2018 IEEE 34th International Conference on Data Engineering

Seeping Semantics: Linking Datasets using Word Embeddings for Data Discovery

Raul Castro Fernandez* Essam Mansour◇ Abdulhakim A. Qahtan◇ Ahmed Elmagarmid◇
Ihab F. Ilyas‡ Samuel Madden* Mourad Ouzzani◇ Michael Stonebraker* Nan Tang◇
*MIT CSAIL ◇Qatar Computing Research Institute, HBKU ‡University of Waterloo
{raulcf, madden, stonebraker}@csail.mit.edu
{emansour, aqahtan, aelmagarmid, mouzzani, ntang}@hbku.edu.qa ilyas@waterloo.ca

***Abstract*—Employees that spend more time finding relevant data than analyzing it suffer from a data discovery problem. The large volume of data in enterprises, and sometimes the lack of knowledge of the schemas aggravates this problem. Similar to how we navigate the Web, we propose to identify semantic links that assist analysts in their discovery tasks. These links relate tables to each other, to facilitate navigating the schemas. They also relate data to external data sources, such as ontologies and dictionaries, to help explain the schema meaning. We materialize the links in an enterprise knowledge graph, where they become available to analysts. The main challenge is how to find pairs of objects that are semantically related. We propose SEMPROP, a DAG of different components that find links based on syntactic and semantic similarities. SEMPROP is commanded by a semantic matcher which leverages word embeddings to find objects that are semantically related. We introduce coherent group, a technique to combine word embeddings that works better than other state of the art combination alternatives. We implement SEMPROP as part of Aurum, a data discovery system we are building, and conduct user studies, real deployments and a quantitative evaluation to understand the benefits of links for data discovery tasks, as well as the benefits of SEMPROP and coherent groups to find those links.**

I. INTRODUCTION

Many large organizations have a critical need for *data discovery* systems; their employees spend more time finding relevant data than analyzing it. For example, data scientists at a major pharmaceutical company report that they spend the majority of their time searching for relevant data over more than $4,000$ relational databases, a large data lake, and a myriad of files. There is no fully integrated schema or global schema relating these data sets, and building such an integrated database would take many years and millions of dollars.

As a result, navigating this collection of data is a huge challenge, especially since many analysts are not used to working with databases; some of them are not even computer scientists. Hence, if an analyst wants to identify, for example, which assays are involved in a drug study, she must either have a good understanding of the structure and schema of the different data sources or rely on other employees who do. Moreover, this time-consuming exploration often misses relevant data that remains unknown to the analysts.

Similar to how we find relevant content on the Web today, our key insight is that we can help analysts understand and navigate these datasets by linking related datasets. Finding links between datasets is a well-understood task in many settings. For example, schema matching and data integration tools find *same-as* relationships: fields that are the same in a source and a target schema [1]. On the other hand, knowledge construction tools [2] establish knowledge bases that link concepts and individuals via various types of semantic relationships that can be very expressive (such as *born-in*, *married-to*, and *subclass-of*) and allow reasoning on these relationships.

These existing techniques for mining and identifying relationships between datasets are not appropriate for data discovery. On the one hand, finding same-as relationships (e.g., in schema matching) fails to link datasets that, while not the same, are semantically related. For example, a dataset on drug standards enriches one about clinical trials because it offers more detail on different drugs involved in these trials. So although we are not interested in mapping them directly, having them side by side proved to be useful in practice. On the other hand, creating an expressive and well-crafted knowledge base with a rich ontology is an overkill for data discovery, and is a hugely expensive and labor-intensive manual task when data is highly heterogeneous, even with the benefit of semi-automated tools.

Instead, we propose a fully automatic system that is able to *surface* related datasets to the analyst, who can then easily identify their semantic relationships and employ them in their analytics tasks. We draw on the Web as an analogy: surfacing related documents through a search engine is highly valuable even though Web search engines cannot reason about exact semantic relationships, e.g., that page A *explains* page B, or the content of page C *contradicts* the content of page D.

We show in this paper that a general Web-style knowledge graph is not only useful for data discovery, but is the right compromise between coverage and expressiveness of the discovered links among various datasets and efficiency of its creation. However, as we will show, simple web-style search for data items that have high syntactic similarity is insufficient, as it generates many items that are not truly related. Thus, a more sophisticated approach to discover deeper relationships between data items is needed.

A. Approach Overview

We illustrate our approach with Fig. 1, which shows two tables (*Target_Dictionary* and *Drug_Indication*) of the Drug-

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00093

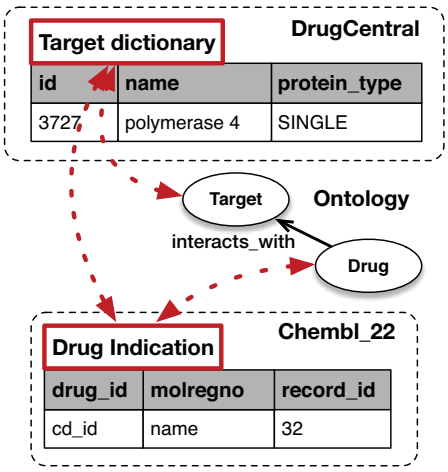

Fig. 1: Example of discovered links in databases

Central [3] and ChEMBL [4] databases, respectively. In this example, our approach first identifies links between the table names and classes of an existing ontology (shown in the center of the figure). By transitively following these links, we establish another link between the tables. These links can then be materialized for consumption and used in data discovery. We materialize the links in an *enterprise knowledge graph (EKG)*; a graph structure whose nodes are data elements such as tables, attributes and reference data such as ontologies and mapping tables and whose edges represent different relationships between node elements. The EKG is part of a larger data discovery project and helps analysts understand how different tables are connected (even from different databases).

B. Technical Challenges and Contributions

When building the links, we need to effectively assess if two objects are related. The criteria to identify related objects needs to go beyond simple syntactic similarity methods (edit or Jaccard similarity of strings), or otherwise many links would correspond to false positives, such as "beautiful bride" and "bountiful bribe", which although are similar according to edit distance, are not semantically related.

Our key idea is to rely on a new *semantic* similarity metric based on *word embeddings* [5]. Word embeddings capture semantic similarity and dissimilarity of words based on the distributional hypothesis [6], which states that words that appear together in some corpus of documents usually share similar meaning or are closely related. For example, "Fahrenheit" and "Celsius" are semantically similar, although not syntactically similar. This new semantic similarity metric helps us identify that "beautiful bride" and "bountiful bribe" are not semantically similar, and therefore should not be used to relate datasets.

As a result of the new metric, the links are of higher quality than those generated using only traditional syntactic similarity measures that introduce many false positives. In addition, the new semantic similarity measure finds new links that would be missed otherwise, such as a link between attributes *isoform - protein* of the chemical databases of the example. Word embeddings address our challenge of finding semantically similar items in a fully automated fashion without needing to determine the exact nature of the semantic relationships.

Despite the benefits of word embeddings, their direct use to solve our task is hindered by two well-known problems: i) multi-word phrases, e.g., "drug interaction"; and ii) out-of-vocabulary terms for which there is not a word embedding. We introduce the concept of *coherent groups* to tackle both problems by combining word embeddings from multi-word phrases; the key idea is that a group of words is similar to another group of words if the average similarity (in the embedding) between all pairs of words is high. This is a novel approach that performs better than existing techniques for multi-word phrases and out-of-vocabulary terms in the embedding literature.

