DomainNet: Homograph Detection and Understanding in Data Lake Disambiguation

ARISTOTELIS LEVENTIDIS, LAURA DI ROCCO, WOLFGANG GATTERBAUER, RENÉE J. MILLER, and MIREK RIEDEWALD, Northeastern University, USA

Modern data lakes are heterogeneous in the vocabulary that is used to describe data. We study a problem of disambiguation in data lakes: *How can we determine if a data value occurring more than once in the lake has different meanings and is therefore a homograph?* While word and entity disambiguation have been well studied in computational linguistics, data management, and data science, we show that data lakes provide a new opportunity for disambiguation of data values, because tables implicitly define a massive network of interconnected values. We introduce `DomainNet`, which efficiently represents this network, and investigate to what extent it can be used to disambiguate values without requiring any supervision.

`DomainNet` leverages network-centrality measures on a bipartite graph whose nodes represent data values and attributes to determine if a value is a homograph. A thorough experimental evaluation demonstrates that state-of-the-art techniques in domain discovery cannot be re-purposed to compete with our method. Specifically, using a domain discovery method to identify homographs achieves an F1-score of 0.38 versus 0.69 for `DomainNet`, which separates homographs well from data values that have a unique meaning. On a real data lake, our top-100 precision is 93%. Given a homograph, we also present a novel method for determining the number of meanings of the homograph and for assigning its data lake attributes to a meaning. We show the influence of homographs on two downstream tasks: entity-matching and domain discovery.

CCS Concepts: • **Information systems → Information integration**; *Data cleaning*; *Extraction, transformation and loading;*

Additional Key Words and Phrases: Data Discovery, homograph detection, network-centrality measures

ACM Reference format:
Aristotelis Leventidis, Laura Di Rocco, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. DomainNet: Homograph Detection and Understanding in Data Lake Disambiguation. *ACM Trans. Datab. Syst.* 48, 3, Article 9 (September 2023), 40 pages.
https://doi.org/10.1145/3612919

1 INTRODUCTION

We consider data lakes that are large repositories of tables where table names, attribute names, and attribute descriptions may be incomplete, ambiguous, or missing [67]. Such table repositories are heterogeneous in many different ways: The same data value (i.e., the value of an attribute in a

This work was supported in part by the National Science Foundation (NSF) under award numbers IIS-1956096 and CAREER IIS-1762268.

Authors' address: A. Leventidis, L. Di Rocco, W. Gatterbauer, R. J. Miller, and M. Riedewald, Khoury College of Computer Sciences, Northeastern University, 360 Huntington Ave, Boston, MA 02115; emails: {leventidis.a, la.dirocco, w.gatterbauer, miller, m.riedewald}@northeastern.edu.

0362-5915/2023/09-ART9 $15.00
https://doi.org/10.1145/3612919

table) or attribute name may refer to different things in different tables, and different values may refer to the same thing. We consider data lakes containing only tabular data and study the specific problem of determining whether a data value that appears in more than one table has multiple meanings. A data value with more than one meaning is a *homograph*. We illustrate the data lake disambiguation problem through an example.

Example 1.1. Consider the small sample of a data lake in Figure 1 showing four tables about different topics. T1 is about corporate sponsorship for efforts to save at-risk species, T2 is about populations in zoos, T3 is about car imports, and T4 is about corporate sales.

In a well-designed database or data warehouse where the semantic types of all columns are known, we can detect homographs by looking for values that appear in more than one distinct semantic type. However, in modern table repositories, we often do not have complete, consistent metadata describing column types, and the names of columns may be missing or uninformative [67]. Another approach to tackling this problem would be to apply word sense disambiguation from **natural language processing (NLP)** [69] or topic modelling [4] by treating each table as a document. Such techniques are excellent at discerning the meaning of words or topics of documents in the context of natural language. However, because of the nature of tables that are often used to express relationships between different types of entities and values, distinguishing between a donor table $T1$ and a zoo table $T2$ is a difficult task. Both tables contain overlapping values, but the context (the information about these values) is quite different in the two tables. Distinguishing between car manufacturers `T3.C2` and corporations `T4.Name` can be even harder because of the prevalence of numerical values.

Entity resolution and disambiguation methods commonly assume a small set of tables about a small number of entity types (which may have the same or different schemas) [21]. In contrast, in a table repository the values to be disambiguated may appear in tables about very different entity types and relationships between them. The ambiguous values need not be named entities, but may be descriptors or any data value in a table. This makes entity resolution inapplicable but opens up new opportunities to use the large network of values and co-occurrences of values in new ways.

Moreover, a data value with multiple meanings may not always be well formed like "Jaguar." For instance, null equivalent values such as the value "-" that appears in the `T1.At_Risk` and `T2.num` columns can also be considered as a value with multiple meanings, as it appears in different contexts (i.e., co-occurs with different sets of values). Besides null equivalent values, misplaced values (i.e., values that were meant to be placed in a different column) can also "appear" as values with multiple meanings. For instance, the values "BMW" and "X4" in table `T3` were misplaced (swapped), so now "BMW" appears as a car model and "X4" as a car manufacturer. With the presence of more tables that correctly list car models and their respective car manufacturers, the values "BMW" and "X4" will appear to have multiple meanings due to this misplacement.

The goal of **entity resolution (ER)** [21] in tables is to determine if two (or a set of) tuples refer to the same real-world entity or not. An important assumption in ER is that the tables being resolved are about the same known entity types. As an example, given a set of tables about papers that include authors as data values, we can determine if two tuples refer to the same paper (i.e., have the same meaning) or different papers. As part of this process, a data value, for example, the data value "X. Wang," may be identified as an ambiguous data value that refers to more than one real-world entity. Schema-agnostic ER techniques have been proposed that do not assume the entities are represented by the same schema [73]. However, these approaches still assume the tables being resolved represent entities of the same type. They would not work for our example, as each table represents entities of different types. In our problem, we are not starting with a small

*T*1

Google	Panda	1M
Volkswagen	Puma	2M
BMW	Jaguar	0.9M
Amazon	Pelican	1.5M
Toyota	-	1500

*T*2

name	locale	num
Panda	Memphis	2
Panda	Atlanta	2
Lemur	National	20
Jaguar	San Diego	8
Pelican	Boston	-

*T*3

C1	C2	C3
XE	Jaguar	UK
Prius	Toyota	Japan
500	Fiat	Italy
BMW	X4	Germany

*T*4

Name	Revenue	Total
Jaguar	25.80	43224
Puma	4.64	13000
Apple	456	370B
Toyota	123	128B

Fig. 1. Running example with data values "Jaguar" and "Puma" having meaning: animals in tables *T*1 and *T*2, and companies in tables *T*3 and *T*4. The value "-" appearing in tables *T*1 and *T*2 represents a missing (or null) value. Values "BMW" and "X4" in table *T*3 are misplaced (swapped due perhaps to a data entry error). How can we use co-occurrence information across a large set of tables to identify values with different meanings as well as null equivalent and misplaced values? In particular, we focus on the problem without relying on information in the header names, which is commonly inconsistent or missing in large open table repositories.

set of tables whose tuples are known to refer to the same type of real-world entities, e.g., all tables contain customer tuples or a small number of types such as research papers, conferences, and authors. We want to understand within a repository with a massive number of tables if the value "Puma" in attribute `T1.At_Risk` (see Figure 1) refers to the same real-world concept as "Puma" in attribute `T4.Name`. Furthermore, we also want to identify null equivalent values and misplaced values ("-" and "BMW" with "X4," respectively, in Example 1.1), which "appear" as if they have multiple meanings.

Disambiguation of words in documents has also been heavily studied [10, 44, 84, 93]. Solutions often rely on language structures or labeled training data. In contrast to documents, which are free text, tables are structured and lack the same intuitive notion of *context* that natural language has. However, they have a different form of context conveyed by the tabular structure. While plenty of research has explored disambiguation of documents, to the best of our knowledge there is no work on disambiguation of large table repositories. This is of importance, because these repositories can contain many data values that have different meanings. As an example, "Not Available" is a well-known way to represent NULL values in a table. "Not Available" is not ambiguous from a natural-language point of view. However, in a data lake it may appear in multiple attributes corresponding to names, telephone numbers, IDs, and so on, making "Not Available" a homograph, meaning "unknown name" or "unknown phone number," and so on.

Determining if a value in a data lake has a single or multiple meanings is unexplored territory. More specifically, our goal is to determine for each value in a collection of tables with possibly incomplete or heterogeneous table and attribute names if it has a single or multiple meanings. Values with more than one meaning (e.g., *Jaguar*) are homographs.[1]

A homograph is not necessarily a single word from a dictionary or a vocabulary. In a data lake, a homograph can be a phrase, initialism (e.g., "NA"), identifier, number, or any blob (data value). We do not assume homographs to be named entities; they can be adjectives or another part of speech. Homographs arise naturally from words used in different contexts, e.g., the classic example of *Apple* as a fruit or a company, or *Jaguar* in Example 1.1. They can also arise due to errors, e.g., when car manufacturer "BMW" is accidentally misplaced into the car model column and vice versa for "X4" in Example 1.1. We consider these now-ambiguous values as homographs. We consider the entire contents of a cell value (attribute value) in a table (i.e., we do not split multiple word values besides trimming off leading and trailing white-space and enforcing the same capitalization) so

[1]A homograph is generally defined as each of two or more words spelled the same but not necessarily pronounced the same and having different meanings. Homonyms, in contrast, generally need to be pronounced the same way and, hence, we consider only homographs.

homographs can be composed of more than one word and can be either textual, numeric, or a combination of both. Notice that updates to the data lake can change a homograph to a value with a single meaning, e.g., when the table with the only alternative meaning is removed; and vice versa.

In this work, we examine the global co-occurrence of data values within a data lake and how such information can be used to disambiguate data values. We show that a local measure is not sufficient and motivate why and how the full network of value co-occurrences enables effective disambiguation. This network exploits table structure that has not been considered in the most commonly studied disambiguation problems such as named-entity disambiguation and entity resolution. Its disambiguation power comes at a price: The value co-occurrence information is massive and it is not obvious how to process it efficiently for disambiguation.

Contributions. We address the data lake disambiguation problem (formally defined later in Definition 1) using a network-based approach called `DomainNet`. Our main contributions are as follows:

(1) We define the problem of homograph detection in data lakes (Section 3.1). Homographs may arise in tables that do not represent the same (or even similar) types of entities, and hence cannot be identified using entity resolution and disambiguation, which we discuss in Section 2.1. The tables may represent complex relationships rather than entities. Homographs may not even be words in natural language and do not appear in natural-language contexts, making language models ineffective.

(2) We present `DomainNet`, a network-based approach that provides a ranked list of data values most likely to be homographs in a given data lake. `DomainNet` is motivated by work on community detection where a community represents a meaning or domain of values (e.g., the value "Jaguar" has two meanings and appears in two domains, namely: animals and car manufacturers). A homograph is then a value that occurs in multiple domains. However, in the homograph detection problem there are an *unknown* and possibly *large* number of meanings for a value, which makes standard community detection techniques applied to the entire dataset inaccurate as we show in Section 6.8. Instead, we identify two measures for finding community-spanning values, the *local clustering coefficient* [91] and the *betweenness centrality* [35], and apply them to a bipartite network representation of our input table repositories. We empirically evaluate their usefulness for homograph detection in Section 6. Our evaluation shows that the latter, despite its higher computation complexity, is a more suitable measure, as it is a global measure over the entire dataset, whereas the former is a local measure that is more prone to local biases in the data.

(3) We extend the capabilities of `DomainNet` so it can identify the number of meanings of a homograph in question as well as group its attributes based on their meaning (e.g., given the homograph "Jaguar," we identify that it has two meanings in Figure 1, and we place attributes `T1.At_Risk` and `T2.name` in one group and attributes `T3.C2` and `T4.Name` in another group, as they correspond to the different domains of animals and car manufacturers, respectively).

(4) We present an evaluation on a synthetic dataset (with *known* ground truth), studying the performance of both centrality measures and motivating the use of the more computationally expensive betweenness centrality. We compare `DomainNet` to a recent *unsupervised domain discovery* algorithm D^4 [71] (any value belonging to multiple domains is a homograph) and show that `DomainNet` performs much better in identifying and ranking the most likely homographs. Even though D^4 does take into account values that may appear as part of multiple domains, they do not handle numerical values well, and their pruning approaches are applied locally, which cannot always capture a global context. We also adapt *supervised semantic type detection* approaches such as Sherlock [43] and SATO [100] for the homograph detection

problem and show that `DomainNet` outperforms them in both real and synthetic datasets. Such supervised techniques cannot effectively deal with tables that contain semantic types beyond the ones they were trained on and thus cannot precisely identify homographs from heterogeneous tables. Additionally, we compare `DomainNet` against *overlapping community detection* algorithms, showing that they are unable to effectively identify homographs as belonging to multiple communities. More specifically, they only identify a small fraction of the homographs as members of multiple communities, resulting in the formation of fewer communities than the ground truth.

(5) We create a disambiguation benchmark from a set of real-world tables used in a recent table-union benchmark [68] and show that we can effectively find naturally occurring homographs in this data (93% of the first 100 retrieved values are homographs based on ground truth). We also create synthetic datasets by systematically introducing homographs into real data and show that betweenness centrality achieves 85% accuracy (for the first 50 retrieved values) when homographs are injected into attributes.

(6) We show the impact of homographs on two downstream tasks: domain discovery and entity matching. For domain discovery [71], we observed the algorithm's performance starts to go down with the introduction of as few as 50 homographs (injected into a clean unambiguous real data lake). As the number of homographs increases, the accuracy of the domain discovery algorithm deteriorates. We also show that the presence of homographs can impact the performance of a state-of-the-art entity matching technique (DITTO [59]) especially in highly textual datasets. These observations further motivate the usage of our `DomainNet` as a pre-processing step for various data integration tasks.

(7) The scalability of our approach depends on the size of the data lake vocabulary (the number of values) and on the density of the network (number of edges), which is a measure of the overall co-occurrence of values across all tables. We use real data (from NYC open data) with a vocabulary size of 1.5M to show that we can construct the `DomainNet` network in 3.5 minutes and identify homographs in 27 minutes using an approximation of *betweenness centrality* based on sampling. Moreover, we show that the `DomainNet` network we use to compute *betweenness centrality* can be significantly compressed allowing for orders of magnitude speedup, bringing the *betweenness centrality* computation time in this case to well under a minute.

