31.73	7909.14	31.32	81.75	45.29
Sider-Drugbank	SondaA	25	3.26	51.19	284.51	92.68	98.49	95.49
	SondaC	25	3.31	53.62	289.05	92.68	98.49	95.49
	S-based	5	5.0	51.9	500.29	90.12	99.3	94.49
	S-agnostic	3	3.0	41.91	358.93	93.74	97.79	95.72
Sider-Diseasome	SondaA	20	2.02	10.75	24.19	52.44	95.35	67.67
	SondaC	20	2.03	10.2	23.11	52.44	95.35	67.67
	S-based	4	4.0	7.91	99.04	59.25	90.7	71.67
	S-agnostic	3	3.0	1.35	49.44	61.33	88.95	72.6
Dailymed-Sider	SondaA	40	1.42	101.7	263.1	98.39	99.37	98.88
	SondaC	40	1.34	34.35	210.91	99.87	99.87	99.87
	S-based	8	8.0	28.17	1385.41	96.85	97.99	97.42
	S-agnostic	4	4.0	16.19	759.43	96.85	97.99	97.42
Diseasome-Sider	SondaA	20	1.85	12.63	17.78	97.62	95.35	96.47
	SondaC	20	1.85	9.1	13.66	97.62	95.35	96.47
	S-based	4	4.0	6.37	51.43	85.11	93.02	88.89
	S-agnostic	2	2.0	2.06	27.34	85.11	93.02	88.89
Drugbank-Sider	SondaA	40	5.88	81.49	208.78	98.61	99.29	98.95
	SondaC	80	9.92	70.43	375.57	97.92	99.29	98.6
	S-based	8	8.0	53.9	273.07	92.76	99.65	96.08
	S-agnostic	26	26.0	24.56	281.62	92.46	99.65	95.92
NYT-Geonames	SondaA	10	2.4	44.54	1840.09	17.51	74.23	28.34
	SondaC	10	2.35	25.14	1621.12	18.33	74.51	29.42
	S-based	2	2.0	22.39	709.52	17.55	23.53	20.11
	S-agnostic	5	5.0	11.57	1002.44	26.07	50.2	34.32
NYT-DBPedia(Geo)	SondaA	5	1.02	43.03	998.8	65.23	75.21	69.86
	SondaC							
	S-based							
	S-agnostic							
NYT-DBPedia(Per)	SondaA							
	SondaC	10	1.2	203.13	5209.35	81.94	97.11	88.88
	S-based	1	1.0	65.93	3516.51	56.89	14.27	22.81
	S-agnostic	4	4.0	25.85	5019.58	30.55	12.02	17.25
NYT-DBPedia(Corp.)	SondaA	5	1.55	25.85	1516.55	52.58	90.28	66.46
	SondaC	5	1.55	28.34	1707.08	71.02	82.09	76.15
	S-based							
	S-agnostic							
NYT-Freebase(Geo)	SondaA	20	3.59	39.89	819.35	82.55	94.43	88.09
	SondaC	40	4.04	31.57	978.81	86.39	95.0	90.49
	S-based	4	4.0	24.23	1308.69	58.07	78.7	66.83
	S-agnostic	5	5.0	12.84	749.24	60.26	64.38	62.25
NYT-Freebase(Corp.)	SondaA	15	3.02	30.51	911.26	78.06	88.27	82.85
	SondaC	15	3.09	22.09	986.65	73.21	88.17	80.0
	S-based	3	3.0	16.73	1468.65	62.47	69.05	65.6
	S-agnostic	2	2.0	13.23	622.9	63.05	43.04	51.15
NYT-Freebase(Person)	SondaA	15	2.12	50.75	1117.39	88.67	96.2	92.29
	SondaC	45	3.16	45.22	1396.57	94.62	95.62	95.12
	S-based	3	3.0	46.14	3536.16	71.6	73.57	72.57
	S-agnostic	4	4.0	25.87	1321.02	86.21	28.68	43.04

TABLE II

Sonda F1-measure (between precision and recall) compared to ExampleDriven and other tools that participate on the OAEI 2011 benchmark.

Dataset	SondaA	SondaC	KnoFuss+GA	AggreementMaker	SERIMI	Zhishi.links	ExampleDriven
DBPedia - Geo.	0.63	0.63	0.89	0.69	0.68	0.92	0
DBPedia - Corp.	0.91	0.91	0.92	0.74	0.88	0.91	0
DBPedia - People	0.96	0.96	0.97	0.88	0.94	0.97	0
Freebase - Geo.	0.90	0.90	0.93	0.85	0.91	0.88	0
Freebase - Corp.	0.87	0.87	0.92	0.80	0.91	0.87	0
Freebase - People	0.96	0.96	0.95	0.96	0.92	0.93	0
Geonames	0.63	0.63	0.90	0.85	0.80	0.91	0
Average	0	0	0.93	0.85	0.89	0.92	0

TABLE III

Sonda F1-measure (between precision and recall) compared ExampleDriven and other tools that participate on the OAEI 2010 benchmark.

Dataset	$Sonda_A$	$Sonda_C$	SERIMI	ObjectCoref	Rimon	ExampleDriven
Sider-Dailymed	0.63	0.61	**0.66**	-	0.62	0
Sider-Diseasome	**0.90**	0.90	0.87	-	0.45	0
Sider-Drugbank	0.93	0.93	**0.97**	-	0.50	0
Sider-TCM	0.92	0.92	**0.97**	-	0.79	0
Dailymed-Sider	0.93	**0.94**	0.67	0.70	0.62	0
Drugbank-Sider	**0.80**	**0.80**	0.48	0.46	-	0
Diseasome-Sider	**0.95**	**0.95**	0.87	0.74	-	0
Person11-Person12	0.95	0.95	**1.00**	0.99	**1.00**	0
Person21-Person22	0.45	0.45	0.46	0.95	**0.97**	0
Restaurant1-Restaurant2	**0.98**	**0.98**	0.77	0.81	0.88	0
Average[3]	**0.84**	**0.84**	0.77	-	-	0
Average[4]	0.84	**0.85**	0.71	0.78	-	0
Average[5]	**0.84**	**0.84**	0.80	-	0.73	0