Semantic Propagation System. Our approach to address the above challenges is in the form of a semantic propagation mechanism, called SEMPROP. Semantic propagation consists of identifying high quality links between datasets that help analysts understand the data, i.e., which datasets are connected, ultimately facilitating data discovery. Using a DAG, SEMPROP combines semantically-filtered syntactic links, new semantic links identified by our semantic matcher based on word embeddings, and the output of a structural summarizer—which exploits the structure of reference data such as ontologies to further curate links. In summary, our contributions are:

- We define the semantic propagation problem, and show how to generate links to help users with data discovery tasks (Section II-B).
- We introduce a semantic matcher based on what we call *coherent groups*, which allow us to use word embeddings to find links. (Section III).
- We introduce SEMPROP, an end-to-end system that orchestrates a series of traditional syntactic matchers along with our new semantic matcher and structural summarizer. SEMPROP achieves high quality links without requiring human intervention (section IV).

Evaluation. We evaluate SEMPROP using datasets from three different domains (Section V). We conduct a user study to demonstrate the value of finding links between semantically related objects, report on our experience on a real deployment, and quantitatively evaluate the benefits of SEMPROP, including coherent groups for generating links with ground truth retrieved from real datasets. As an example, we demonstrate how our mechanism finds high quality links that help discover data across three drug databases, a private one from our collaborators, and two public databases.

II. OVERVIEW

A. Notation

Source Elements. The *source elements* of a database are the union of the names of all relations and attributes, denoted by $\mathbf{SE} = \mathbf{N} \cup \mathbf{A}$, where $\mathbf{N}$ refers to all relations and $\mathbf{A}$ all attributes. Take Fig. 1 for example, the source elements of the database *DrugCentral* are {Target dictionary, id, name, protein_type}.

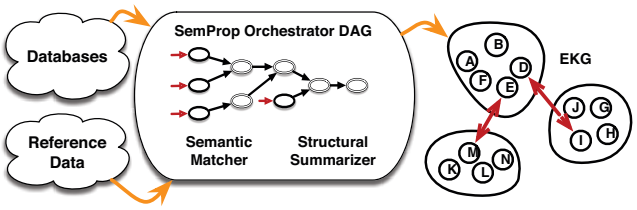

Fig. 2: SemProp overview

Relations may come from different data repositories. They may have been created by different people under different assumptions and using different naming conventions. When finding links for discovery, we will consider both the relation names as well as the attribute names, i.e., the source elements. This is necessary to cover databases that represent their entities at different granularity levels.

Ontologies. An *ontology* contains a set of *classes* and relationships between *classes*. A *class* represents the concept related to a set of entities, e.g., *Protein*. A *relationship* is a binary predicate that represents a connection between two classes in an ontology, e.g., *Drug interacts_with Target*. Many ontologies also contain a hierarchical organization of their classes, e.g., *Protein* is a subclass of *Chemical Compound* in the Experimental Factor Ontology (EFO) [7]. Each class typically has a name, and possibly other associated metadata, but ontologies do not generally contain actual data instances.

Reference Data. This is data that is considered gold standard in a domain. We consider two types: 1) ontologies and 2) mapping tables that indicate that two attributes are equivalent, and how their values map onto each other.

Semantic Links. We start by creating links between reference data and source elements; specifically, we create a link between a source element e in $\mathbf{SE}$ and a class c in $\mathbf{C}$ if their relevance, measured by an evidence function $f(e, c)$, is greater than a predefined threshold θ. For example, we may determine that a link exists if the similarity between a class name and an attribute name is greater than a threshold. We will introduce different *matchers* whose job is to identify links.

Transitive Link Propagation. Once we obtain the links, we check which links can transitively complete the graph, and create additional links between two source elements e_i, e_j. For example, if (i) both source elements have links to the same ontology class, and (ii) they have links to different classes in the ontology, and these are related, e.g., they are parent-child, or one class has a relationship with respect to the other that is specified in the ontology. We discuss in detail in section IV-C.

B. Problem and Solution Overview

Problem. Given a set of source elements, and a set of reference data, the problem of *semantic propagation* is to identify links between source elements and reference data and use these to transitively find additional links between source elements, drawing relationships from reference data.

Solution Overview. An overview of our approach, SEMPROP, is shown in Fig. 2. SEMPROP uses a set of matchers, including our new semantic matcher based on word embeddings, as well as several other syntactic matchers, augmented with negative signals from our semantic matcher (Section III). These matchers are orchestrated by a DAG that produces a set of links after using a structural summarizer (section IV). In this paper, we are concerned with generating the links. Later, for evaluating our approach, we materialize the links in an enterprise knowledge graph (EKG), which is a data structure that maintains all the links and makes them accessible to users.

Scope of our solution. In this paper, we focus on generating links for data discovery. When working with large repositories of datasets, our aim is to offer links that allow users to quickly narrow down their discovery needs to a handful of datasets. There is nothing fundamental that prevents our solution from finding links between individual values as well. Doing so, however, risks polluting results with too many links, and could limit performance when working with billions of values.

III. SEMPROP MATCHERS

In this section, we first introduce word embedding techniques that map a single word to a vector representation (Section III-A). We then present coherent groups to combine embeddings together for multiple words when they are semantically close to each other (Section III-B). We then discuss the semantic matcher, SEMA in (section III-C). We finish the section with a brief discussion of some syntactic matchers we use in our evaluation (section III-D).

A. Word Embeddings

The distributional hypothesis states that words appearing in the same context share meaning [6]. Recent advances in language models based on distributed representations capture the intuition of the distributional hypothesis and have been used in different tasks such as topic detection, document classification, and named entity recognition. Word embeddings based on neural probabilistic language models [8] such as Word2Vec [9] or Glove [10] have achieved high accuracy in capturing different similarity metrics of words. For example, in one such model trained on 6 billion words from Wikipedia, the words *"salary"* and *"wage"* have a similarity of 0.68, the words *"kilometer"* and *"mile"* of 0.85, and the words *"isoform"* and *"protein"* (from our example) have a similarity of 0.88. Crucially, in the same way word embeddings capture words that are semantically similar, they capture dissimilarity as well, which is equally important for our task—we can use dissimilar words to identify false positives, as will be explained later. Once trained, a word embedding model takes as input a term t and outputs a vector x_t that represents t in some d-dimensional vector space, where d is a parameter of the trained model.

Unfortunately, we cannot directly use word embeddings for our problem for two reasons. First, word embeddings are learned for single words, but many source elements contain multiple words, and we want to measure their similarity with other multi-word elements, e.g., compare *"Drug Description"*

with *"Building Description"*. Second, oftentimes, words are not in the dictionary of the embeddings we have (such words are referred to as unknowns i.e., `unk`). To tackle these two problems, we introduce the concept of **coherent group**, which is a technique to combine word embeddings and deal with *unk* words. Before explaining coherent groups in detail, we discuss existing methods in the literature.

Combining Word Embeddings. Perhaps the most direct approach for combining word embeddings is presented in [11], which introduces a method to produce a vector from sentences and paragraphs. Although we could use this method in principle, its performance is not good enough when we have only attribute names, which consist of a few words and are typically not grammatically complete sentences. We demonstrate this in the evaluation section. Simpler approaches used in practice are to combine words using pairwise mean, min or max [12]. Of those, averaging word embeddings (AWE) is quite popular [13], [14], [15]. We found these approaches did not yield good results in our case. We also show this in the evaluation section (V-C). Other more principled approaches exist to combine word embeddings. For example, in [16], the authors fit a probabilistic model over the corpus of word embeddings and then apply the Fisher kernel framework [17] to combine multiple word embeddings into a single fixed-dimensional one. This would, however, require knowledge about a good probability distribution for the word embeddings and is computationally expensive. In contrast to these methods, we propose coherent groups, which is a practical and simple way of combining word embeddings that works well in practice for our problem of semantic propagation.

B. Coherent Groups based on Word Embeddings

The idea behind coherent groups is that, instead of combining the word vectors directly—which is challenging as discussed above—, we reason about the similarity of two groups of words, such as two attributes, through their pairwise similarities, i.e., the distances of their word representations in the embedding.