We first introduced `DomainNet` in Reference [56]. This extended journal manuscript extends our approach to tackle two new problems: (1) identify the number of meanings of a homograph and (2) group the attributes that a homograph appears in by their meaning (the new Section 4). We believe ours is the first method that determines the number of meanings of a given homograph in a large collection of tables and also the first to associate values within tables with a specific meaning.[2] We also use a compression technique that dramatically improves the scalability of our approach (the new Section 3.4). The experimental section has been expanded to evaluate the accuracy of our approach for assigning attributes to meanings (the new Section 6.4) and the effect of `DomainNet` compression on scalability (the new Section 6.6). Additionally, we compare `DomainNet` against supervised semantic type discovery approaches [43, 100] and overlapping community detection algorithms [20, 63, 94, 96]. Finally, the conference version [56] introduced only a single downstream task, domain discovery. In this extended version, we now consider a state-of-the-art deep learning entity matching algorithm as a downstream task, DITTO [59]. Importantly, we have

[2]Though disambiguation has been considered within natural language interfaces to SQL where the goal is to disambiguate query terms, it is not to disambiguate values within a large collection of tables.

also extended the three open source benchmarks for homograph detection described in Section 5 to include the number of meanings of a homograph—a number that can be varied to test the robustness of an approach.

The remainder of this article is organized as follows: In Section 2, we discuss existing work on disambiguation. In Section 3, we introduce our approach and describe how applying centrality measures on a graph representation of the data lake can be used to identify homographs. In Section 4, we expand upon our data lake disambiguation problem to allow us to identify the number of meanings of a homograph as well as group its attributes by their meaning. Section 5 summarizes the datasets used in our experimental evaluation and discussion presented in Section 6. We conclude and outline possible future directions of our work in Section 7.

2 FOUNDATIONS OF DISAMBIGUATION

Disambiguation has been studied in several contexts in NLP, data management, and broadly in AI and data science. We analyze how this work can be applied to disambiguation in large table repositories.

2.1 Entity Resolution

Entity Resolution (ER) identifies records (also called tuples) across different datasets (or sometimes corpora) that represent the same real-world entities. ER is generally applied to structured and semi-structured data including tables and RDF triples [37]. Some ER approaches also identify ambiguous values as part of the resolution process. For example, using collective entity resolution over two types of tables (e.g., papers and authors) one can identify if a value, say, "X. Wang," refers to different authors [9]. Similarly in familial networks, one can resolve synonyms (different values that refer to the same person) and identify homographs (same value used to refer to different people) [52].

ER assumes that the information to be resolved or disambiguated is of a single known type (e.g., resolving customer tuples or patient records) or from a small set of types (e.g., authors, their papers, and publishing venues). Some work, called schema-agnostic ER, does not require that all data be represented using the same schema [21]. However, all these approaches start with the assumption that two or more tables (or corpora) are describing the same type of entities [73, 74, 83].

In data lake disambiguation, we seek to find ambiguous values even when we do not know what type of entities a table is describing. We also do not know if different tables are describing the same or different entities. Hence, we cannot apply collective models or other resolution models that rely on this knowledge.

Example 2.1. Given the four tuples with `Jaguar`: `[BMW, Jaguar, 0.9M]`, `[Jaguar, San Diego, 8]`, `[XE, Jaguar, UK]`, and `[Jaguar, 25.8, 43224]`, does `Jaguar` have the same meaning? These four tuples correspond to four different types of facts: donors and the amount they contribute to protect an endangered species, animals in zoos, car models, and economic information about companies. ER schema-agnostic algorithms are insufficient for resolving (or disambiguating) values within these heterogeneous tables, because they rely on the hypothesis that the tables they examine refer to the same type of real-world entity.

2.2 Semantic Type Detection

One possible approach to data lake disambiguation is to discover semantic types for all attributes (columns) and then label a value appearing in different semantic types a homograph. In the running example, identifying the semantic type of `T1.At_Risk` and `T2.name` as animal and mammal, respectively, and knowing that mammals are animals, one can infer that Jaguar is not a homograph there. In contrast, recognizing T3.C2 is of type "Car Manufacturer," which is neither a sub- nor super-type of animals, implies that Jaguar in T3 and T1 represents a homograph. Here, we discuss different approaches to semantic type discovery and to what extent they could be used for homograph detection.

Knowledge-based Techniques. There has been considerable work on semantic type detection in the Semantic Web community that uses external knowledge from well-known ontologies including DBpedia [54], Yago [88], and Freebase [12]. Most solutions have been applied to Web tables [29, 30, 55] that are small (in comparison to tables in other data lakes [67]) and have rich metadata (table and attribute names).

Hassanzadeh et al. [39] use a map-reduce approach to find similarity between a (column, data value) pair from a table with a (class, instance label) pair from the **Knowledge Base (KB)**. Ritze et al. [79] match Web tables to DBpedia to profile the potential of Web tables for augmenting knowledge bases with missing information. These approaches cannot infer type information for an attribute that it is not part of the KB. Unfortunately, the coverage of values from data lakes in Open KBs is low (a recent study reports about 13% [68]), limiting their applicability.

Supervised Techniques. An alternative approach is to use **machine learning (ML)** to infer the semantic type of attributes. ML solutions utilize a variety of graphical models (Conditional Random Fields [38], Markov Random Fields [60]) as well as Multi-level Classification [89] and Deep Learning [43]. Sherlock [43] uses features about the values in an attribute to classify some of the attributes in a data lake into one of 78 semantic types (such as address or horse jockey) [43]. A more recent solution, called SATO [100], augments this approach and shows that using row information can improve the classification accuracy for the same 78 semantic types. These approaches require large amounts of labeled training data and are limited to only a fixed pre-defined set of types. Nonetheless, we consider them as baseline approaches for identifying homographs in our experiments. Any value that appears in more than one semantic type is considered a homograph.

Unsupervised Techniques. Unsupervised semantic type discovery algorithms have only recently started to be studied. We discuss two unsupervised algorithms, one for semantic type discovery, D^4 [71], and one for table unionability search [68].

D^4 provides an unsupervised approach with a focus on assembling all the values of each semantic type in a data lake [71] (these values are called a "domain"). They propose a data-driven approach that leverages value co-occurrence information to cluster values that are from the same domain. Heuristics attempt to deal with ambiguous values that may appear in multiple domains. In our context, D^4 can be used to label values that appear in multiple domains as homographs. This indeed serves as a baseline method for detecting homographs in our experiments. However, as we later show in Section 6.9, D^4's performance can still be impacted negatively by the presence of homographs, suggesting that DomainNet can be used as a pre-processing step to improve D^4's performance.

Table Union Search (TUS) [68] solves a different problem. Given a query table, they find a set of tables from the lake that are most unionable with it. To do so, they provide several similarity measures that are used collectively to calculate how unionable two attributes are. This work can use both ontological and semantic (word embedding) signals when present to determine unionability over heterogeneous attributes. TUS does not attempt to find/label homographs, but could be adapted for that problem by identifying shared values between a pair of non-unionable columns. However, such an approach would not be scalable, since all pairs of columns would have to be compared and labeled as unionable or non-unionable. One of our benchmarks has over 9,000 columns. Real table repositories reported in other studies have over 65,000 columns [104].

2.3 Disambiguation in Related Areas

Word-sense disambiguation (WSD) [44, 69], i.e., the task of identifying which meaning of a word is used in a sentence, is an important problem in computational linguistics. Although a human can proficiently perform this task on a document, constructing algorithms that perform this task effectively is still an open research problem. Techniques proposed so far range from dictionary-based methods, which use the knowledge encoded in lexical resources (e.g., WordNet) [69], to more recent solutions in which a classifier is trained for each distinct word on a corpus of manually sense-annotated examples [75]. Additionally, completely unsupervised methods have been proposed that cluster occurrences of words, thereby inducing word senses, i.e., word embeddings [44]. The aforementioned solutions rely on information (or latent information) about the structure of sentences including grammatical rules. Finally, while solutions that do not rely on grammar also exist, they only operate on documents and not tables [10, 84].

Another relevant sub-task in Natural Language Processing is **Named-Entity Recognition (NER)**, which has been proposed as a possible solution for disambiguation [93]. NER seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, and so on. NER systems have been created that use linguistic grammar-based techniques as well as statistical models [3]. A special case of the NER problem is the author name disambiguation problem [33, 85]. Authors of scholarly documents often share names, which makes it hard to distinguish each author's work. Hence, author name disambiguation aims to find all publications that belong to a given author and distinguish them from publications of other authors who share the same name. Different solutions have been proposed using graphs [57]. However, the graph structure proposed is largely domain-specific. The graph contains not only the information about the co-authorship and published papers, but also venue of the paper published, year of research activities, and so on. Overall, NER approaches are effective when they operate over specific domains and structured text, but they cannot be used directly over a large set of heterogeneous tables to identify homographs (especially if the homographs are null equivalent values or due to value misplacement). Also note that values in tables need not be named entities. The value `ca` may refer to the named entity `California`, but it may also be an abbreviation for the word `circa` or an initialism for the adjective `closed-access`. Similarly, a number like `10` may be a homograph referring to both a rating and to a street number.

Disambiguation of values also plays an important role in **entity matching (EM)** techniques, i.e., the problem of determining whether two data entries refer to the same real-world entity. For example, when comparing a pair of entries for products from two different tables, it is important to discern if the shared values and their respective contexts with other available attributes (e.g., price, product description, product code) can result in a match (same product) or not. With the emergence of deep learning techniques in NLP using modern transformer architectures (e.g., BERT [26], XLNet [95], RoBERTa [61], and DistilBERT [81]), EM techniques [14, 59] using them have seen their performance significantly improved, especially in benchmarks that are highly textual. One such state-of-the-art EM technique is DITTO [59], which uses fine-tuned pre-trained transformer-based language models and is further optimized by injecting domain knowledge, text summarization, and data augmentation. However, EM techniques are not suitable for discovering all homographs in a data lake, since two rows from different tables that contain the same value (with the same meaning) may not necessarily match, given the table context. For example, consider the two tuples A = `[UK, Jaguar, 25.80]` and B = `[Bob, Smith, Jaguar, XE, London, 45 Sheffield St.]`. Tuple A describes car brands and their revenue, and tuple B describes car owners and their address. Even though `Jaguar` refers to the car brand in both cases, the two tuples would not match using EM techniques, since they communicate different information. Moreover, EM techniques are focused on identifying matching and non-matching data entries from two tables (or a small number of tables) and not from a large and heterogeneous table repository like a data lake.

Natural language interfaces (NLI) for databases take advantage of user interaction to resolve ambiguities. NLIs are primarily designed for non-expert SQL users to facilitate querying by expressing queries in natural language [2]. Over the past few decades a large variety of NLI systems have been developed, such as keyword-based systems [11, 47, 82], pattern-based systems [25, 99], parsing-based systems [48, 58], grammar-based systems [1, 28, 86], as well as more recently machine learning approaches [15, 23, 90, 102]. Systems using different approaches are able to handle different levels of SQL complexities (e.g., keyword-based systems cannot handle aggregations or long-range dependencies, whereas parsing-based and grammar-based systems can). More importantly ambiguities (either from the specified values or from the query logic) are resolved mostly interactively by asking the user to confirm the values and/or logic understood by the system or by having the user choose from a set of options. Many NLIs are designed to operate over databases from a specific domain or using a specific type of queries and would not generalize very well to arbitrary databases and SQL queries. Although recent machine learning-based NLI approaches are more generalizable and designed without a specific domain in mind, such methods are usually evaluated over Spider [98], a large benchmark consisting of 10,181 questions and 5,693 SQL queries from 200 databases covering 138 domains. Despite the size of Spider, it is still far from the heterogeneity and size of data lakes that can contain thousands of domains and millions of unique values [67].

Another area of related work that can potentially be re-purposed to disambiguate values belonging in multiple communities is overlapping community detection [5, 24, 45, 92]. Overlapping community detection algorithms exploit the graph structure to identify nodes that can possibly belong to multiple communities (i.e., they overlap with two or more communities) [5, 92]. There is a large variety of techniques employed by overlapping community detection algorithms, such as clique expansion, link clustering, label propagation, and many others. We discuss a few of those methods that we later compare against `DomainNet`. Big-Clam [94] is a scalable overlapping community detection algorithm that combines non-negative matrix factorization methods with block stochastic gradient descent to identify node cluster affiliations. LPANNI [63] detects overlapping community structures by adopting a fixed label propagation sequence that is based on the ascending order of node importance and a label update strategy that is based on neighbor node influence. DANMF [96] is a deep autoencoder-like non-negative matrix factorization method that can learn hierarchical mappings between the original network and the final community assignment. It has implicit low-to-high level hidden attributes of the original network learned in the intermediate layers. Core Expansion [20] is a recent approach that tries to detect communities without computing the modularity score. It automatically detects the core of each possible community in the network and then iteratively expands each core by adding nodes to form the final communities. The expansion process is based on a neighborhood overlap measure. Overlapping community detection algorithms can be adapted to solve for the homograph discovery problem by labeling nodes that belong in more than one community as homographs. We compare the efficacy of overlapping community detection algorithms against `DomainNet` in Section 6.8.

3 DISAMBIGUATION AND HOMOGRAPH MEANINGS GROUPING USING DOMAINNET

We now present our proposed solution, `DomainNet`,[3] for finding homographs in a data lake, their number of meanings, and a grouping of attributes by their meaning.

[3]The code for `DomainNet` and our benchmarks is available at https://github.com/northeastern-datalab/domain_net

	Fiat	Toyota	Apple	Puma	Jaguar	Pelican	Panda	Lemur
T2.name					1		1	1
T1.At Risk				1	1	1	1	
T4.Name		1	1	1	1			
T3.C2	1	1			1			

Fig. 2. Example 3.1: Incidence matrix: The vertical axis shows attributes, the horizontal axis shows data values.

3.1 Problem Definition

In data lakes, attribute and table names can be missing or misleading (with many ambiguous terms such as “name,” “column 2,” or “detail”) [67]. Well-curated enterprise lakes may have more complete metadata, but even they do not follow the unique name assumption—which states that different attribute names always refer to different things. As a result, many data lake search approaches rely solely on the table contents [27, 32, 104, and others]. In a similar vein, in `DomainNet`, we investigate to what extent data values and the co-occurrence of data values within attributes can be used to determine if a value is a homograph, and if so, how many meanings it has in the data lake.

Definition 1 (Data Lake Disambiguation). Given a collection of tables with possibly missing, incomplete, or heterogeneous table and attribute names. Let t be a tuple, A an attribute, and $\Pi_A(t) = v$ be an attribute value in any table. For any attribute value v that appears in more than one attribute (column) or table, determine if v has a single meaning or more than one meaning. Values that only have one meaning are called *unambiguous values,* and values with more than one meaning are *homographs.*

Example 3.1. In Figure 1, the data value `Jaguar` is a homograph, because it is part of the animal domain in Tables $T1$ and $T2$, and in Tables $T3$ and $T4$ it is part of the car manufacturers domain. Other values such as `Panda` and `Toyota` are unambiguous, since they only have a single meaning across *all* tables. `Puma` is also a homograph, appearing as an animal and a company. Figure 2 displays which values co-occur with `Jaguar` in the same column using an incidence matrix: The vertical axis shows the different values, and the horizontal axis the different attributes occurring in the data lake.

Note that homographs need not be values from a dictionary. They can be any data value that appears in a table. Another example of a homograph is the data value `01223`, which in some attributes may refer to a Massachusetts zip code and in others to an area code near Cambridge, UK, and in yet others to the suffix of an Oil Filter Element Replacement product code.

In a well-curated database or warehouse, we may know the domain of each attribute (e.g., “Animal Name” vs. “Company Name”) and can leverage it to identify homographs. However, in large non-curated table repositories with possibly missing and ambiguous table and attribute names, we cannot rely on such information to be readily available.

3.2 `DomainNet`: Viewing Values as a Network

In data lakes, without *a priori* knowledge of table semantics or types, we take a network-based approach to understanding the meaning of repeated data values. We propose to detect homographs using network measures. For that purpose, we can interpret the co-occurrence information about values across different attributes using a network representation in which nodes represent data values and edges represent the fact that two values co-occur in at least one column (attribute) in

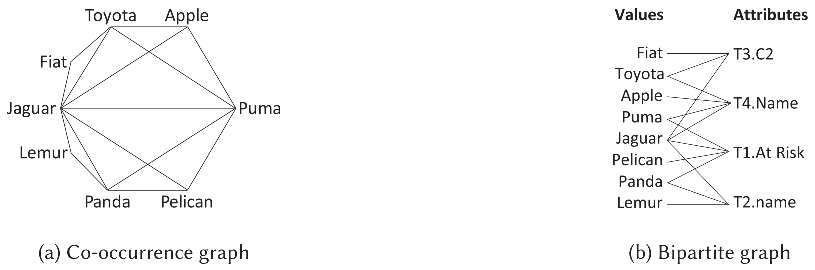


Fig. 3. Example 3.3: Two graph representations of a portion of Figure 1.

the data lake. Notice that while we do not use the column names, we do use the table structure (the fact that a value appears in a certain set of columns and co-occurs with other specific values). So, we are not treating tables as documents (i.e., bags-of-words based on their cell values), as usually done in information retrieval tasks.

Example 3.2. In Figure 3, we depict the values from the same four attributes shown in Example 3.1. Figure 3(a) shows the value co-occurrence network. Notice that by removing both "Puma" and "Jaguar," the remaining nodes become *disconnected* into two components. This captures the intuition that those two values are pivotal in that they bridge two otherwise disconnected meanings or graph components.

Whereas this representation allows us to apply straightforward metrics from community detection, it comes at a high cost: The representation uses more space than the original data lake. The incidence matrix is sparse and has as many entries, as there are cells in the data lake (Figure 2). In contrast, the co-occurrence graph increases quadratically in size with respect to the cardinality of attributes (the size of the vocabulary) in the data lake (Figure 3(a)). Consider a single column with 100 values. The incidence matrix represents this information with 1 row, 100 columns, and 100 entries. The co-occurrence graph represents this with 100*99/2 = 4,950 edges across 100 nodes.

Thus, we use a more compact network representation that allows us (after some modifications) to apply network metrics to discover pivotal points (Figure 3(b)). `DomainNet` uses a bipartite graph composed of (data) value nodes and attribute nodes. The attribute nodes represent the set of attributes and the value nodes the set of data values across all attributes in the lake. Every data value is treated as a single string; it is capitalized and has its leading and trailing white-space removed to ensure consistent comparison of data values across the lake. Notice that each data value, even if found in multiple attributes, is represented by one single value node in the graph. An edge is placed between a value node and an attribute node if the data value appears in the attribute (column) corresponding to that attribute node. Data values that appear in more than one attribute are candidates for being homographs. Notice that our bipartite graph representation is conceptually similar to an inverted index, as it allows us to quickly identify co-occurring nodes while saving a lot of space. Inverted indices have been widely used in information retrieval for efficient indexing as well as to encode term co-occurrences [34]. However, because our bipartite representation is still a graph, we can apply graph measures as well as graph compression techniques, as we later discuss in Section 3.4, to improve the efficiency of our method in identifying likely homographs.

Example 3.3. Figure 3(b) shows a portion of the `DomainNet` representation for Figure 1 using only the four attributes of Example 3.1.

In the `DomainNet` bipartite graph, we call two data values *neighbors* if they both appear in the same attribute (and hence there is a path of length two between them in the graph). Similarly, two attributes are *neighbors* if they have at least one data value in common (and hence there is a path of length two between them). For a data value node v, $N(v)$ denotes the set of all its value neighbors. We also define the *cardinality of a data value* node v as the number of neighbors $|N(v)|$, which is the number of unique data values that co-occur with v. If n is the number of value nodes and a the number of attribute nodes, then the number of edges in a DomainNet graph over real data tends to be much less than $n \cdot a$.

Tables to Graph. Recent work on embedding algorithms in relational databases [6, 16, 53] use a graph representation of tables. Like DomainNet, they model values and columns as nodes. Depending on the problem addressed, some approaches also include nodes for rows and tables. Like in our approach, column names are not assumed to be present or unambiguous.

Koutras et al. [53] and Capuzzo et al. [16] use a tripartite graph representation in which every value node is connected with its column node and its row node. Such an approach works well for the tasks of tuple-level entity resolution and for schema matching (a similar task to semantic type discovery). In our example, Panda in $T1$ and $T2$ are not homographs, but the row information makes them seem quite different.

In contrast, Arora and Bedathur [6] use a homogeneous graph using only data value nodes that are connected with each other if they appear in the same row of the table. They do not use the value co-occurrence information within a column, making homograph detection using solely row context inappropriate in large heterogeneous datasets.

3.3 Homograph-disambiguation Methodology

Intuitively, data values that frequently co-occur with each other will form an implicit domain or community in DomainNet, with many paths of varying length between them. Homographs will span two or more communities. Notice, however, that we do not know *a priori* what the communities are or even how many there are. While there is a rich literature on community detection, many approaches require knowledge of the possible communities such as the number of communities [18]. Others are parameter-free, meaning they can learn the number of communities [40, and others]. However, in our problem the number is not only unknown, it may be massive. A data lake from open data repositories with just a modest number of tables may have hundreds or thousands of attributes representing possibly different domains. Miller [66] states that in their study of open data, tables have, on average, 16 attributes but with a large variance, stating that some tables have many hundreds of attributes. Nevertheless, work in overlapping community detection could be utilized to identify homographs as cell value nodes that are part of multiple communities. We indeed make this comparison in our experiments (see Section 6.8).

As an alternative, what we propose in this article is to use network centrality measures that can be defined without prior knowledge of how many communities exist, their overlap, or the distribution of attribute cardinalities. The intuition behind centrality measures is to capture how well connected the neighbors of a given node are. We define variants of these measures appropriate for the DomainNet bipartite graph. We then discuss to what extent these measures may distinguish whether a data value has a single meaning or multiple meanings (the latter being a homograph).

Local Clustering Coefficient as a homograph score. The **local clustering coefficient (LCC)** [91] for a given value node measures the average probability that a pair of the node's neighbors are also neighbors with each other, i.e., the fraction of value-neighbor triangles that actually exist over all possible triangles.

The LCC metric is usually defined over unipartite graphs (such as the co-occurrence graph in Figure 3(a)). We use the definition of value-neighbors (recall the set of all value neighbors of a value node u is $N(u)$) to generalize LCC to our bipartite graph.

The pairwise clustering coefficient of two data value nodes v and w is defined as the Jaccard similarity between their neighbors.

$$c_{vw} = \frac{N(v) \cap N(w)}{N(v) \cup N(w)}.$$

Given a graph G and a value node u, the LCC is defined as the average pairwise clustering coefficient among all the node's value neighbors:

$$c_u = \frac{\sum_{v \in N(u)} c_{vu}}{|N(u)|}. \tag{1}$$

The LCC of a node u can be computed in time $O(N(u)^2)$ and provides a notion of the importance of a node in connecting different communities.

Hypothesis 3.4 (Homographs using LCC). A value node corresponding to a value that is a homograph will have a lower local clustering coefficient than a value node with a single meaning.

Intuitively, we expect unambiguous values to appear with a set of values that co-occur often and thus have high LCC scores. This behavior should be less common for homographs, which may span values from different communities, as they appear in various contexts depending on their meaning.

Despite LCC's computational simplicity, the measure as defined in Equation (1) is no more than the average Jaccard similarity between the set of attributes that a value co-occurs with. Unfortunately, it is well-known that *Jaccard similarity is biased to small sets.* As a consequence, *the measure is not as effective in real data lakes,* where attribute sizes are often considerably skewed. Our experiments will confirm this downside of LCC.

Betweenness Centrality as a homograph score. The LCC of a node is fast to compute, but it only considers the local neighborhood of a value. In a data lake, the local neighborhood may not be sufficient. In particular, the local neighborhood may not include values that are members of the same community but happen to not co-occur. To overcome these two problems (missing values in the neighborhood and attributes with very different cardinalities), we look at metrics that take a more global perspective on the network.

The ***betweenness centrality*** **(BC)** of a node measures how often a node lies on paths between *all other nodes* (not just the neighbors) in the graph [35]. One way to think of this measure is in a communication network setting where the nodes with highest betweenness are also the ones whose removal from the network will most disrupt communications between other nodes in the sense that they lie on the largest number of paths [70].

Consider two nodes v and w. Let σ_{vw} be the total number of shortest paths between v to w, and let $\sigma_{vw}(u)$ be the number of shortest paths between v to w that pass through u (where u can be any node).[4] The betweenness centrality of a node u is defined as follows, where v and w can be any node in the graph:

$$BC(u) = \sum_{v \neq u, w \neq u} \frac{\sigma_{vw}(u)}{\sigma_{vw}}. \tag{2}$$

By convention, $\frac{\sigma_{vw}(u)}{\sigma_{vw}} = 0$ if σ_{vw} (and therefore $\sigma_{vw}(u)$) is 0.

Intuitively, a homograph appears with sets of values that do not or rarely co-occur across those sets, and thus the shortest paths between such non-co-occurring nodes would have to go through

[4]Since the bipartite graph used in `DomainNet` is not homogeneous, we also examined other variations of BC, such as considering only values nodes as end points for the examined shortest paths. We found that using all nodes in the BC definition provided empirically the best results for finding homographs.

the homograph node. Conversely, unambiguous values appear with a set of values that also co-occur a lot, and thus the shortest path between them does not unnecessarily have to go through one or a few nodes.

Hypothesis 3.5 (Homographs using BC). A value node corresponding to a homograph will have a higher betweenness centrality than a value node with a single meaning.

Example 3.6. The LCC scores of the `Jaguar` and `Puma` data value nodes in Figure 1 are 0.36 and 0.43, respectively. The LCC scores of the other data value nodes that appear more than once, `Toyota` and `Panda`, are somewhat higher at 0.46. The BC scores of the `Jaguar` and `Puma` value nodes in Figure 1 are 0.025, 0.003, respectively. The BC of the other value nodes that appear more than once, `Toyota` and `Panda`, are at 0.002. Since this example only uses four small tables, it does not expose the possibly different rankings between LCC and BC scores but suggests that BC, even on small graphs, is more discerning.

Complexity of BC. Calculating the BC for all nodes in a graph is an expensive computation. A naive implementation takes $O(n^3)$ time and $O(n^2)$ space (n denotes the number of nodes in the graph). The most efficient algorithm to date is Brandes' algorithm [13], which takes $O(nm)$ time and $O(n + m)$ space (for unweighted networks), where m is the number of edges in the graph. Notice that this algorithm is still expensive if the graph is dense (i.e., $m >> n$).

The high time complexity of BC motivated approximations, which usually sample a subset of nodes from the graph and thus do not calculate all shortest paths. One common sampling strategy is to pick nodes with a probability that is proportional to their degree (nodes with high degree are more likely to appear in shortest paths). Riondato and Kornaropoulos [78] provide an approximation algorithm via sampling with offset guarantees. Geisberger, Sanders, and Schultes [36] provide an approximation algorithm without guarantees that performs very well in practice. The complexity of the approximate BC is $O(sm)$, where s is the number of nodes sampled. We chose Geisberger, Sanders, and Schultes [36] to approximate betweenness centrality to benefit most from its short runtime on large graphs.

3.4 `DomainNet` Graph Compression

Although our bipartite `DomainNet` graph is designed to be sparse as described in Section 3.2, for the purposes the BC computation there is topological information in the graph that we can leverage to further speedup the computation. More specifically there are many cell nodes in the bipartite `DomainNet` graph that are connected to the same set of attribute nodes. For example, the value "Lion" may always appear in attributes where the value "Tiger" also appears, so the cell nodes corresponding to values "Lion" and "Tiger" will be connected to the same set of attribute nodes. In the context of BC this translates to "Lion" and "Tiger" having the same BC scores, because for any pair of nodes where there is a shortest path between them through cell node "Lion," there is another path with the same length that goes through the cell node "Tiger." Nodes such as "Lion" and "Tiger" that always appear in the same set of attributes are quite common in large data lakes, so if we can group them together and reduce the graph size, then the computation of BC can be significantly improved.