Setting. We have a set of elements, SE, where each $e_i \in SE$ contains one or more words, such as an attribute or class name. For example, an element e_i with two words, "Drug Description" will be represented by two vectors $x_1, x_2 \in WE$, where WE is the word embeddings available to us. We refer to the set of vectors that represent an element e_i as $V(e_i)$. Given two such elements e_1 and e_2, we want to determine whether they are semantically similar or dissimilar, according to the distance between the words' vector representations, i.e., the union of $V(e_1)$ and $V(e_2)$.

The coherency factor of a set of vectors, X, is the average of the all-pairs similarities between its elements:

$$F(X) = \frac{\sum_{x_i, x_j \in X} sim(x_i, x_j)}{|X|}$$

where $sim(x, y)$ is the cosine similarity, which is a good similarity measure for comparing two vector embeddings. A coherent group is a set of words with a coherency factor greater than a threshold, δ. Adding a new vector to a coherent group will either keep the group coherent, i.e., if $F > \delta$, or render it incoherent, i.e., $F < \delta'$. Intuitively, a coherent group contains words that fall in the same semantic space—they share some meaning. We explain next how we use them as part of the semantic matcher, and how we solve the second challenge of using word embeddings, dealing with unknown words.

C. Semantic Matcher using Coherent Groups

Our semantic matcher, SEMA, uses coherent groups to generate positive and negative signals. We refer to it as SEMA(+) when it produces positive signals, i.e., pairs of elements that form a coherent group, and SEMA(-) when it produces negative signals, i.e., the coherency factor of the elements' vectors is below δ'. To make this concrete, we define a function CG that takes a group of vectors and by computing their coherency factor determines whether they comprise a positive or negative signal:

$$CG(X) = \begin{cases} 1 & \text{if F(X)} > \delta \\ -1 & \text{if F(X)} \leq \delta' \end{cases}$$

where 1 indicates a positive signal and -1 a negative one. In the simplest case, $\delta = \delta'$. We explain how we use these groups to find links followed by a discussion of two special cases.

Finding Positive and Negative Signals. We first pre-process the source elements and classes by tokenizing the names and filtering stop words. We remove symbols such as '-' or '_', also optionally employ a *CamelCase* tokenizer that transforms words of the form *SocialSecurity* into *social security*. It is easy to add new tokenizers if necessary. The output of the pre-processing stage is a bag of words for each source element and for each class. Bag-of-words is a good abstraction for this application because source elements consist of a few words and not of grammatically complete sentences, for which the word order—which is lost with the BOW model—may matter.

We check which pairs of bag of words, i.e., attributes and classes, have a coherency factor above δ; these are the positive signals. We also check which ones have a coherency factor below δ'; these are the negative signals, which we will use to filter out false links produced by other matchers, as we will see in the next section. For example, a syntactic matcher may find a link between *Cell type* and *Bell type*, as they only differ in one letter. However, SEMA(-) indicates a negative signal, thus it can be filtered out. Before giving more details about how we use these in practice, we discuss special cases that occur while computing coherent groups.

Special Cases. We consider two special cases: (1) some words from our source elements or classes will not be in the vocabulary since word embeddings are pre-trained from a closed word set, and (2) some words are identical, which can cause them to be over-weighted.

Unknown words $\langle$`unk`$\rangle$. The first case occurs when a word does not exist in the embedding model we have. This can happen for two reasons. First, the word may be a domain specific word that is not commonly used, such as *namalwa*,

TABLE I: Example formats and errors

Chembl	EFO	CLO
P3HR-1	P3HR1	P3HR-1 cell
UMUC3	UM-UC-3	UM-UC-3 cell
G-401	G401	G-401 cell
SNU-16	SNU16	SNU-16 cell
NAMALVA	NAMALWA	NAMALWA cell
Lyphoma	Lymphoma	lymphoma

which corresponds to a drug. Second, data may be dirty: typos in words would not likely appear in our vocabulary—unless they are so common that they are learned. For example, the name *namalwa* is not in the word embedding dictionary we learn and, in addition, in the ChEMBL database it is misspelled as *namalva* (see Table I). The word *Lymphoma* is in our vocabulary, but is also misspelled in the ChEMBL database, which prevents us from obtaining an embedding for it. When we cannot find a word in our dictionary for either of the previous cases, we say we have an $\langle \texttt{unk} \rangle$.

We can treat `unks` in two different ways. First, we can simply ignore them—not taking them into account while computing the coherency factor. Alternatively, we could penalize the similarity metric when we find an `unk`, for example, by saying that $sim(x, y)$ in that case is 0, or some negative value. The intuition for the first case is that if the data is dirty we should not penalize the existence of `unk`, but we should do it if the word is too specific and cannot be compared with other words. In practice, making this decision depends on the quality of the learned dictionary as well as the perceived quality of the existing links. For our evaluation, we choose to penalize *unks* with a similarity of 0 because data is generally clean.

Discussion on model specificity. There is a tension between training a word embedding model on either a domain-specific corpus or a general purpose one. The former may lead to fewer *unks*, while the latter may lead to a better model of the language, therefore capturing the semantic relationship of words in a more precise fashion. We evaluate experimentally the practical differences of using models trained on different corpora in the evaluation section (V-C)

Treating identical words. The second special case occurs when we are comparing words that are exactly the same, i.e., their similarity is 1. In this case, we have the choice to count these cases, which will contribute positively to the similarity value, or ignore them, which will put more weight on the other words in the group (our choice in the evaluation). One argument for the first option would be that we can use the semantic matcher to subsume a simple syntactic matcher that computes exact similarity of words. However, there are two reasons why trying to subsume syntactic matchers is a bad idea. First, syntactic matchers are more sophisticated as they can find approximate matches, e.g., with an edit distance below some threshold. Second, if two words are exactly the same, but they are an `unk`, then SEMA will not find them, but a syntactic matcher will. For these two reasons, the semantic matcher cannot subsume a syntactic matcher (experiment in section V).

D. Syntactic Matchers

The semantic matcher we just presented focuses on finding links between source elements and ontologies. We use two additional matchers that are implementations of state-of-the-art instance-based and schema-based matchers [18]. We use these matchers to complement the semantic matcher, as well as to identify links to lookup tables, that are abundant in some databases of large organizations.

Instance-based Matcher. We use an instance-based matcher that computes the Jaccard similarity between two sets of data values. This matcher is useful when there are data instances available, e.g., we are finding links against a lookup table or a knowledge base (which contains data).

Syntactic Similarity of Names. There exist cases in which name similarity is high but our semantic matcher does not yield results because the words are not in the vocabulary the embedding was trained on. We use a syntactic matcher that finds the similarity of the attribute and table name, we call it SYNM. Despite its simplicity, it is very efficient in cases in which the vocabularies in a domain are well aligned. We use a state-of-the-art matcher in this case as well.

IV. SEMANTIC PROPAGATION

In this section, we first study how to combine the various matchers to compute links (Section IV-A). We then introduce a structural summarizer that curates links by using the structure available in ontologies (Section IV-B). We conclude by describing how to use the existing links to find additional ones by transitively following them in the ontologies (Section IV-C).

A. Orchestrating Matchers

The order of different matchers and matching operators changes the resulting links. For that reason, we need to decide what is a reasonable combination. We propose to use a directed acyclic graph (DAG) to combine these matchers, with the target of producing links with the best recall and precision.

Matchers and Matching Operators. To assemble such a DAG, we have different matchers—that produce semantic and syntactic links—and matching operators. A *matcher* takes source elements and ontologies as input and produces links. A *matching operator* takes links as input and outputs links. Matchers and matching operators are then orchestrated by a semantic propagation graph.