Graph summarization techniques such as graph compression, grouping, and simplification have been extensively studied [8, 62, 65] and applied in various applications, such as query handling [64, 77], pattern discovery [22, 101], graph databases [46, 50], or for semi-supervised learning from few labels [72]. The idea of compressing the graph by identifying sets of nodes that are solely connected with another set of nodes for the purpose of speeding up the exact BC computation was first explored by Sariyüce et al. [17], where they propose various graph shattering[5] and compression techniques that reduce the size of the input graph. Let $\Gamma(v)$ denote the set of nodes connected to node v. We say two nodes u, v are *identical* if and only if $\Gamma(v) = \Gamma(u)$. Given a graph G, Sariyüce et al. [17] generate a compressed graph G' by compressing each set of identical values into a single *compressed* node. They then can run Brandes' BC algorithm [13] on G' with a few small modifications and obtain the BC scores for all nodes in G. In short, this is achieved by recording the number of nodes each compressed node (called the "ident" score [17]) in G' corresponds to in G. The computation speedup comes from the fact that the compressed graph G' will have fewer nodes and edges than the original graph G. Since Brandes' algorithm has a complexity of $O(nm)$, the speedup can be significant if the compression is large. For instance, if the number of nodes and edges is halved in the compressed graph, then we would expect a quadruple speedup.

Although Sariyüce et al. [17] use their graph compression techniques to compute the exact BC scores for each node in a graph, we can still re-purpose them to compute the approximate BC scores for each node if we want an even faster answer. As with many other BC approximation algorithms [7, 19, 36, 78], we can sample a subset of the nodes in the compressed graph G' and compute all the shortest paths that start from our sampled nodes.

One way to sample for nodes in the compressed graphs is using uniform random sampling, but, as we will demonstrate in Section 6.6, that can lead to a biased sampling. This is because many compressed nodes correspond to multiple nodes in the original graph, but they are sampled with the same probability as uncompressed nodes. This can lead to large inaccuracies in the approximation, since compressed nodes that correspond to many nodes in the original graph can have a large impact in the BC scores if not sampled adequately. Therefore, to accommodate for this imbalance, we can use weighted random sampling over the compressed graph where the probability of sampling a node is proportional to its *ident* score.[6]

3.5 DomainNet Overview

In this section, we describe the implementation of an end-to-end system that allows users to disambiguate data lakes using our proposed methodology. Our system has three steps, as illustrated in Figure 4: (1) construct DomainNet graph; (2) calculate measures; and (3) rank measures.

DomainNet *graph construction.* The input is a set of raw data tables from relational databases, CSV files, or any other open data format. It is important to note that we do not require any information in regards to types, attribute names, or the semantics of relationships between tables. We build our bipartite graph as described in Section 3.2.

Graph measure computation. Using the DomainNet graph constructed in the previous step, our system computes both LCC and BC scores for each value node (Section 3.3) or compressed value node (Section 3.4). We show empirically in Section 6.1 that BC outperforms LCC in homograph detection.

Graph measure ranking. Nodes are ranked by their centrality score (ascending order for LCC measures and descending order for BC measures) the top-ranked data values are presented to a user.

4 HOMOGRAPH MEANINGS GROUPING USING DOMAINNET

Having extracted a ranked list of possible homographs from a data lake as described in Section 3.5, we extend our approach to answer the following problem: *Which columns (attributes) that the*

[5]Graph shattering is a technique similar to graph decomposition, where the original graph is partitioned into a set of smaller subgraphs. The algorithm is then run on the subgraphs, which is faster, and the results from all the subgraphs are combined to get the final answer in the original graph.
[6]Uncompressed nodes are given an *ident* score of 1.

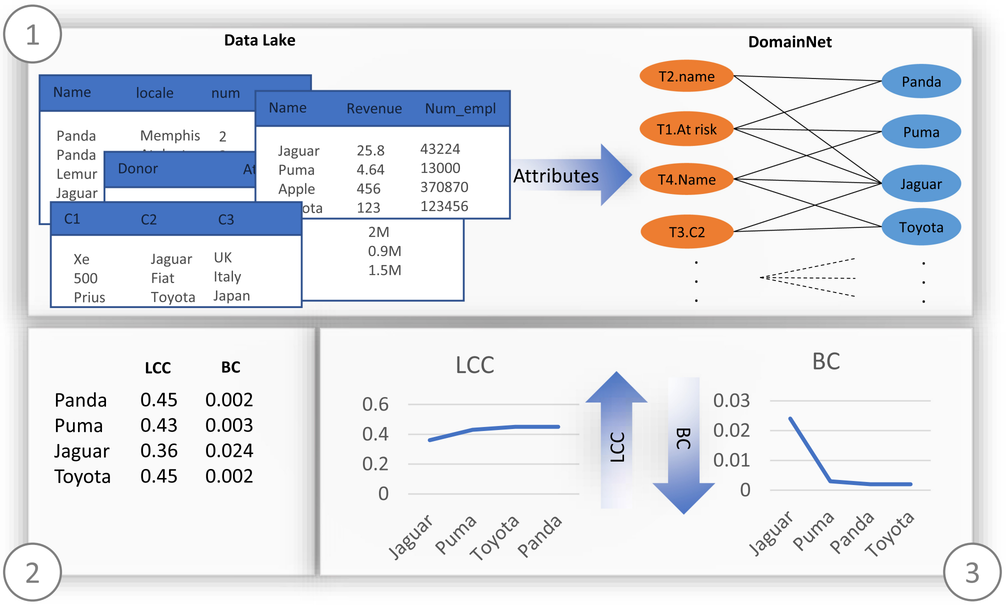


Fig. 4. Disambiguation system on DomainNet. (1) Construct a DomainNet graph from a data lake. (2) Calculate BC and LCC scores for each value node in the graph. (3) Rank value nodes by their scores (either increasing for LCC or decreasing for BC).

homograph appears in have the same meaning? Notice that we assume that the data value of interest is a homograph. More concretely, we define the problem as follows:

Problem 4.1 (Homograph Meanings Clustering). *Given a data lake and a homograph that appears in a set of attributes, group these attributes by their meaning.*

Answering this problem can provide useful knowledge to a data scientist especially in the context of data exploration and data cleaning. For instance, if a homograph has "too many," (e.g., more than 15) meanings, then it may have been used as a substitute for a null or missing value (something we will refer to as a *null value homograph*), as it is very unlikely for a "real" word or value to be found in some many different contexts. For example, solving Problem 4.1 given the homograph "Jaguar" and the tables in Figure 1 will group attributes T3.C1 and T4.Name together (i.e., the value "Jaguar" found in these two attributes has the same meaning, which is a name of a car manufacturer). Similarly, attributes T1.At_Risk and T2.name are grouped together (i.e., the value "Jaguar" in these two attributes has the meaning of the animal). With this information, a data scientist can improve the data quality of the tables, for example, by renaming (or annotating) the column headers to include disambiguating information. Notice that the grouping of attributes into meanings (domains) may also help identify data entry errors. In our running example Example 1.1, if the value "BMW" has two meanings, where all attributes $T3.C1$ share the same meaning (car manufacturer) and $T3.C1$ is in a singleton group, then this may indicate that the value "BMW" has been misplaced in this attribute.

4.1 Grouping Attributes of a Homograph by Their Meaning

In this section, we describe our method for solving Problem 4.1. Let h be the homograph in question and let $Attr(h)$ denote the set of attributes (columns) the homograph h appears in. Our goal is to

$T1$

country1	capital1
Cuba	Havana
Barbados	Bridgetown
⋮	⋮

$T2$

country2	capital2
Cuba	Havana
Bahamas	Nassau
⋮	⋮

$T3$

title1	genre1
Cuba	Adventure
Cobra	Action
⋮	⋮

$T4$

title2	genre2
Cuba	Adventure
The Sixth Sense	Drama
⋮	⋮

Fig. 5. Example 4.2: Four tables where the value "Cuba" is a homograph, as it appears as a country in tables $T1$ and $T2$ and as a movie title in tables $T3$ and $T4$. Note that the vertical dots indicate that there are more rows in the table that have been omitted for brevity.

find $\mathcal{N}'$ sets (or groups) of attributes, $G_1, \ldots, G_{\mathcal{N}'}$, where $2 \le \mathcal{N}' \le |Attr(h)|$ such that $\bigcup_{i=1}^{\mathcal{N}'} G_i = Attr(h)$ and $\bigcap_{i=1}^{\mathcal{N}'} G_i = \emptyset$. The sets of attributes $G_1, \ldots, G_{\mathcal{N}'}$ correctly solve Problem 4.1 if and only if all the instances of the homograph h in each attribute group have the same meaning. So, if in the ground truth there are $\mathcal{N}$ meanings for homograph h and our solution is correct, then $\mathcal{N}' = \mathcal{N}$ and the sets of attributes $G_1, \ldots, G_{\mathcal{N}'}$ will correspond to the correct clustering of h's attributes based on their meaning.

One way to group the set of attributes of a homograph by their meaning is to find a measure for the similarity of the values between two attributes. Popular similarity measures in this context include Jaccard similarity [32] and containment [103]. More sophisticated measures such as unionability [68] that can overcome value heterogeneity (e.g., two columns whose values do not overlap but still belong to the same domain) have also been proposed.

We choose the Jaccard similarity, because it is inexpensive to compute and performed well in our experiments. More specifically, given a homograph h, we compute a distance matrix of size $|Attr(h)| \times |Attr(h)|$ by computing the pairwise Jaccard distances[7] between the cell values contained in each pair of attributes. We can think of this process as an assignment of each attribute to a point in $\mathbb{R}^{|Attr(h)|}$ where ideally attributes that belong to the same domain will be spatially close to each other. Therefore, given the distance matrix, we would like to cluster its rows so each cluster corresponds to attributes where instances of homograph h in them have the same meaning. There are a myriad of clustering algorithms, but given that we want to identify spatially close points (i.e., in our case, attributes) a density-based clustering algorithm would be fitting. We selected DBSCAN [31], because it is efficient and it can automatically identify the number of clusters, unlike k-means and many other clustering algorithms. DBSCAN identifies a point in a dense region as an initial cluster and then expands this cluster by including neighboring points that are "sufficiently close." Points in low-density regions that are not assigned to a cluster are labeled as outliers.

There are two key parameters in the DBSCAN algorithm: Epsilon (ε) and Minimum Points (`minPts`).

— The parameter ε corresponds to the maximum distance between two points to be considered in the neighborhood of each other. Note that this is not a limit on the maximum distance between two points in a cluster, since the clusters are iteratively expanded. In general, a larger epsilon leads to fewer clusters. We demonstrate in Example 4.2 and Section 4.2 that this is the most important parameter to set, as it can have a large impact on the accuracy of the discovered clusters.

— The parameter `minPts` corresponds to the minimum number of points that are an ε distance away from each other to be considered a cluster. In our case, we set `minPts` to 1, as the minimum cluster size we want to obtain is 1 point (i.e., one attribute).

[7]The Jaccard distance is just one minus the Jaccard similarity.

Notice that our choice of Jaccard similarity is not in contradiction with our claims about it in Section 3.3. Generating the pairwise Jaccard distances across all pairs of attributes in the dataset and running DBSCAN on this larger matrix will not be effective. Jaccard similarity cannot meaningfully express the differences between attributes at a global scale; however, if applied locally for the attributes of a value we have identified as a homograph, then it can yield much more meaningful results.

Example 4.2. Consider the tables shown in Figure 5 where the value "Cuba" is a homograph, as it appears both as a country and as a movie title. These are snippets of actual tables with 200–1,000 rows taken from a benchmark we present in Section 5.1. Our goal is to solve Problem 4.1 and ideally group the attributes {`T1.country1, T2.country2`} in one cluster and {`T3.title1, T4.title2`} in another cluster. We start by building the pairwise Jaccard distances matrix between the 4 attributes, which comes out to:

$$M_{\text{Jaccard Distance}} = \begin{array}{r} \\ \text{country1} \\ \text{country2} \\ \text{title1} \\ \text{title2} \end{array} \begin{array}{c} \begin{array}{cccc} \text{country1} & \text{country2} & \text{title1} & \text{title2} \end{array} \\ \begin{pmatrix} 0 & 0.15 & 0.99 & 0.99 \\ 0.15 & 0 & 0.99 & 0.99 \\ 0.99 & 0.99 & 0 & 0.8 \\ 0.99 & 0.99 & 0.8 & 0 \end{pmatrix} \end{array}$$

Notice how the Jaccard distance between the two country attributes is relatively small at 0.15. The Jaccard distance between the two title attributes is fairly large at 0.8, but the distance between a country attribute and a title attribute is even higher at 0.99. Such a situation is quite common in practice, as two domains with the same semantic type can still have a small intersection (e.g., there are more unique movie titles than countries and thus the intersection between two movie title columns can still be relatively small). However, as long as the intersection across two semantically different domains is even smaller, we can still correctly perform clustering on our pairwise distances matrix.

In this example, to extract the right clusters using DBSCAN, we need to set ε to be $0.8 \leq \varepsilon < 0.99$. If $\varepsilon < 0.8$, then at least 3 clusters will be generated, as the two title attributes are at a distance greater than the assigned ε. Similarly, if $\varepsilon \geq 0.99$, then only 1 cluster will be generated, since all values are at a distance less than the assigned ε. Finding the right ε can be challenging, but a possible simple heuristic would be to remove all pairwise distances that are very large (e.g., anything above 0.95) and then set epsilon to the max of the remaining pairwise distances in the matrix. We explore this heuristic and propose an improvement in Section 4.2.

4.2 Choosing the ε Parameter for DBSCAN

To motivate our algorithm that chooses ε in practice, we first discuss an ideal scenario where the ground truth about the semantic type of each relevant attribute is given. Having the semantic type ground truth means that we also know the correct groups of attributes $G_1, \ldots, G_{N'}$ that solve Problem 4.1. So, using DBSCAN, how should we set ε to extract the same groups? Let D_{G_i} be the multiset of the pairwise distances between all attributes in G_i, so if our distance measure is Jaccard $D_{G_i} = \{J(a,b) | \forall a, b \in G_i\}$ where $J(a,b)$ denotes the Jaccard distance between attribute a and b. We choose $\varepsilon = \max(\max(D_{G_1}), \ldots, \max(D_{G_n}))$; in other words, ε is set as the largest pairwise distance between attributes of the same type. Running DBSCAN with the following ε and minPts = 1 will extract the correct groups of attributes as long as there is no pairwise distance between attributes of different types that is a distance less than or equal to ε. To see why, consider the pairwise Jaccard distances matrix shown in Example 4.2. If we are given the ground truth of the semantic types and choose our ε as $\max(\max(\{0.15, 0.15\}), \max(\{0.8, 0.8\})) = 0.8$, then when we run DBSCAN, we will correctly find groups {`T1.country1, T2.country2`} and {`T3.title1, T4.title2`}. However, assume now a case where J(`T2.country2, T3.title1`) $= 0.50$, then even when we are given the ground truth and choose $\varepsilon = 0.8$, DBSCAN will not be able to extract the correct clusters. This is because there will be a pairwise distance between two attributes with different types (specifically, between `T2.country2` and `T3.title1`) that is smaller than a pairwise distance between two attributes with the same type (specifically, between `T1.title1` and `T2.title2`). Therefore, there is no ε value for which DBSCAN can correctly group the attributes.