Semantic Propagation Graph. The *semantic propagation graph* is a DAG $G_s(V_s, E_s)$. Each node u in V_s can be a matcher or a *matching operator*. Each directed edge (u, v) represents the order in which the links flow between nodes in the graph. It is possible to combine matchers and matching operators into different DAGs. We allow users to configure DAGs, but we provide here a particular configuration, SEMPROP—which works well in practice—and explain its rationale.

SEMPROP graph. We use three different matchers (single stroke oval in Fig. 3) and three different matching operators (double stroke oval). Our matchers consist of: (i) the SYNM

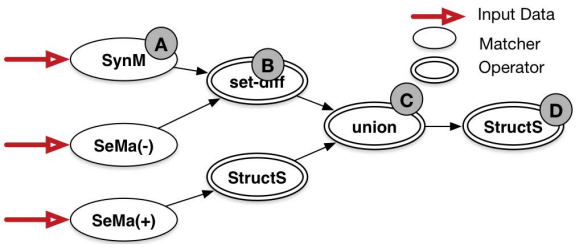

Fig. 3: Semantic Propagation DAG, SEMPROP

matcher which produces links based on the syntactic similarity of names of the source elements; and (ii) the SEMA matcher, which uses coherent groups to find positive links, SEMA(+), as well as negative ones, SEMA(-). The matching operators are the *union* and *set-difference* operators on input links, as well as the STRUCTS matcher (Section IV-B).

We give the source elements and ontologies to our matchers (red arrows in Fig. 3). One possible approach to compose our matchers is to first union the output of the semantic and syntactic matchers (annotation C in Fig. 3). However, while this would yield good recall, the precision may suffer due to the false positives produced by SYNM and SEMA, which add to each other. To avoid such situation, we first use the negative signals from the semantic matcher to remove false positives from the syntactic matcher, through the set-difference operator (B in the figure). We then add the positive matchings (C). Last, we use STRUCTS to curate the produced links.

Alternative semantic propagation graphs. In some cases, users may have domain knowledge to inform better orchestration plans. For example, for a repository of data in which schema names are not descriptive, users may opt out from using the attribute level matchers and rely only on the relation matchers. For very flat ontologies, where the structure will not provide much meaning, users may choose not to use the structural summarizer to avoid missing any true positive. We make it easy for users to tune the DAG based on their needs through a fluid API, i.e., method cascading, which is easy to use, and we perform some safety checks, such as warning of cycles in the DAG definition.

B. Structural Summarizer

Many of the output links of the SEMPROP DAG link the same source element to multiple different classes in an ontology. This is mainly due to two reasons: (i) no matcher is exact; they produce some spurious results and (ii) the same source element may match multiple classes but at different granularities, i.e., different levels of the ontology. In the latter case, these multiple links may be all correct, e.g., an attribute *Cell* may match to *Plant Cell* as well as to *Human Cell* and to the ancestor of these two classes, *Cell Line*. Some of the links may also be wrong, e.g., a syntactic matcher may identify *Cell Type* link to *Plant cell type*, *Human cell type*, and also *Bell type*.

Our structural summarizer, STRUCTS, summarizes the multiple links into fewer, more general links by exploiting how the classes are related to each other in the ontology. When all the links are correct, such as in the first example above, summarizing helps reduce the number of generated links generated transitively—which helps reduce redundant information. In the second case, summarizing can help eliminate spurious results, such as *Bell type*, which will not have the same ancestor as *Plant* and *Human cell type*.

Algorithm 1: Structural Summarizer

```
1  def structuralSummarizer(links, cuttingRatio):
2    hierarchySeqs = getSequences(links)
3    trie.addSeqs(hierarchySeqs)
4    (linksToSum, cutter) = findCutter(trie,cuttingRatio)
5    outputLinks = []
6    newLink = reduceTo(linksToSum, cutter)
7    outputLinks.add(newLink)
8    return outputLinks
9  def findCutter(trie, numSequences, totalNumSequences,
10                cuttingRatio):
11   chosenChild, numReprSeqs = chooseMaxReprChild(trie)
12   ratioCut = numReprSeqs / totalNumSequences
13   if ratioCut > cuttingRatio:
14     return findCutter(chosenChild, numReprSeqs,
15                       totalNumSequences)
16   else:
17     linksToSum = retrieveLeafsFromNode(chosenChild)
18     return linksToSum, chosenChild
```

Algorithm. STRUCTS is implemented using Algorithm 1. When a source element has more than a specified number of links, `summaryThreshold`, we invoke STRUCTS with the set of found links (note the check for `summaryThreshold` is not in the algorithm; we start with the invocation at line 1). We obtain the list of *ancestor sequences*, `hierarchySeqs`, for each of the classes of the ontology (line 2), with each sequence starting with the root of the ontology and finishing with the corresponding class. We then represent all the sequences in a trie (line 3). Each node of the trie maintains a counter of the number of sequences that contain it. At this point, we want to find the `cutter` node, i.e., the most specific ontology class that summarizes the largest number of links (line 4).

The `findCutter` function (line 10) finds the deepest node in the trie that summarizes at least a percentage of the total links, a concept captured by the heuristic variable `cuttingRatio`. The function chooses the child of the current node (line 12) that represents the largest number of links—this information is kept in the counter we maintain while feeding the trie with sequences. When there is no child that represents more links than `cuttingRatio`, the function returns all the children of the current node (line 15) as well as the current node, which becomes the cutter. At this point, we have obtained a subset of the input links, `linksToSum`, that can be summarized with the `cutter` class. The algorithm now checks whether it is possible to summarize a large enough number of links, given by the `cuttingRatio` heuristic and if so, it creates a new link, `newLink` that summarizes the represented links (line 6).

We use STRUCTS twice in our SEMPROP DAG (see Fig. 3). First, at the output of SEMA(+), to eliminate common spurious links produced by the semantic matcher. Second, at the end of the DAG, to curate the final links.

Intuition of the `cuttingRatio` parameter. If we aimed to summarize all the links, then we would force STRUCTS to find an ancestor class which also covers the potential false

Algorithm 2: Transitive Link algorithm

```
def findTransitiveLinks(matchings):
  links = []
  # class -> [sch1, sch2...]
  invMatchings = createInvMapping(matchings)
  for class in invMatchings:
    ancestors = ancestorsOfClass(class)
    for ancestor in ancestors:
      if ancestor in invMatchings:
        l = checkAndCreateLink()
        if l:
          links.add(l)
   relations = getRelationsOf(class)
   for r in relations:
     targets = getTargetsOf(class, r)
     for target in targets:
       if target in invMatchings:
         l = checkAndCreateLink()
         if l:
           links.add(l)
  return links
def checkAndCreateLink(sch1, sch2, r):
  if checks(sch1, sch2):
    return createLink(sch1, sch2, r)
  else:
    return False
```

positives in the links; this would lead to poor (very generic) links. Hence we need to omit some of the links. However, if we do not summarize at least a large fraction of links, we may be discarding valid links, therefore deteriorating the quality of the generated links. In short, we aim to summarize as many links as possible but not all—so as to not include false positives. This suggests the `cuttingRatio` parameter value should be high. We show this experimentally in the evaluation section.

C. Transitive Link Generation

When two source elements e_1 and e_2 have links to classes of an ontology, and the classes of the ontology have some relationship between them, we can create links by transitively following those relationships. We show the pseudocode for this simple approach in Algorithm 2. The produced links do not have a type, they just indicate that two source elements are related to each other. However, we annotate links with descriptive labels that indicate the reason why they were created.

Depending on how the classes are related in the ontology, we create three kinds of annotations. We create *isA* annotation when two source elements have a link to classes in the ontology that are hierarchically related. We create a *named-relation* annotation when the links against the classes of the ontology have some non-hierarchical relation, in which case we use the name of the relation as the name of the annotation. Last, we can create an *equivalence* annotation when links point to the same class in the ontology.