In practice, we will not have the ground truth of the semantic types for each attribute and so we cannot exactly choose ε as we discussed above. However, there is a simple heuristic, as we briefly mentioned in Example 4.2 to choose an ε. Notice that, ideally, our choice of epsilon should correspond to the largest pairwise distance between attributes of the same type, so, if we assume that distances between attributes of different types is typically very large (e.g., ≥ 0.95), then we can take all the pairwise distances from the pairwise distances matrix, filter out those that are very large, and pick epsilon as the maximum of the remaining pairwise distances. More formally:

HEURISTIC 4.3 (GREATEST ε). *Given a homograph h, a matrix M corresponding to the pairwise distances between the attributes that h appears in, and a distance threshold τ, choose $\varepsilon = \max(\{m_{ij} \mid m_{ij} < \tau\})$.*

As we shall see in our experiments in Section 6.4, Heuristic 4.3 performs quite well when we choose a simple threshold (e.g., $\tau = 0.99$), but we can come up with an even better way to choose ε that does not require us to specify a hard threshold. The main downside of Heuristic 4.3 is that even with a large threshold, τ, there are cases that it can filter out pairwise distances between attributes with the same type. For instance, homographs such as “Lincoln” that appear both as a first name but also as a city name suffer by the presence of an arbitrary threshold imposed by Heuristic 4.3. More specifically, the pairwise Jaccard distance between two city name columns can many times be larger than a specified threshold τ, since the domain of cities is much larger than that of first names in this benchmark. So, two columns that are both about cities may have a very small intersection. Nevertheless, even though the pairwise distance between attributes of the same type is very large, it most likely is not as large as the pairwise distance between attributes of different types, so there does exist an ε for which DBSCAN will identify the correct groups of attributes. We can find such an ε by estimating the density distribution of all the pairwise distances and choosing the left boundary of the last peak (the peak with the highest pairwise distance) in the density distribution as ε. We can estimate the density distribution of the pairwise distances using **Kernel Density Estimate (KDE)**, which fits a probability density distribution function over the input of the pairwise distances.

HEURISTIC 4.4 (KDE ε). *Given a homograph h, a matrix M corresponding to the pairwise distances between the attributes that h appears in, compute the KDE distribution using its pairwise distances and pick ε as the left boundary of its last peak (the peak with the highest pairwise distance).*

The reason we choose ε as the left boundary of the last peak is because we assume the last peak corresponds to the pairwise distances of attributes with different types, so if we choose an ε that corresponds to the pairwise distance just before the last peak (i.e., the left boundary of the last peak), then it should eliminate the clustering of attributes with different types together. We illustrate that process in Figure 6(b), which shows the density distribution found using KDE for the pair distances of all attributes that appear with the homograph “Lincoln.” Notice how close the last two peaks are. The last peak corresponds to the pairwise distances between attributes of different types, whereas the small second-to-last peak corresponds to the pairwise distances between city attributes where “Lincoln” appears in. The leftmost boundary of the last peak corresponds to a

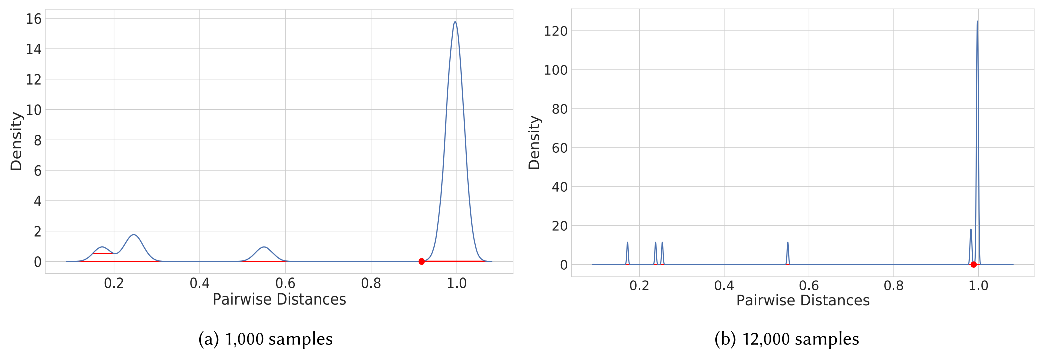


Fig. 6. Pairwise distance density distribution between the attributes of the homograph "Lincoln." The distribution in Figure 6(a) is generated by running KDE with 1,000 samples and in Figure 6(b) with 12,000 samples. The red dot in each figure indicates the chosen value for ε when applying Heuristic 4.4. The red horizontal lines indicate the width of each peak.

distance of about 0.988, so using Heuristic 4.4, we also set $\varepsilon = 0.988$ and running DBSCAN correctly groups all attributes found with "Lincoln," which in this case has three meanings, as it appears as a car brand, a first name, and as a city name.

An important parameter when running KDE over the pairwise distances is the number of samples (i.e., distinct distance positions on the x-axis in Figure 6) for which we estimate the density. In Figure 6(b), we have chosen 12,000 samples, whereas in Figure 6(a), we have chosen 1,000 samples. Notice that, because we did not estimate the density at enough points in Figure 6(a), the pairwise distances between the city attributes and pairs of attributes with different types are all represented in the last peak. Therefore, using Heuristic 4.4, we will extract an $\varepsilon = 0.918$, which is not optimal, since now the city attributes will no longer be grouped together. However, if we pick too many samples, then that is also a problem, as that can lead to overfitting to the pairwise distances and may create multiple peaks that correspond to the last peak in Figure 6(b), thus the selected ε would be too high and lead to too many groups. One way to handle this problem is to choose the number of samples to be dependent on the range of the non-zero pairwise distances. For instance, the pairwise distances in Figure 6(b) range from about 0.15 to 1, so a considerable number of samples is needed to detect the second-to-last peak separately from the last peak. Conversely, if the range is from 0.85 to 1, then we can separate the peaks with fewer samples and in fact choosing more samples will lead to overfitting. Empirically, we found a good way to dynamically assign the number of samples for KDE is to use the following formula:

$$\text{number of samples} = \frac{\text{baseline number of samples}}{1 - \log_2(\text{range})}, \tag{3}$$

where range corresponds to the difference between the largest pairwise distance and the smallest non-zero pairwise distance. Notice that when the range is equal to 1, then the number of samples is just equal to the baseline number of samples and as the range decreases, we choose fewer samples to reduce overfitting.

5 DATASET DESCRIPTION

Homograph detection in data lakes is a new problem and no benchmarks are available for it. While many data lakes exist, they do not contain labels that identify the homographs. In addition to being a hugely expensive task when done manually, homograph labeling is not a one-time effort: When

Table 1. Four Datasets and Their Statistics

	#Tables	#Attr	#Val	#Hom	Card(H)	#M
SB	13	39	17,633	55	151–1,966	2–3
TUS - I	1,253	5,020	163,860	0–5,000	0-500	2–8
TUS	1,327	9,859	190,399	26,035	3–22,703	2–100
NYC-EDU	201	3,496	1,469,547	N/A	N/A	N/A

the content of the data lake changes, an unambiguous value can become a homograph or vice versa. Hence, benchmark design in this context constitutes a non-trivial contribution in itself.

We introduce the four datasets used for the evaluation of DomainNet. The first is a new synthetic benchmark and the other three contain real data. The second is an adaptation of the TUS Benchmark [68] that uses real tables from UK and Canadian open-data portals. We adapt TUS for our problem. The third is a modified version of TUS, called TUS-I, where we systematically inject homographs. The fourth, used to evaluate scalability, is a real dataset from NYC Education Open Data, which was also used to evaluate a domain discovery approach [71].

Table 1 summarizes detailed statistics about the datasets. For each, we list the number of tables, the total number of attributes (columns) across all tables, the number of unique values in the data lake, the total number of homographs, the range of cardinalities of any homograph[8] (Card(H)), and the range of the number of distinct meanings, #M, (based on ground truth) the different homographs have across the data lake. All our datasets are publicly available.[9]

5.1 Synthetic Benchmark (SB)

We designed a small fully synthetic, but real-world inspired, data lake for a systematic validation of our approach. It consists of 13 tables generated using Mockaroo,[10] which lets the data creator specify data sources from various categories.

Each table has 1,000 rows, except for two tables that contain countries and states. We used the real numbers of countries and US states of 193 and 50, respectively. There are 55 data values that are homographs, e.g., Sydney (city or name), Jamaica (city or country), Lincoln (first-name, car, or city), CA (country or state abbreviation), and Pumpkin (grocery product or movie title).

5.2 Table Union Search Benchmark (TUS)

In the absence of homograph-labeled large real data lakes, we set out to find a closely related benchmark that we could adapt to our purposes. Unfortunately, while there are many table-based benchmarks, even some for data-semantics-related problems, they generally prove hard to adapt. For example, the VizNet corpus [41] used in semantic type detection in tables [43, 100] provided ground-truth labels for only a small fraction of the columns in the repository, making ground-truth discovery of all homograph labels practically impossible. We therefore selected the TUS benchmark [68], which contains real data and provides a ground-truth mapping for *each* column to the set of columns in the repository that it is unionable with. This enables us to automatically label all homographs. Let $U(a)$ denote the set of columns (attributes) a given column a is unionable with and notice that a is always unionable with itself, hence $a \in U(a)$. Let $Attr(v)$ be the set of columns (attributes) a data value v appears in. Converting the TUS benchmark into our bipartite

[8]Recall the definition of the cardinality of a homograph node v as $|N(v)|$, which is the number of unique data values that v co-occurs with.
[9]All our datasets can be found at: https://github.com/northeastern-datalab/DomainNet-Datasets
[10]https://www.mockaroo.com/

graph representation, we can automatically label data values as "unambiguous" or "homograph" based on the unionability ground truth.

Definition 2 (Homograph in the Table Union Search Benchmark). A data value v is a homograph if there exist two attributes a and a' in $Attr(v)$ such that $U(a) \neq U(a')$; otherwise, v is an unambiguous value.

Intuitively, a data value is a homograph if it appears in at least two different columns that are not unionable (and hence have different types). For instance, assume value "USA" appears in columns `X1.country` and `X2.location`. If the corresponding two columns are unionable, i.e., $U(\texttt{X1.country}) = U(\texttt{X2.location}) = \{\texttt{X1.country}, \texttt{X2.location}\}$, then we can conclude that "USA" is an unambiguous value. In contrast, the columns containing the value "Jaguar" in the zoo or donor tables from Figure 1 are not unionable with either the company or car model tables, and hence "Jaguar" would be labeled a homograph.

Based on Definition 2 there are 164,364 unambiguous values and 26,035 homographs in the TUS benchmark, suggesting homographs are very abundant in real data lakes. Notice that attribute cardinalities in TUS are highly skewed, a common phenomenon in data lakes for open-data repositories [67]. Hence, this benchmark provides a "stress-test" for our approach. How well can it deal with both small and large cardinalities of attributes containing a homograph (in TUS, these cardinalities range from 3 to 22,703)?

5.3 TUS with Injected Homographs (TUS-I)

Having real data is important, but we also need to understand the performance of our solution as the number of homographs in a data lake changes. To this end, we modified the TUS benchmark as follows: First, we removed all 26,035 homographs. Second, we carefully introduced artificial homographs with different properties. Since the artificial homographs are now the only ones in the data lake, we can measure how their properties affect our detection algorithm.

A homograph is injected by selecting two different data values from two columns that are not unionable. These original values are then replaced by a new unique value such as "InjectedHomograph1." We only replaced string values with at least three characters. In our experiments, we vary the minimum allowed cardinality of the attributes containing values replaced with an injected homograph. We also vary the number of meanings of an injected homograph. This allows us to evaluate the effectiveness of our approach in identifying homographs with respect to the cardinality and number of meanings of the homographs.

6 EXPERIMENTAL EVALUATION

The main goal of the experiments is to evaluate how well `DomainNet` performs in terms of precision and recall for identifying the homographs in the benchmark datasets. We are particularly interested in determining if the more expensive BC provides significant improvement over LCC (Section 3.3). Since a homograph candidate must appear in at least two different table columns, `DomainNet` pre-processes the input to remove data values that appear only once in the data lake. As a result, the corresponding graph representation has about 3% fewer nodes in the TUS benchmark and 30% fewer nodes in SB.

We begin with a detailed evaluation of `DomainNet`, comparing it to the state-of-the-art unsupervised domain discovery technique and also evaluating its performance on the benchmarks introduced in the last section. We consider the homograph detection problem along with the identification of the number of meanings. We also evaluate `DomainNet` scalability. We then compare `DomainNet` against supervised (deep learning-based) approaches for semantic type detection and community detection algorithms. Finally to illustrate the importance of the homograph detection

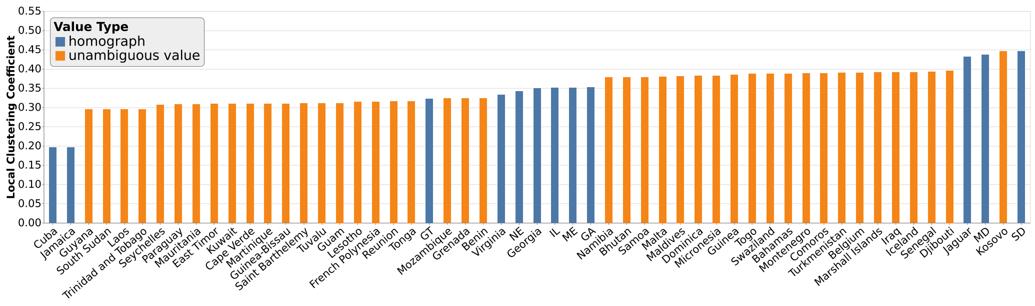


Fig. 7. The top-55 data values with the lowest local clustering coefficients. Homographs are scattered throughout and do not necessarily have low LCC coefficients.

problem, we evaluate the impact of homograph detection on two important data integration tasks: domain discovery and entity matching.

Comparison baselines. There is no previous work that directly explores homograph detection in data lakes (Section 2). Previous work on the related problem of semantic type detection and domain discovery is generally supervised, i.e., requires labeled training data. Hence, we begin with a comparison with the recently proposed state-of-art unsupervised domain-discovery algorithm D^4 [71]. We used the original code provided by the authors[11] with its default parameter settings. When applied to a data lake, D^4 assigns attributes to the discovered domains. A natural way to identify homographs then is to identify data values that appear in more than one of those domains. We compare D^4 to `DomainNet` on the synthetic benchmark, as it only contains string values. D^4 discovers domains only for string data, making it ineffective on the TUS benchmark, which contains real data with many numerical attributes. We also consider the deep-learning supervised semantic type detection algorithms Sherlock [43] and SATO [100] as baselines (Section 6.7), along with state-of-the-art community detection algorithms (Section 6.8). None perform well, as they are not designed for the disambiguation task.

Measures of success. We generally measure precision and recall, which are reported for the k top-ranked homograph candidates identified by each of the algorithms. By default, k is set to the true number of homographs in the data lake.

Software implementation. We implemented `DomainNet` in Python 3.8 using NetworKit[12] [87] to calculate exact and approximate BC scores over our bipartite graph. This is a Python library for large-scale graph analysis whose algorithms are written in C++ and support parallelism. All our experiments were run on a commodity laptop with 16 GB RAM and an Intel i7-8650U CPU.

6.1 Fully Synthetic Benchmark (SB)

We first use the SB to compare the homograph rankings obtained using the LCC and BC measures (Section 3) to study their ability to identify homographs. The bipartite graph for SB is relatively small, consisting of 17,672 nodes (17,633 data-value nodes and 39 attribute nodes) and 19,473 edges. We calculated the LCC and BC for each node in the graph and examined how these scores differ between homographs and unambiguous values.

Which measure is better at discovering homographs? Figure 7 shows the top-55 data values based on LCC. For LCC, lower scores should in theory indicate a greater likelihood of being a homograph.

[11]The code is available at https://github.com/VIDA-NYU/domain-discovery-d4
[12]https://networkit.github.io

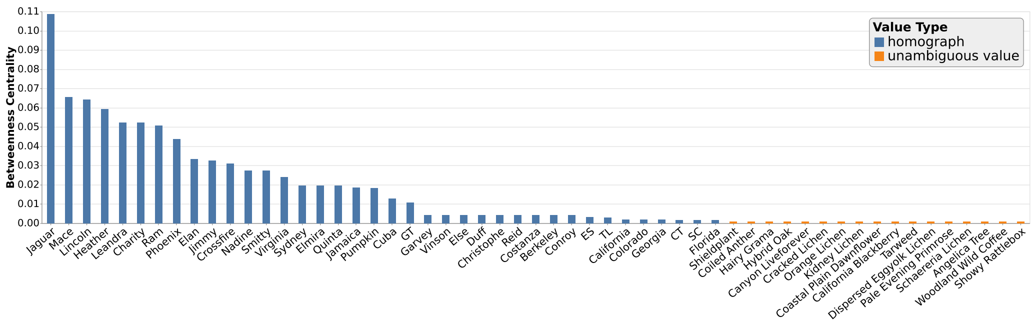


Fig. 8. The top-55 data values with the greatest betweenness centrality scores. In the top-55 data values, 38 of them are homographs. The homographs not in the top-55 are country/state abbreviation homographs.

Notice how more than 75% of the top-ranked data values are not homographs, meaning that a large number of unambiguous values have smaller LCC scores than the homographs. This is mainly caused by unambiguous values from small domains that do not co-occur often with many values in their domain. This confirms our hypothesis from Section 3 that LCC may not work well when homographs appear in small domains. In fact, the majority of the 55 homographs in the dataset have LCC scores significantly above 0.45, and so it is not necessarily true that homographs have low LCC cores. Overall, the results indicate that LCC scores do not provide an effective separation between homographs and unambiguous values.

However, the BC scores result in a vastly better top-55 result, as shown in Figure 8. Here, 38 out of the top-55 BC scores correspond to homographs. This is a much improved outcome over the LCC scores in Figure 7. But what happened to the remaining 17 homographs that are not in the top-55? We noticed that the remaining 17 homographs have betweenness scores of nearly zero and they all are values corresponding to homographs that are abbreviations of country and state names. Recall that these are the only two tables in SB with fewer than 1,000 tuples, where the state table contains only 50 tuples. This means that the BC score for values in these small domains cannot be very large, as there cannot be as many shortest paths that would pass through the homograph in question.

An explanation for the low BC scores for these homographs is the fact that there is considerable intersection between the country and state values, which is not the case with other homographs (e.g., the car brands and cities intersect only on the value Lincoln and Jaguar). This relatively large intersection also reduces the BC scores for those homographs, as the number of shortest paths connecting two nodes between cities and states is much larger. For example, going from the country code GR to the state code MA, the shortest path could be using the homograph AL (which is for Albania/Alabama) or CA (which is for Canada/California) or any other homograph between countries and states. As a result, those homographs receive lower BC scores, because the denominator in Equation (2) becomes large.

How good is previous work at finding homographs? As discussed earlier, we compare `DomainNet` against a competitor based on D^4 [71]. When applied to the SB dataset, D^4 discovers four domains corresponding to `Country`, `Country Code`, `Scientific Animal Name`, and `Scientific Plant Name`. It maps the domains on 14 out of 39 table columns (attributes) in SB. Among these 14 attributes, there are 21 of the 55 homographs. Overall, when considering the top-55 results returned, the D^4-based algorithm disambiguates homographs in SB with a precision, recall, and F1-score of 38%. Using the BC score, `DomainNet` achieves for the top-55 results a precision, recall, and F1-score of 69%.

Table 2. Percentage of the 50 Injected Homographs Appearing in the Top-50 Results vs. Cardinality of the Data Values Replaced by the Injected Homograph

Cardinality of replaced values	> 0	≥ 100	≥ 200	≥ 300	≥ 400	≥ 500
% of injected homographs in top 50	85%	93.5%	93.5%	95%	94.5%	97.5%

(Numbers are averages of four runs for each threshold).

Table 3. Percentage of Injected Homographs in the Top-50 According to Betweenness Centrality while Varying the Number of Meanings of the Injected Homographs

# meanings of injected homographs	2	3	4	5	6	7	8
% of homographs in top 50	97.5	97.5	98.5	98.5	100	100	100

6.2 Experimental Evaluation on TUS-I

We now study the BC-score-based version of `DomainNet` in more detail on the large real-world dataset TUS-I with the injected homographs. Due to the cost of running BC for each node, all BC scores are approximated using 5,000 samples.[13]

How does cardinality affect homograph discovery? Recall that after removing all original homographs in TUS, the TUS-I dataset only contains the homographs we methodically injected to study a specific effect on betweenness centrality. We ran our experiments by randomly selecting 50 pairs of values from different domains[14] and replaced them with our 50 injected homographs. Each experiment was repeated four times with a different seed for selecting the values for replacement. Since the number of homographs in our experiment is always 50, in an ideal scenario the top-50 BC scores would correspond to exactly those injected homographs.

We found that cardinality has the expected impact on BC scores in terms of separating homographs and unambiguous values. If the data values chosen for replacement have a not-too-small cardinality (i.e., they co-occur with many other values), then the BC score of their injected homograph was notably higher. We confirmed this observation in Table 2, where we varied the cardinality threshold for the data values chosen for replacement. Overall, as we increased the cardinality threshold, a larger percentage of the injected homographs ranked in the top-50. In fact, if the replaced values had a cardinality of 500 or higher, then `DomainNet` consistently ranked at least 48 of the 50 injected homographs in the top 50. For reference, the largest attribute in TUS has 25,000 values and over half of all attributes have more than 500 values.

How does the number of meanings of a homograph affect homograph discovery? In addition to varying the cardinality of the replaced values, we examined how the number of meanings of the injected homographs impacts their BC-based rankings. The number of meanings of an injected homograph is the number of values replaced for each injected homograph. The replaced values are all chosen from different domains to ensure that the injected homographs have consistently the specified number of meanings. We explored injected homographs with the number of meanings in the range 2 to 8 for replaced data values with a cardinality of 500 or higher. Table 3 shows that as we increase the number of meanings, `DomainNet` becomes better at discovering them. This is consistent with our intuition for betweenness centrality, since homographs with more meanings are more likely to be hub nodes that connect multiple sets of nodes with each other in our bipartite graph representation of the data lake.

[13]A common heuristic for the sample size is about 1%–3% of the total number of nodes in the graph. This works well in practice with sparse graphs like `DomainNet` [36]. We will further test the validity of this heuristic in Section 6.5.
[14]Different domains in the TUS benchmark context means values from columns that are not unionable with each other.

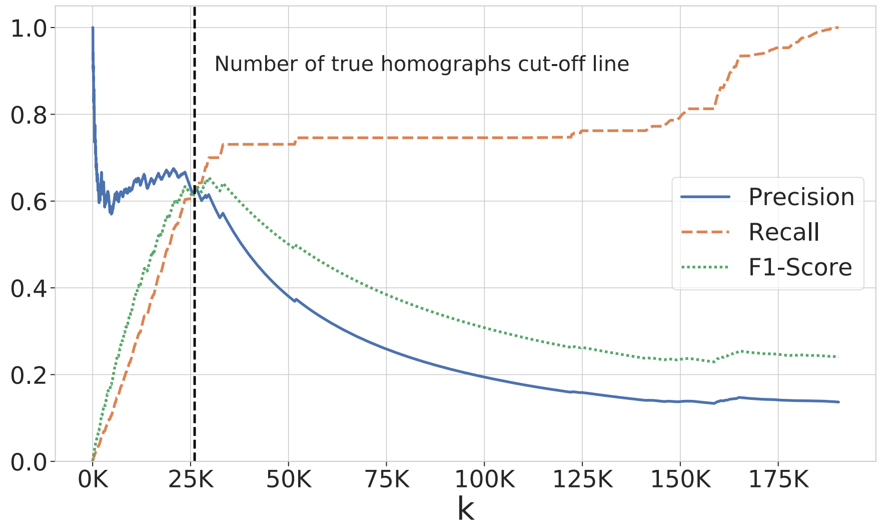


Fig. 9. Top-k evaluation on the TUS dataset. The vertical line at k=26,035 denotes the number of true homographs in the dataset.

6.3 Homographs in TUS Benchmark

Last, we explore the performance of `DomainNet` with betweenness centrality on the real TUS dataset with its 26,035 real homographs. Since the number of homographs is large, we not only report precision, recall, and F1-score for the top-26,035 results, but for all top-k with k from 1 all the way to the number of nodes in our graph, i.e., 190,399. We do not compare against the D^4-based algorithm for homographs, because D^4 operates only on string attributes, and given the large number of numerical attributes, the D^4 coverage will be even lower than in SB (where it only finds domains for 14 out of 39 attributes).

How does our approach perform on a real open-data benchmark? Figure 9 shows the summary of our top-k evaluation results. Notice that for relatively small values of k such as $k = 200$, our method can identify homograph values with high precision (0.89). Naturally, as we increase k, precision decreases and recall increases. At $k = 26{,}035$ (vertical line in Figure 9), which is the number of true homographs in the TUS benchmark, we achieve a precision, recall, and F1-score of 0.622. The highest F1-score occurs at $k = 29{,}633$, where precision, recall, and F1-score are 0.615, 0.7, and 0.655, respectively.

It is important to emphasize that our approach is completely unsupervised and does not assume any external knowledge about the tables or their values. Existing state-of-the-art methods that tackle data integration tasks as described in Section 2 cannot be readily used for homograph identification or their coverage is severely limited (e.g., knowledge-based approaches like AIDA [97]).

Below, we report the top-10 values and their BC scores from the TUS benchmark:

— "Music Faculty" $\longrightarrow$ 0.00064
— "Manitoba Hydro" $\longrightarrow$ 0.00045
— "50" $\longrightarrow$ 0.00029
— "1800ZZMALDY2" $\longrightarrow$ 0.00028
— "." $\longrightarrow$ 0.00027
— "Conseil de développement" $\longrightarrow$ 0.00025

— "125" → 0.00023
— "2" → 0.00022
— "Biomedical Engineering" → 0.00022
— "SQA" → 0.00016.

All 10 data values are homographs based on the ground truth. Notice that, from a natural-language perspective, these 10 values do not seem to be homographs, but a closer look at the data revealed good reasons why they were labeled as homographs. For example, the value *Music Faculty* appears in two distinct contexts: as a geographic location/landmark in transportation-related tables as well as a department in university-related tables.

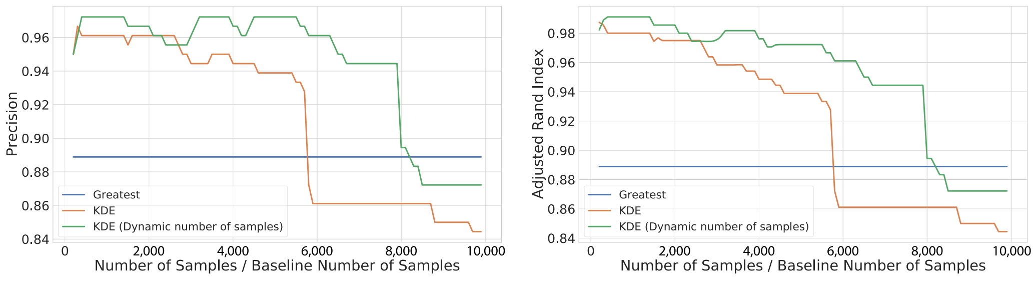
The value with the fifth-highest BC score is the period character. This may seem bizarre, but the period is used extensively as a null replacement in a large variety of tables and thus it acts as a homograph with a very large number of meanings. Finally, notice that we identify numerical values such as 50, 125, and 2, which appear in a variety of contexts such as addresses, identification numbers, quantity of products, and so on. Numerical values are traditionally difficult to deal with in many data-integration tasks, hence being able to identify some of them in a completely unsupervised manner is a notable step toward better coverage for numerical values.