V. Evaluation

We want to evaluate the quality of the links found by SemProp. In particular, we ask: (i) What are the roles of the different matchers in SemProp? (ii) How do real users benefit from the addition of links in SemProp? (iii) What are the merits of our coherent groups? (iv) Can we have a faster approach to filter inadequate ontologies to make SemProp more efficient? (v) Can our techniques improve the quality of other state-of-the-art schema matching tools? (vi) Can SemProp help curate primary key foreign key (PKFK) relationships?

Datasets. We describe the datasets used in our evaluation:

- Chemical: We use the ChEMBL database and the DrugBank database. As reference data we use Experimental Factor Ontology (EFO), UniProt, and GO. ChEMBL and DrugBank have around 70 tables with around 600 attributes each. EFO has around 20K classes and GO around 80K. We obtained ground truth for this dataset by manually inputting every source element of the databases in the web interface of EFO, and then creating links to the first concept that appeared there. We handed our ground truth to domain experts to verify its quality. Following this laborious process, we found 127 links (about $1/6$ of the total source elements).
- Massachusetts Open Data: We use 300+ CSV files from the Massachusetts Open Data portal and used DBPedia [19] as reference ontology. We could not obtain ground truth for this dataset, so we use it as part of a user study.
- Environmental Data: We used around 30 open datasets related to Environmental Protection Agency (EPA) in the US, as well as some datasets from the open data initiatives of the governments of the UK and Australia. As a reference ontology, we use the environmental ontology [20]. We did not have a way of obtaining ground truth for this dataset, so we use it as part of a user study.

Setup. We used standard hardware for all our experiments: a Macbook Pro 3Ghz Intel Core i7 with 8GB RAM and an SSD. Our SemProp mechanism is implemented as part of Aurum [21], a data discovery system which builds, maintains and serves an enterprise knowledge graph (EKG): the structure in which we materialize all the links, and that we use throughout this evaluation. By default, we use word embeddings with 100 dimensions built with GloVe [10] on 6B+ tokens extracted from Wikipedia [22], which is the model that worked better for us. We also evaluate alternative models in Section V-C.

A. Merits of Different Matchers in SemProp

Our first experiment aims to understand the relative merit of the different matchers, as well as demonstrating the rationale for our SemProp DAG for combining matchers. For this purpose, we split SemProp DAG into different *slices*, which are labeled with letters from A to C in Fig. 3. We then executed each slice alone and measured the precision and recall of the output links with respect to the ground truth we obtained for the ChEMBL dataset, which we use here. We obtained four sets of results by configuring SynM with four different thresholds, $\delta = 0.2, 0.4, 0.6$ and 0.8. All these results are shown in Table II, which has the different slices as rows and the different SynM thresholds as columns, and shows precision, recall and the F-measure for each combination. The last line corresponds to running the entire SemProp DAG.

What is the quality of SynM? We used SynM to obtain a baseline for this experiment. SynM corresponds to slice

TABLE II: Slicing SemProp DAG (precision/recall (F-measure); numbers are in percentages. Columns indicate SynM threshold)

SemProp Slice	0.2	0.4	0.6	0.8
A	0.4/63 (0)	5/34 (8)	21/5 (8)	80/3 (6)
B	1/60 (1)	21/32 (25)	28/5 (8)	80/3 (6)
C	2/69 (3)	20/43 (27)	21/17 (18)	22/15 (18)
SemProp	**47/66 (55)**	41/43 (41)	29/22 (25)	28/20 (23)

A in SEMPROP (see table II). It uses the banding method of locality-sensitive hashing to identify source elements similar to ontology classes. We consider a link when the similarity is above a threshold δ. We configure the matcher with different thresholds that yield different tradeoffs in precision and recall. The results for slice A in Table II show the recall is higher with a low threshold. Because we prefer higher recall for discovery, we choose the SYNM threshold of $\delta = 0.2$. However, at this point users would find many spurious links in the EKG.

Example of wrong syntactic results: We show an example that explains the low precision of SYNM (slice A).

```
relationship_type -> Episodic Ataxia Type 7
drug_indication -> Serotonergic Drug
```

In the first example, the word *type* appears in both sides, which causes the link. In the second example, it is the word *drug* that appears in both sides of the link. When we look at the surrounding words, however, it is easy to see these are not semantically related to each other, even though there are entire words that are syntactically identical.

Does SEMA(-) help curate false positives? We used SEMA(-) to detect spurious cases (false positives) such as the one above; the results are B in table II. Ideally, the negative signals should cancel only false positives and not true positives, hence, increasing precision without affecting recall. Using as a baseline the results for SYNM with threshold 0.2, we apply set-difference, removing the negative signals found by SEMA(-). Doing this doubles the precision and maintains a high recall. Despite removing half of the false positives, hence the improvement in precision and recall shown in the table.

Example of false positives that are removed: The negative signals remove the examples shown above as well as other, less obvious, but wrong links such as:

```
site_components -> Biopsy Site
```

In this example, the word *site* in the ontology side refers to *the physical site from which a tissue was removed*. SEMA(-) captured that biopsy site and components do not often appear together—they are not semantically related—and deemed the link as incorrect, therefore cleaning a spurious result.

Can we improve recall with SEMA(+)? We take the results from SEMA(+) and add them to the previous result, which corresponds to slice C of the SEMPROP DAG (see table II). The recall improves consistently for all thresholds. Adding the links from SEMA(+) increases the recall from 60% to 69% as shown in Table II.

Example of correct results that are added: Some of the correct positive links that were added are:

```
isoform -> Protein
assays -> Pcr
```

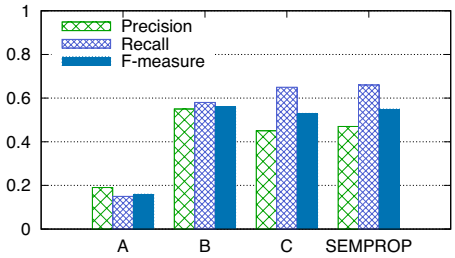

Fig. 4: The effect of adding STRUCTS after different slices (A, B, C) in the SEMPROP pipeline, with threshold set to 0.2. SEMPROP is optimized to achieve the highest recall.

The first one relates *isoform*, a type of protein, with *Protein*. The second one relates *Pcr*—polymerase chain reaction, a technique to amplify a segment of DNA—with assay.

Advantages of adding StructS. To understand better where STRUCTS helps the most, we run it after each slice (A/B/C) and compare the results with and without it. We configured the syntactic matcher with $\delta = 0.2$, which yields the best results. The results for this experiment are in Fig. 4.

Results: We choose $\delta = 0.2$ for this experiment. When we apply STRUCTS immediately to the output of SYNM (A in the figure), the precision increases up to 19%, but the recall drops to 15% from the original 63%. This is because the many false positives produced by SYNM means STRUCTS does not find structures to summarize links. When we apply STRUCTS after B, i.e., after eliminating false positives with SEMA(-), we see a boost in precision. In this case, STRUCTS has an opportunity to exploit any structure of the remaining links, hence further removing spurious ones. At this point we have reached the maximum F1 score in Fig. 4.

Why does the DAG continue then? The answer, one more time, is that we care mostly about recall for discovery. The precision value has an impact on how many links users must assess before finding a useful one. The recall value determines how many datasets we could connect successfully, therefore opening up more opportunities for discovery. For this experiment, ($\delta = 0.2$) Slice C, which adds links produced by SEMA(+) improves the recall and hurts the precision a bit. Running STRUCTS once again at the end of the pipeline, which is equivalent to running SEMPROP, increases the precision without hurting the recall. Therefore, for our recall-oriented application, SEMPROP achieves the best result.