6.4 Homograph Meanings Clustering

We solve the problem of grouping attributes containing a homograph into groups that reflect the different meanings of the homograph using the method described in Section 4.1. We experiment with the choice of ε using both Heuristic 4.3 and Heuristic 4.4 as well as our choice of the number of samples that can either be fixed for any range of distances or assigned dynamically using Equation (3). In particular, we examine the precision in identifying the number of meanings for all homographs in the SB as well as the quality of the clusters of attributes (i.e., how well are attributes grouped together that contain the same meaning instances of a homograph) generated after running DBSCAN. Figure 10(a) shows the average precision in identifying the correct number of meanings for each homograph across three different methods: (1) *Greatest* assigns an ε using Heuristic 4.3 with threshold $\tau = 0.99$, (2) *KDE* assigns an ε using Heuristic 4.4, and (3) *KDE (Dynamic number of samples)* also assigns an ε using Heuristic 4.4, but the number of samples used by KDE is specified using Equation (3). Notice how the precision using *Greatest* in Figure 10(a) is constant, since it is not using KDE for which we vary the number of samples. Also the x-axis measure for *KDE (Dynamic number of samples)* in Figure 10(a) corresponds to the baseline number of samples, as the number of samples is determined using Equation (3).

As shown in Figure 10(a) both *KDE* and *KDE (Dynamic number of samples)* outperform *Greatest* for a large range of number of samples. Moreover, dynamically determining the number of samples rather than using a fixed number of samples when running KDE also seems to give a better precision, as we are less prone to overfitting the distributions with a small range of pairwise distances. This is further confirmed by looking at the deteriorating precision of *KDE* and *KDE (Dynamic number of samples)* as we increase the number of samples or baseline number of samples, respectively. In that case, excessive sampling can often lead to overfitting where there can be multiple distinct peaks in the KDE distribution that correspond to pairwise distances between attributes of different types, and as a result Heuristic 4.4 becomes less accurate. Notice also that *KDE (Dynamic number of samples)* retains a higher precision for a larger range of baseline number of samples than *KDE,* as it is able to reduce overfitting by using fewer samples for distributions of pairwise distances with a small range.

Figure 10(b) shows the average quality of the identified clusters for all homographs after running DBSCAN across the three different methods: *Greatest*, *KDE*, and *KDE (Dynamic number of samples).*



(a) Number of meanings precision

(b) Clustering of attributes quality

Fig. 10. Figure 10(a) shows the average precision in identifying the correct number of meanings across all homographs in the Synthetic Benchmark. Note that for the *KDE (Dynamic number of samples)* the x-axis measure corresponds to the baseline number of samples, since the number of samples is determined using Equation (3). Figure 10(b) shows the quality of the identified clusters of attributes across all homographs in the Synthetic Benchmark. The measure of quality used is the adjusted Rand index, where the closer a score is to 1, the higher the quality of the identified clusters.

Since we know the ground truth clusters in the SB, we evaluate the quality of our clusters using the adjusted Rand index [42]. The adjusted Rand index is an extrinsic evaluation measure for clustering that computes a similarity between two clustering assignments by considering all pairs of samples and counting pairs that are assigned to the same or different clusters in the predicted and ground truth clusters [76]. The score is further adjusted to account for chance and ranges from −1 to 1, where a score of 1 indicates a perfect prediction of clusters. As seen in Figure 10(b) the discovered clusters using DBSCAN are fairly close to the ground truth, which means that for most homographs, we can accurately group their attributes based on their type. The effectiveness of each of the three methods coincides with the pattern we observed for precision in Figure 10(a), where *KDE (Dynamic number of samples)* performs the best, as it reduces overfitting that can happen especially as we increase the number of samples for KDE.

The number of meanings of the homographs in SB is relatively small (only 2 or 3; see Table 1). Hence, to further stress-test our method, we also tested our precision and clustering quality by injecting *null value homographs* (meaning homographs with a large number of meanings) in the SB with results consistent with Figure 10. More specifically, we took the SB and injected null values by manually placing various null equivalent strings such as “missing” or “undefined” across multiple columns of different domains. In this way, SB contains some homographs (i.e., null values) with many more meanings than usual, and we can check if we can accurately identify the number of meanings and cluster the attributes for these injected null values. We found that, in most cases, we also predict a much larger number of meanings for the injected null values. Assuming that a large number of meanings often represents a null or a filler value, if we can find a number of meanings larger than a threshold, then we can still correctly identify this particular type of homograph. For example, the injected null value “missing” appears in 17 semantically different attributes, and our approach with KDE predicts that it appears in 16 distinct clusters, so even though the exact precision for the number of meanings is off, the clusters that we do find agree in a large part with the ground truth clusters.

6.5 Scalability

As discussed in Section 3.5, Step 1 (graph construction) and Step 2 (centrality measure computation) are the most computationally expensive in our approach. In this section, we examine empirically the scalability of these steps.

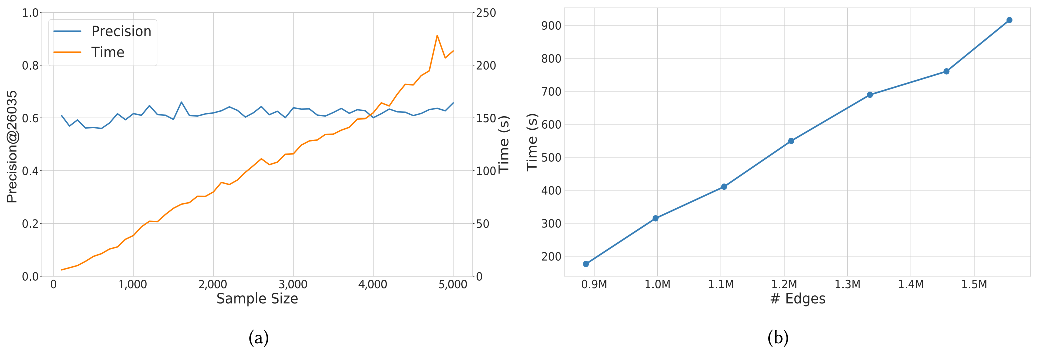


Fig. 11. Figure 11(a) shows the precision at k (where k is the number of homographs in the dataset) and execution time at various sample sizes for approximate BC on the SB and TUS datasets. Exact BC on TUS took 150 minutes with a precision of 0.631. Figure 11(b) shows the runtime of approximate BC for various sized subgraphs based on the NYC education dataset.

The time to construct our bipartite graph is dependent on how long it takes to scan all input tables, which is a relatively fast operation. For example, the bipartite graph for the TUS dataset takes about 1.5 minutes to construct, which is how long it takes to read through each table in the dataset.

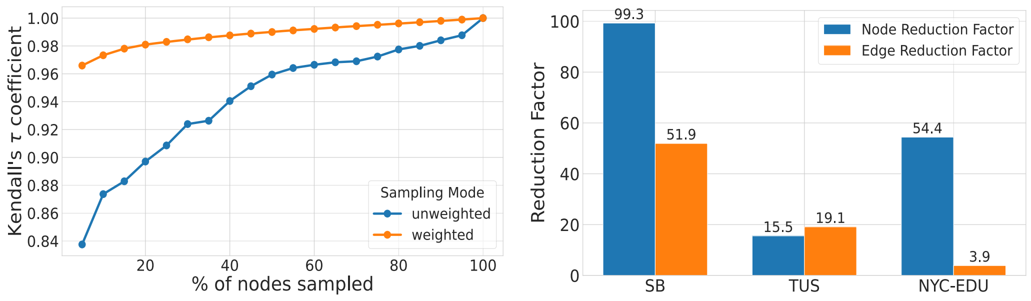
The runtime of Step 2 depends on the graph measure used. LCC is a local measure that is efficient to compute, but, as we demonstrated in Section 6.1, it is not as effective in finding homographs as BC is. Computing the LCC score for every node in the TUS dataset takes 4 seconds. For the global measure BC, since we are more interested in the score rankings rather than the scores themselves, approximating BC via sampling can significantly decrease the runtime without compromising quality.

In Figure 11(a), we examine how precision and runtime vary as we change the number of samples used for the approximate BC algorithm [36] on the TUS benchmark. Even for a small sample size (e.g., 1,000), precision stabilizes at 0.6. Notice that 1,000 samples correspond to around .5% of the nodes in the TUS graph, and it takes about 40 seconds for the algorithm to complete. The BC approximation has a complexity of $O(sm)$, where s is the number of nodes sampled, and m the number of edges in the graph. Based on the literature and testing on our graphs, we found that sampling 1% of the nodes provides a good approximation of BC that is very consistent with the score rankings produced by the exact BC computation.

We also considered a bigger data lake to further test execution times—the NYC education open data dataset as used in D^4 [71]. The bipartite graph representation of that dataset has roughly 1.5M nodes and 2.3M edges, which is an order of magnitude larger than the bipartite graph for the TUS dataset. The graph was constructed in 3.5 minutes, and the BC scores for every node were computed in 27 minutes using approximate BC on 1% of the nodes (~15k nodes).

To examine how runtime scales with graph size, we extracted random subgraphs[15] of various sizes from the bipartite graph used for the NYC education dataset. We ran approximate BC for each graph by sampling 1% of its nodes and measured the runtime. Figure 11(b) shows that runtime increases linearly with graph size (i.e., number of edges), which is in accordance with the $O(sm)$ complexity of the approximate BC algorithm.

[15]The subgraphs were constructed by randomly selecting an attribute node and adding all its connecting value nodes. We repeat by selecting another attribute node until the subgraph reaches the desired size (within some margin).



(a) (b)

Fig. 12. Figure 12(a) shows the impact of the number of nodes sampled and sampling scheme to the accuracy of the approximate BC on the TUS dataset when using graph compression. Figure 12(b) shows the node and edge reduction factors on the SB, TUS, and NYC-EDU datasets when applying node compression.

6.6 Graph Compression Experiments

In this section, we examine the computational speedup we gain by compressing our bipartite `DomainNet` graph before calculating its BC as described in Section 3.4. Figure 12(a) shows the accuracy of the approximate BC scores on the TUS dataset when using graph compression as we vary the percentage of nodes used for sampling as well as our mode of sampling. The y-axis measure in Figure 12(a) corresponds to Kendall's τ coefficient [49], which is a statistic to measure the similarity between two rankings. If two rankings are identical, then they have a Kendall's τ coefficient of 1, and if they are reversed, then $\tau = -1$. Each point in the graph corresponds to a run of approximate BC with node compression and a specified percentage of nodes that were sampled using either a weighted or unweighted scheme. All nodes in the graph are then ranked in descending order by their computed approximate BC scores, and that ranking is then compared with the ranking produced by running exact BC. As seen in Figure 12(a), as the number of nodes sampled increases, the better the approximate BC becomes. Moreover, there is a clear improvement in the accuracy of the approximate BC scores when using weighted sampling, since we are more likely to choose large *ident* scored compressed nodes, which have larger impact in the BC rankings.

Figure 12(b) shows the amounts by which the bipartite `DomainNet` graphs for the SB, TUS, and NYC-EDU datasets were compressed. As seen, both the number of nodes and edges were reduced significantly across all datasets. For instance, the graph from the NYC-EDU dataset has originally $\approx$ 1.47M nodes and was reduced to have only 27,014 nodes (i.e., the number of nodes was reduced by 98%). The amount of reduction is highly dependent on the data repetition and overlap across tables in the data lake, so the TUS dataset, which has a lot of tables that were generated by cutting larger tables horizontally and vertically, leads to compressed nodes that are generally smaller and thus the reduction factor is not as high against other datasets. Nevertheless, TUS still has a sizable 15$\times$ reduction factor in the number of nodes in its compressed graph.

This reduction in the number of nodes and edges leads to significant computational speedups in BC. Running exact BC on the TUS takes about 150 minutes. After applying node compression to the graph, we can compute the exact BC for all nodes in only 38 seconds, a 237$\times$ speedup. A comparison of the exact BC runtimes with and without node compression is shown Table 4.

6.7 Comparison against Supervised Methods

Supervised methods for semantic type detection (see Section 2.2) such as Sherlock [43] and SATO [100] can be adapted to solve our problem of identifying all homographs in a data lake. This can

Table 4. Runtime Comparison of Exact BC with and without Node Compression on the Synthetic Benchmark (SB), Table Union Search (TUS) Dataset, and NYC-Education Dataset

Dataset	Exact BC runtime without compression	Exact BC runtime with compression
SB	5.52 sec	0.005 sec
TUS	150 min	38 sec
NYC-EDU	>5 hours	162 sec

Table 5. Homograph Detection Performance Using Sherlock and SATO over the TUS Benchmark against DomainNet

Measure	DomainNet (@k = 26,035)	Sherlock	SATO
Precision	0.622	0.745	0.178
Recall	0.622	0.500	0.833
F1-score	0.622	0.597	0.296
Runtime	38 seconds	22 minutes	6 hours

The runtime reported for DomainNet is using exact BC with graph compression.

be done by first using their pre-trained models to predict a semantic type for each column in each table. Then a value is identified as a homograph if it is found in columns that are assigned at least two different semantic types. Notice that it is not necessary for the semantic type assignment to be accurate, as long as it is differentiable between columns that are truly semantically different, so homographs can still be identified. For example, if the value "empty" is found in two columns that are assigned the semantic types "first_name" and "plant_name," but in ground truth their assigned types should have been "middle_name" and "plan_name," respectively, then the value "empty" will still be correctly identified as a homograph, even though the semantic types assigned were not accurate. However, incorrect semantic type predication will have an impact in correctly grouping the meanings of the instances of the homographs. Also notice that this adaptation will only return a set of values determined as homographs or unambiguous values and will not provide a ranked list of homographs as DomainNet does.