How does STRUCTS affect other values of δ? Above, we explained that the strategy that leads to the best quality links is to configure the syntactic matcher with a low threshold and rely on SEMA(-) to filter out spurious links. Therefore, we chose $\delta = 0.2$. At the same time, we have seen that SEMPROP gives the best results (as per Fig. 4) for that value of δ, i.e., F-measure of 66. To verify that choosing a small δ is the best strategy, we repeated the experiment of Fig. 4, but for δ values of 0.4, 0.6 and 0.8 (the remaining in table II. The F-measure values without and with STRUCTS are, respectively, 21/41, 21/25 and 21/23; this is much worse than with lower thresholds. This makes sense: with higher values of δ there

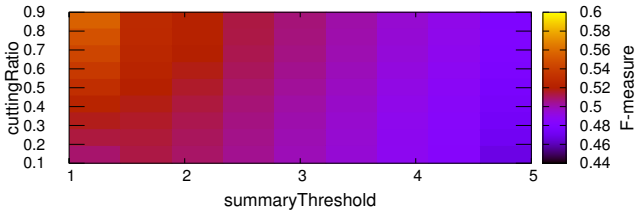

Fig. 5: Sensitivity of STRUCTS to `summaryThreshold` and `cuttingRatio` parameters on the ChEMBL dataset

are fewer links produced, which leads to fewer opportunities of summarizing them, thus a lesser impact of the STRUCTS.

Studying parameter Sensitivity of STRUCTS The quality of STRUCTS depends on the parameters: `cuttingRatio` and the `summaryThreshold`. The effect of these two parameters is shown in Fig. 5. The figure shows how the best F-score is achieved for `cuttingRatio` of more than 0.7, i.e., higher values indicate that the algorithm will only summarize when a large portion of the links are children of an ancestor. As mentioned above, higher values indicate the aim to summarize a large proportion of links but not all—therefore letting some headroom to discard false positives. The best values for the `summaryThreshold` are achieved as soon as there is more than one link between a source element and an ontology class. A higher value just prevents summarizing other links.

Our conclusion from this experiment is that the intuition that the two parameters aim to capture is indeed shown experimentally for this dataset.

Why the semantic matcher does not suffice to populate the EKG? As explained earlier in Section III, this is due to the existence of out-of-dictionary vocabulary. To confirm this, we ran an experiment in which we generated links using only the positive signals generated by SEMA. The maximum precision achieved is 19% and the recall only 12%, which confirms that using word embeddings alone does not solve the semantic propagation problem. Instead, as demonstrated earlier, they complement and improve recall and precision when used in combination with other matchers.

In summary, by judiciously combining our matchers and matching operators, SEMPROP achieves both good precision and recall. Our technique helps to automatically find links between source elements and ontologies with high quality without human intervention.

B. User Studies: SEMPROP in the Wild

In this section we evaluate the usefulness of the semantic links with a user study (Section V-B1), and with an interview with experts to get their feedback about our links (Section V-B2). We also report on an experiment to demonstrate the utility of links to surface relevant datasets from a large repository of sources (Section V-B3).

1) EKG User Study: The goals of our user study were to (i) learn whether the links help users understand complex schemas with which they are not familiar with and (ii) whether links created by SEMPROP are more useful for this task than those generated by using only syntactic matchers.

To evaluate these questions, we used the ENVIRONMENTAL DATA dataset. We built an EKG using links created by both SEMPROP and an alternative version that used only syntactic matchers (called SYNPROP).

We showed the individuals in the study the links from the two EKGs (we did not tell them they were from different EKGs). We then asked them to choose the group that better helps understand the underlying schema.

Study Procedure. We recruited 12 individuals with a computer science background, familiar with basic database concepts, and working in CS-related tasks.

Both SEMPROP and SYNPROP find a large number of links; to limit the number shown to users we applied stratified sampling. Specifically, each stratum was formed by links that connected the same table. We then selected randomly among those, reducing the total number of mixed links to 100.

To reduce the effect of congruence, expectation and avoid the framing effect, we split the links into 4 groups, which were randomized in a way that preserved the same ratio of links per split. We further randomized the order within each split 3 times, for a total of 12 splits. We then presented one split to each individual, so that 3 people would see the same set of links in a random order. Regarding the second task, we alternated the order in which we showed the groups to users.

Each user was given a short document (approx. 500 word) that introduced the data, the overall task, and an overview of the format of the links and meanings of the ratings. This included an example. Users were also given a spreadsheet with the generated links and asked to input their ratings.

The main result is that 8 out of the 12 users chose the group of links generated by SEMPROP as more relevant to understanding the schema than the links generated by SYNPROP, demonstrating that users appreciate the value of links generated by SEMPROP more than those generated by the modified version that uses only syntactic ones, SYNPROP.

To complement the previous result, we asked users to evaluate the usefulness of individual links. Users rated 29% of the semantic links as very useful, and 27% in the case of the syntactic links. However, users rated many more syntactic links as "not useful" (41% vs 34%), because syntactic links have not been filtered out by the semantic matcher, and therefore contain low quality links between unrelated data elements.

2) Real Scenario: We now describe the experience of using SEMPROP with expert users at a pharmaceutical company. We generated links for an internal database with over 200 tables (2.5TB), which is used daily by around 1000 analysts. Analysts have usually access to an ER diagram of the database, as well as an internal wiki page that the employees have created. We wanted to understand whether the EKG was useful to facilitate the navigation of the schema. We created an ontology with the concepts and relationships that we extracted manually from our interpretation of the wiki and used it as the reference data (the validity of the ontology was independently verified by a user of the database and one of the maintainers of the wiki page). We then interviewed a senior analyst, who is a user of the database, and the original author of the wiki page.

We highlight here some impressions: 1) The identified links are useful to quickly locate data. Because the ontology captures the information in the wiki page, which is written with vocabulary our users were familiar with and contains tacit knowledge to the company, the links quickly related those concepts with specific relevant source elements in the schema; 2) links between source elements are useful to understand the semantics of PKFK relationships—this is a feature we had not anticipated (evaluated in section V-F).

Follow-up session: We built an EKG of their internal database, ChEMBL [4] and DrugCentral [3], and we used EFO [7], GO [23] and Uniprot [24] as reference data. Our goal was to find links across databases which would help them understand how their schemas relate to each other. Some of the links were especially insightful, e.g., (”chembl22.variant_sequences.isoform”) (”is_a”) (”drugcentral.target_dictionary.protein_type”.) In this session, they pointed out that the ontologies are updated frequently, and new ones appear often; it would be good to know quickly which ontologies are helpful to explain concepts in the data they know. This inspired us to provide an approximate, but fast implementation of SEMPROP which helps analysts to quickly decide if the ontology is worthy, before adding it to the EKG (section V-D). We further discuss this feature and its performance later in this section.

3) Surfacing relevant Datasets: In this experiment, we wish to demonstrate the utility of links to surface, i.e., highlight, relevant datasets from within a large repository of heterogeneous and independently created sources. We used the MASSACHUSETTS OPEN DATA (described at the beginning of the section) for this experiment.

We generated around 300 links between tables of the repository, which we gave to three users, who had the task of determining whether the tables in each link were relevant to each other or not. We took the majority voting out of those ratings and found that the users agreed on 58% of the links being relevant. This is a good result because it means that users find 1 relevant link for each pair of links we generate. Since the baseline is not having any links at all, users would need to compare, for each table of interest, whether it is related to all the other tables, a time-consuming manual effort.

In conclusion, we demonstrated that the semantic links generated by SEMPROP are better than those generated by a syntactic matcher alone with 3 different datasets. One, CHEMBL for which we had ground truth, and two others, ENVIRONMENTAL DATASET and MASSACHUSETTS OPEN DATA that we used as part of user studies.

C. Evaluating Coherent Groups

SEMPROP depends on SEMA, which depends on coherent groups. We explore other methods for combining WE as well as WE trained on different corpuses.

Why not using other methods to combine word embeddings? We implemented 3 other methods to combine word embeddings [12]. Averaging WEs reduces by 30% the precision and recall achieved with coherent groups. While adding WEs or concatenating and padding WEs performs slightly better than averaging, they still do significantly worse than coherent groups. In fact, in this case, many of the hits are due to the other syntactic matchers of SEMPROP. An alternative to combine word embeddings is *doc2vec* [11], which we trained with the descriptions and documentation of EFO, GO and ChEMBL. It achieved a precision of 36% and recall of 46%, again underperforming coherent groups.

TABLE III: Comparison of methods for combining WE

Methods	Wikipedia	ChEMBL +EFO +GO	PubMed
Coherent Groups	**47/66**	17/60	24/55
AWE	34/47	12/45	23/49
Adding	41/50	14/50	20/49
Concatenate	40/49	13/50	20/41

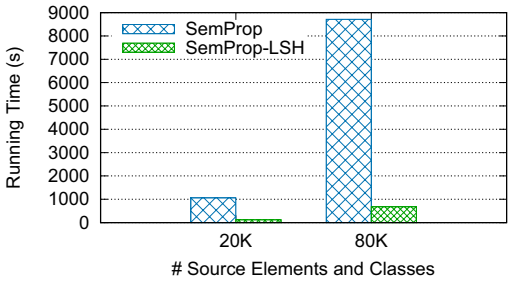

Fig. 6: SEMPROP vs the LSH-based method

Do we benefit from WE trained with domain-specific data? We used word embeddings trained with Wikipedia data as well as with the corpus used to train doc2vec, and with a corpus we generated from PubMed [25]. The PubMed word embeddings consist of 17M vectors that cover the life sciences domain. The model trained with Wikipedia data achieved the best results. This is not surprising, a WE model is as good as it models the language. Presumably the Wikipedia corpus covers the English language better than the PubMed one.

D. A Fast Approach for Filtering Ontologies

Because the running time of SEMPROP increases as new ontologies and databases are added, but not every ontology is useful for discovery, we found it useful to quickly peek through links before deciding to fully run SEMPROP to completion. Fortunately, coherent groups have a property that makes it easy to implement a lightweight, fast method. At their core, coherent groups rely on dot products of vectors to obtain the cosine similarity. Thus, they can be implemented using locality-sensitive hashing [26], which can reduce the complexity of finding links from $O(n^2)$ (all-pairs comparison) to $O(n)$. We implemented a lightweight version of the semantic propagation mechanism, SemProp-LSH, that relies only on an LSH-based implementation of coherent groups. This approach uses LSHForest [27] and indexes the source elements twice. The first index corresponds to the vector embedding obtained directly from the dictionary, which we use to find positive links. The second index corresponds to a rotation of the vector that permits us to find negative links.

TABLE IV: Results of running SEMPROP on COMA baseline

Method	Sim. Threshold	Precision (%)	Recall (%)	F1 (%)
COMA	0.2	18	35	23
	0.3	23	35	27
	0.4	49	24	32
COMA+B	0.3	48	31	37
COMA+C	0.3	34	43	37

To be useful, SEMPROP-LSH must run much faster than the original semantic propagation mechanism and still yield good links for users to be able to make a decision. Fig. 6 shows an order of magnitude difference between SEMPROP-LSH and SEMPROP; thus, it is useful to bootstrap the search with the LSH-based approach to preselect which ontologies are more promising. For example, when running ChEMBL and DrugCentral with EFO and GO (near 90K source elements and classes), we obtain results within 10 minutes if we use the lightweight method, in contrast to the 2 hours required by the vanilla semantic propagation method. We observed that the GO ontology in this case does not add many links to either database, but contributes close to 60K classes, making the process slower. By doing an early assessment with the LSH-based method, we can easily prune the GO ontology, reducing the running time to 16 minutes.

E. Improving a State-of-the-Art Matcher

Schema matching tools are not appropriate for semantic propagation because they typically involve human interaction and only produce matchings to the reference data. However, we want to understand if SEMPROP can improve the quality of the matchings obtained from these tools, so that we can leverage the advanced techniques for the semantic propagation problem. To this end, we used the community edition of COMA 3.0 (COMA for short), to find matchings between source elements and ontology classes. We then use our SEMPROP DAG, where we replace SYNM with COMA.

COMA setup. COMA provides an initial suggestion of the matchings for a user to confirm/reject. It provides context-dependent matching, fragment-based matching and reuse oriented matching. To obtain the best results possible, we used all the possible context-dependent matchers (that ran to completion) and manually interacted with the tool as requested.

Baseline COMA results. We run COMA with different thresholds; 0.3 yields the best results, 23% precision and 35% recall. Note that the precision is much higher than that of our plain SYNM matcher, while the recall is lower, see A in table II. This makes sense – COMA is a schema matching tool designed to produce correspondences for data exchange and integration. High precision is important, as wrong results lead to bad correspondences. In contrast, in semantic propagation it is more important to achieve high recall, making sure no relevant links are left behind. The question we evaluate here is whether COMA can benefit from SEMPROP.

COMA + SEMPROP results. We choose the threshold that yields the highest recall for COMA, i.e., $\delta = 0.3$ as our starting point. When removing false positives with SEMA(-) (corresponds to B) we doubled the precision, from the original 23% up to 48%. It is interesting to look at sample false positives we were able to remove:

```
class_level -> Clark level
mesh_heading -> reading
```

At the same time the recall slightly reduces to 31% from 35%. However, as in the case of SEMPROP, adding the positive links with SEMA(+) (C in the table), raises the recall up to 43%, reducing this time the precision to 34%. We chose the 0.3 threshold because it yielded the highest recall. For completeness we also run our mechanism on COMA with threshold of 0.4 (highest F-measure), but the end result is an F-measure of 36%, with precision of 40% but a lower recall of 33%.

F. Semantic-based Curation for PKFK

Although this paper focuses on finding links, we envision the EKG to maintain other types of links, such as PKFKs, as these indicate how to join tables, and hence are useful for discovery tasks. Because finding PKFKs across many databases is an expensive process, there are approximate methods in the literature, such as Powerpivot [28], which has a much lower running time at the cost of some false positives PKFKs.

We want to evaluate how well SEMPROP can help curate some of those false positives. If we find a negative link between a candidate PKFK, we remove it, as it is likely a false positive. For example, many keys are integers that span a similar range, which leads to false positives such as a *drug_id* being a PKFK with *employee_id*. We use SEMA(-) to find negative signals, which may indicate false positives. Then we remove those.

We ran an implementation of the techniques from Powerpivot and obtained a list of candidate PKFKs for ChEMBL. We obtained 972 PKFK candidates, which, using the declared PKFKs in the ChEMBL schema, correspond to a precision of 6% and a recall of 63%. By using SEMPROP, we were able to remove 444 of those candidates, that is 45% of the candidates. The good news is that all the removed candidates were false positives, thus leaving the recall intact. Some examples of false positive candidate PKFKs detected by SEMPROP are:

```
irac_classif.irac_class_id -> docs.volume
hrac_classif.hrac_class_id -> ligand_eff.sei
```

VI. RELATED WORK

Discovery and EKG. Our work is related to Goods [29], Helix [30] and the Q system [31]. Goods [29] aims at finding similar datasets by harnessing query logs, instead of creating semantic links with a mechanism such as the one presented here. Helix [30] uses an instance-based approach to discover links; it does not work with pure ontologies and it does not leverage word embeddings. Last, the Q system [31] answers discovery queries through a *template-based* interface. The system represents *is-a* connections used to query ontologies, and not to find additional links in the data.

Schema and Ontology Matching. We consider:

(1) Table-Table matching. Tools such as COMA [32], Harmony [33], S-Match [34] and CLIO [35] have been developed for schema matching. Most available tools are interactive tools

with a focus on finding the *is-a* relationship between attributes from different relations. In contrast we (i) find relationships beyond the *is-a* type to facilitate data discovery; and (ii) leverage ontologies to link tables.

(2) Table-Ontology matching. We differentiate between: (i) column understanding, and (ii) column pair understanding within the same table. For (i), there are techniques for discovering schema semantics [36], [37], which are not automatic and require human intervention. For (ii), there are approaches for annotating Web tables using knowledge bases [38], [39], which is also known as *table understanding*. We are different in that: (a) We need an automatic process to bootstrap the link generation without human intervention. (b) They are designed to explain one table from one ontology, instead of finding relevance across many tables via multiple ontologies; (c) none of them leverage the benefits of word embeddings.

(3) Ontology-Ontology Matching. Link discovery between ontologies [40] finds *is-a* relationships across knowledge bases. Tools such as CODI [41] and LogMap [42] were developed for this purpose. We differ in that we target finding relationships across relations by leveraging ontologies and use word embeddings. Their output, *is-a* relationships among ontologies, is complementary to semantic propagation.

Ontology Based Database Access. Systems like BootOX [43] are designed for transforming relational databases into resource description framework (RDF) ontologies and generate the links between the database entities and the generated ontology classes. Similarly, Ontop [44] and D2RQ [45] are proposed to query RDMSs using SPARQL queries by viewing an RDMS as a virtual RDF graph. These tools do not generate links between source elements and reference data.

VII. Conclusions

Seeping Semantics revolves around two ideas: i) coherent groups, which leverage word embeddings to assess the semantic similarity across elements that contain multiple words, and ii) an orchestrator that ensembles multiple matchers to identify high quality links. We implemented SemProp as part of Aurum[21][1], our data discovery prototype, and evaluated its effectiveness in real scenarios.

Acknowledgements: We thank the users and collaborators that participated in the user studies.

References

[1] A. Y. Halevy, A. Rajaraman *et al.*, "Data Integration: The Teenage Years," in *VLDB*, 2006.
[2] C. Zhang, C. Ré *et al.*, "DeepDive: declarative knowledge base construction," *Commun. ACM*, 2017.
[3] DrugCentral, "DrugCentral," http://drugcentral.org/download, 2017.
[4] DB: ChEMBL_21, "ChEMBL: a database of bioactive drug-like small molecules," https://www.ebi.ac.uk/chembl/downloads, 2016.
[5] J. Goikoetxea *et al.*, "Single or Multiple? Combining Word Representations Independently Learned from Text and WordNet," in *AAAI*, 2016.
[6] Z. S. Harris, "Distributional structure," *Word*, 1954.
[7] EFO, "EFO: Experimental factor ontology," http://www.ebi.ac.uk/efo/, 2017.

[1] https://github.com/mitdbg/aurum-datadiscovery

[8] Y. Bengio, R. Ducharme *et al.*, "A Neural Probabilistic Language Model," *JMLR*, 2003.
[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR Workshop*, 2013.
[10] J. Pennington, R. Socher *et al.*, "GloVe: global vectors for word representation," in *EMNLP*, 2014.
[11] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *ICML*, 2014.
[12] C. De Boom *et al.*, "Repr. Learning for Very Short Texts Using Weighted Word Embedding Aggregation," *Pattern Recogn. Lett.*, 2016.
[13] T. Kenter, A. Borisov *et al.*, "Siamese CBOW: Optimizing Word Embeddings for Sentence Representations," *ACL*, 2016.
[14] H. Zamani and W. B. Croft, "Estimating Embedding Vectors for Queries," in *ICTIR*, 2016.
[15] I. Vulić *et al.*, "Monolingual and Cross-Lingual Information Retrieval Models Based on (Bilingual) Word Embeddings," in *SIGIR*, 2015.
[16] S. Clinchant and F. Perronnin, "Aggregating continuous word embeddings for information retrieval," in *CVSC*, 2013.
[17] T. S. Jaakkola and D. Haussler, "Exploiting Generative Models in Discriminative Classifiers," in *NIPS*, 1999.
[18] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *VLDB*, 2001.
[19] P. N. Mendes *et al.*, "DBpedia: A Multilingual Cross-domain Knowledge Base," in *LREC*, 2012.
[20] P. L. Buttigieg, C. Mungall *et al.*, "envo," https://github.com/EnvironmentOntology/envo, 2017.
[21] R. C. Fernandez, Z. Abedjan *et al.*, "Aurum: A data discovery system," in *ICDE*, 2018.
[22] G. P. built models, "GloVe: global vectors for word representation," https://nlp.stanford.edu/projects/glove/, 2017.
[23] GO, "GO: Gene ontology consortium," 2017.
[24] UniProt, "UniProt: Protein ontology," http://www.uniprot.org/database/DB-0181, 2017.
[25] PubMed, "MEDLINE/PubMed Data," https://www.nlm.nih.gov/databases/download/pubmed_medline.html, 2017.
[26] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *STOC*, 1998.
[27] M. Bawa, T. Condie *et al.*, "LSH Forest: self-tuning indexes for similarity search," in *WWW*, 2005.
[28] Z. Chen, V. Narasayya *et al.*, "Fast Foreign-key Detection in Microsoft SQL Server PowerPivot for Excel," *VLDB*, 2014.
[29] A. Halevy, F. Korn *et al.*, "Goods: Organizing Google's Datasets," in *SIGMOD*, 2016.
[30] J. Ellis, A. Fokoue *et al.*, "Exploring Big Data with Helix: Finding Needles in a Big Haystack," *SIGMOD Rec.*, 2015.
[31] P. P. Talukdar, M. Jacob *et al.*, "Learning to Create Data-integrating Queries," *VLDB*, 2008.
[32] S. Massmann, S. Raunich *et al.*, "Evolution of the COMA Match System," in *OM*, 2011.
[33] L. Seligman, P. Mork *et al.*, "OpenII: an open source information integration toolkit," in *SIGMOD*, 2010.
[34] F. Giunchiglia, P. Shvaiko *et al.*, "S-Match: an algorithm and an implementation of semantic matching," in *ESWS*, 2004.
[35] R. J. Miller, M. A. Hernández *et al.*, "The Clio Project: Managing Heterogeneity," *SIGMOD Rec.*, 2001.
[36] M. Taheriyan, C. A. Knoblock *et al.*, "Leveraging Linked Data to Discover Semantic Relations Within Data Sources," in *ISWC*, 2016.
[37] ——, "Learning the Semantics of Structured Data Sources," *Web Semant.*, 2016.
[38] P. Venetis, A. Halevy *et al.*, "Recovering Semantics of Tables on the Web," *VLDB*, 2011.
[39] C. A. Knoblock, P. Szekely *et al.*, "Semi-automatically Mapping Structured Sources into the Semantic Web," in *ESWC*, 2012.
[40] M. Nentwig, M. Hartung *et al.*, "A survey of current link discovery frameworks," *Semantic Web Journal*, 2015.
[41] J. Huber, T. Sztyler *et al.*, "CODI: combinatorial optimization for data integration–results for OAEI 2011," in *OM*, 2011.
[42] E. Jiménez-Ruiz *et al.*, "LogMap 2.0: towards logic-based, scalable and interactive ontology matching," in *swat4ls*, 2012.
[43] ——, "BootOX: practical mapping of RDBs to OWL 2," in *ISWC*, 2015.
[44] D. Calvanese *et al.*, "Ontop: answering SPARQL queries over relational databases," *Semantic Web Journal*, 2017.
[45] C. Bizer and A. Seaborne, "D2RQ - treating non-RDF databases as virtual RDF graphs," in *ISWC*, 2004.