Using the pre-trained models of Sherlock and SATO and the adaptation outlined above, we compare their homograph detection performance against DomainNet over the TUS, TUS-I, and NYC-EDU datasets. Additionally, we compare their performance in identifying the number of meanings of homographs as well as correctly clustering together the same meaning instances of a homograph. Table 5 shows the performance and runtime of Sherlock and SATO over the TUS benchmark against DomainNet at $k = 26{,}035$. DomainNet outperforms both Sherlock and SATO, but Sherlock performs quite well and has an overall higher precision (but lower recall and F1-score as well as slower performance). Sherlock is able to perform relatively well, since the TUS benchmark contains tables with columns that have semantic types over which Sherlock has a good coverage. We hypothesize that in datasets that are even more heterogeneous, Sherlock would perform considerably worse, a hypothesis that we confirmed when looking at the homograph detection results over the NYC-Education dataset shown in Table 6. One possible explanation for the very low performance results for SATO is due to the construction of the TUS benchmark. The TUS benchmark is constructed from larger tables that are sliced horizontally and vertically to construct many smaller table subsets, and SATO, unlike Sherlock, is a method that also considers row context information for its semantic type detection. The table splitting may be a cause for SATO's worse performance (in homograph detection) over Sherlock. Note that SATO predicted that 130,859 of the 190,399 unique values of the TUS dataset were determined to belong in columns with at least

Table 6. Homograph Detection, Number of Homograph Meanings, and Same Meaning Clustering Performance Using Sherlock over the NYC-Education Dataset against DomainNet over the Top-29 Non-numeric Values

Measure	DomainNet	Sherlock
Homograph Detection Precision	0.793	0.517
Number of Meanings Precision	0.690	0.310
Assigned Meanings Adj. Rand Index	0.720	0.383

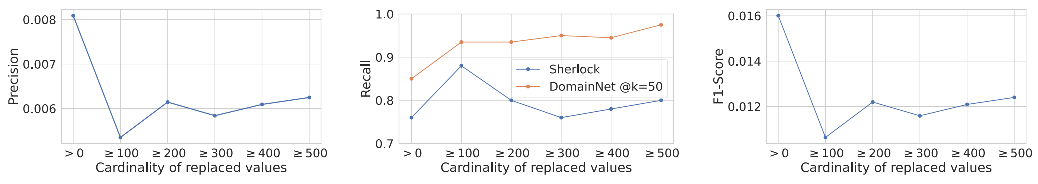
two semantic types (i.e., homographs) even though there are only 26,035 homographs in ground truth. Because of this behavior as well as SATO's slow running time, in the remainder of our experiments, we compared only against Sherlock as the best supervised semantic type detection method for homograph detection.

To test Sherlock on an even more heterogeneous dataset, we used the NYC-Education dataset. Since we do not know the homograph ground truth in the NYC-Education dataset, we selected the top-100 highest BC values identified using DomainNet and manually labeled them if they were indeed homographs. In the top-100, a total of 71 numeric values were present, which we removed to allow for a fair comparison against Sherlock, since it does not perform as well on numeric-only columns. This leaves us with 29 unique text values of which many were indeed homographs. For example, the value "Chemistry" appears both as a course as well as a title/description of a statue/landmark in a school. The value "–" appears as a null equivalent value for phone numbers, addresses, corporation names, and so on. The value "EMER" appears as a first name as well as a shorthand for emergency in project budget tables. As seen from Table 6. Sherlock's precision is noticeably worse than in the TUS benchmark due to the presence of columns with semantic types that do fall into the 78 types Sherlock was trained on. As a consequence, the number of meanings predicted for each non-numeric value in the top-29 is also low. Similarly the quality of the clusters that group the same meaning instances of a homograph together are also low for Sherlock when compared to DomainNet.

We also tested Sherlock against DomainNet over our TUS-I benchmark, where we inject a small number of homographs over a cleaned dataset with no known homographs. More specifically, we experimented by varying the cardinality of the data values replaced by the injected homographs exactly as in Section 6.2. Figure 13 shows the performance of Sherlock vs. DomainNet, where although Sherlock achieves a slightly lower recall than DomainNet, its precision is extremely low. This is because Sherlock wrongly assigned different semantic types to many time and product code columns and thus a large number of values that were not homographs were identified as homographs.

6.8 Comparison against Overlapping Community Detection Techniques

We compare the performance of overlapping community detection algorithms over our bipartite graph to identify homographs. As discussed in Section 3.3, identifying homographs using community detection techniques can be challenging, as the number of communities can be very large in massive repositories of tables. Moreover, the communities are not disjoint, since values (specifically, homographs) can be members of multiple communities. We ran four SOTA overlapping community detection algorithms, namely, Big-Clam [94], LPANNI [63], DANMF [96], and Core Expansion [20], over the bipartite graph representation for the SB and the TUS benchmark. We used the implementations provided by cdlib [80], a popular Python library that provides implementations for multiple community detection algorithms with the aim of improving evaluation



(a) Precision (b) Recall (c) F1-Score

Fig. 13. Homograph detection performance over the TUS-I benchmark using Sherlock vs. DomainNet as the cardinality of the replaced values is varied. Notice that DomainNet's precision and F1-score at $k = 50$ is the same as its recall plot but is not shown to make Sherlock's data readable.

Table 7. Performance of Overlapping Community Detection Algorithms over the Synthetic Benchmark (SB)

Algorithm	#Communities	#Nodes in multiple communities	Runtime (seconds)	Precision	Recall	F1-score
Core expansion [20]	33	36	20.8	0.75	0.49	0.59
LPANNI [63]	16,438	0	32.3	-	-	-
Big-Clam [94]	2	0	42.3	-	-	-
DANMF [96]	8	0	76.5	-	-	-

Table 8. Performance of Overlapping Community Detection Algorithms over the Table-Union-Search (TUS) Benchmark

Algorithm	#Communities	#Nodes in multiple communities	Runtime (minutes)	Precision	Recall	F1-score
Core expansion [20]	237	4,191	306	0.31	0.05	0.085
LPANNI [63]	-	-	>5 hours	-	-	-
Big-Clam [94]	2	0	25.3	-	-	-
DANMF [96]	7	0	39.8	-	-	-

and benchmarking across them. Any cell-value node that is found in more than one community, we consider to be a homograph.

Table 7 shows the performance of the four algorithms over the SB, and Table 8 shows their performance over the TUS benchmark. As shown in Table 7 and Table 8, the overlapping community detection algorithms with the exception of core expansion [20] are unable to find a cell-value node that belongs in more than one community. Moreover, even though core expansion does find overlapping nodes, its performance is not comparable to that of DomainNet. We also experimented with using the dense cell nodes co-occurrence graph representation rather than our bipartite graph representation, but the graph becomes extremely large, and the overlapping community detection algorithms would not run in a reasonable amount of time (e.g., the cell nodes co-occurrence graph for just the synthetic benchmark has over 7 million edges).

6.9 Impact of Homograph Discovery on Domain Discovery

As shown in Table 1 the number of homographs in a real data lake can be large. To further understand the impact of homographs on existing approaches, we consider the task of domain discovery and examine how knowing homographs *a priori* can benefit them.

We report the results of five different runs of D^4 in Figure 14. The plots show the number of domains found by D^4 (y-axis) as we vary the number and meanings of the injected homographs. To be fair in the comparison and to understand the impact of homographs on the domain discovery task, we use the TUS-I benchmark. We first ran D^4 over the dataset without homographs and then over the same dataset with injected homographs. More specifically, we injected 50, 100, 150, and 200 homographs with 2, 4, and 6 meanings. In all the above configurations, the dataset always

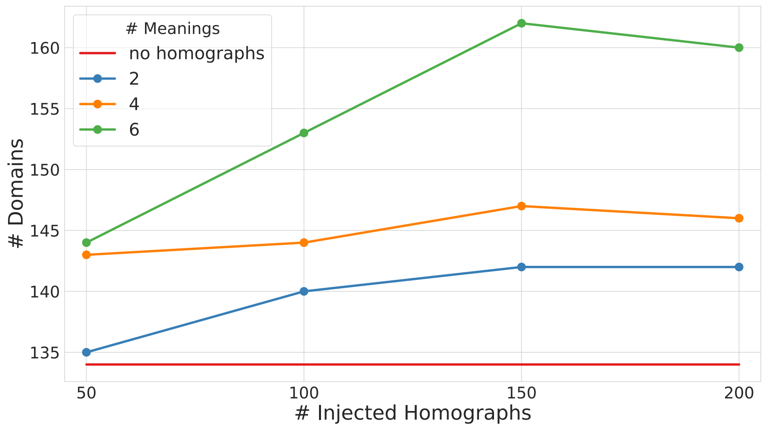


Fig. 14. Number of domains found by D^4 over different TUS-injected datasets. The horizontal red line shows the number of domains found when no homographs were present in the dataset.

had 68 domains based on the ground truth. The horizontal line in Figure 14 shows that D^4 returns 134 domains for TUS-I with no homographs. The difference in the number of domains based on the ground truth and D^4's results is due to the nature of the TUS benchmark [68], as it is created from a set of large real open data tables that were randomly sliced vertically and horizontally. Consequently, in some cases the columns originating from the same table no longer share any values, causing D^4 to discover more domains than there are based on ground truth.

As we increase the number and meanings of the injected homographs, D^4 returns even more domains leading to lower accuracy. D^4's output provides statistics about the maximum and the average number of domains assigned to a column. In the TUS-I with no homographs, that maximum is 2 and the average is almost 1 (i.e., 1.031), and it increases with the number of homographs. With 200 homographs, the maximum is 4 and the average is 1.04. We also ran D^4 on the TUS-I with 5,000 injected homographs to simulate a dataset with a large proportion of homographs as in the TUS benchmark. The maximum domains per column is 22, and the average is 1.7 with a total of 371 domains found. The presence of homographs is negatively affecting D^4 and causing it to erroneously assign larger numbers of heterogeneous domains to attributes as the number of homographs increases. Homograph discovery, therefore, is an important step that can be executed before domain discovery to improve its performance.

6.10 Impact on Downstream Task of Entity Matching

Although entity matching, as we discussed in Section 2, cannot be used to solve the homograph discovery problem, the presence of homographs can have an impact in the performance of even state-of-the-art transformer-based EM methods such as DITTO [59]. To show the impact of homographs, we examined some of the datasets used in DITTO's evaluation, and in each dataset, we introduced some homographs by replacing two previously unambiguous values with a single homograph. For example, in the *(Textual) Abt-Buy* dataset, we can introduce a homograph for the brand name column by replacing all instances of the values "Microsoft" and "Yamaha" with the same string. We ensured that in all datasets and cases where this replacement occurred, our homographs would not change the ground truth for any pair of table rows that were used in evaluating the EM task. Moreover, only about 1% of the pairs of table rows were updated. Table 9 shows the performance of DITTO over four different datasets before and after the introduction of a few homographs. As shown, homographs can have a noticeable impact in the entity matching quality in

Table 9. Impact of Injected Homographs for the Entity Matching Task Using DITTO [59]

Dataset	F1-score (Original)	F1-score (Injected Homographs)	# of entries with homographs	incorrect matching proportion
(Structured) Beers	0.875	0.875	11	0
(Dirty) DBLP-ACM	0.985	0.985	42	0
(Textual) Abt-Buy	0.799	0.793	36	0.17
(WDC) Cameras Small	0.808	0.801	42	0.26

The last column ("incorrect matching proportion") denotes the proportion of wrong prediction out of the entries that were injected with homographs. Notice that over the highly textual datasets such as *(Textual) Abt-Buy* and *(WDC) Cameras Small* about a quarter of the entries with injected homographs resulted in a wrong match/no-match prediction by DITTO.

the two highly textual datasets and thus being able to identify them in a pre-processing step can be beneficial. For instance, in the *(WDC) Cameras Small* dataset, 42 pairs of rows contained our injected homographs, of which 11 pairs were incorrectly predicted for the match/no match task even though they were all correctly predicted before the injection of homographs.

7 CONCLUSION AND FUTURE WORK

We presented DomainNet, a method for finding homographs in data lakes. To the best of our knowledge, this is the first solution for disambiguating data values in data lakes. Notably, our approach does not require complete or consistent attribute names. We showed that a measure of centrality can effectively separate homographs from unambiguous values in a data lake by representing tables as a network of connections between values and attributes.

We compared against an alternative approach using D^4 to identify the semantic domain (type) of attributes [71] and labeling a value a homograph if it appears in more than one domain. Moreover, we also compared DomainNet against supervised semantic type detection techniques (i.e., Sherlock [43] and SATO [100]) and community detection algorithms [20, 63, 94, 96]. Our direct computation of homographs has significantly better precision and recall than the domain-discovery, semantic type detection, and community detection approaches. When we inject homographs into real data, DomainNet is robust to the number of meanings of the homographs, reliably finding homographs with even better accuracy as the number of meanings increases.

In a benchmark created from real data, our method provides a clear separation with high precision of homographs from values that are repeated, but always with the same meaning. The accuracy is influenced by the number of data values with which the homograph co-occurs within the data lake. When this number is too small, the bipartite graph representation is not always sufficient to effectively identify all homographs. In our experiments, the accuracy dropped from 97% to 85% as we reduced this number.

The homographs we discover on real data include phrases with multiple meanings (e.g., Music Faculty referring both to a geographic location and to a University unit). They also include null values (e.g., a dot "." can indicate unknown/missing X where X varies in different contexts) and data errors (e.g., Manitoba Hydro, an electric company, is placed in the wrong column Street Name). In NLP, previous work on disambiguation primarily focuses on the disambiguation of words and named-entities. Our method is purely based on co-occurrence information and does not discriminate between different data types of homographs. In fact, we provide the first approach to disambiguate numerical values in tables (e.g., 25 can be a street number or an ID number).

Our approach is motivated by work on community detection where a community represents a meaning for a value (e.g., animal or car manufacturer). We provided the first technique for determining the number of meanings of a homograph and also determining if two occurrences of a homograph represent the same or different meanings. Importantly, we showed that our approach is robust even if the number of meanings is large.

Identifying homographs from tables in a completely unsupervised manner can play an important role in improving other data-lake analysis tasks. For example, by identifying a homograph and its meanings, if there is a meaning that only occurs in one attribute, then a data scientist can investigate if this is an example of a data entry or shift-error where a value has been placed in the wrong attribute. With such knowledge, we can help not only identify such errors, but clean them as well. We have shown that knowledge of homographs can improve existing data integration techniques such as domain discovery [71] as well as entity matching [59].

To the best of our knowledge, there are no available benchmarks for homograph detection. Our **synthetic benchmark (SB)** and our benchmarks TUS and TUS-I (the latter two use real open data tables [68]) are the first open benchmarks in this area. Importantly, they allow the varying of the number of homographs and also the number of meanings of a homograph.

To design a robust and completely unsupervised solution that scales to large data lakes, we have quite deliberately limited `DomainNet` to use only value co-occurrence information in table columns, ignoring additional structural information like co-occurrence of values in the same row. Our goal was to explore how much this information alone reveals about data value semantics. Given our strong positive results, we believe our metrics should become an important feature that could be used in other problems that involve understanding or integrating tables. An important open problem is to extend `DomainNet` to *collectively* resolve ambiguous metadata and data, perhaps using probabilistic graphical models that have been applied to collectively resolving multiple types of entities at once [52] and to collectively resolving data and metadata inconsistency in schema mapping [51].

REFERENCES

[1] Asma Ben Abacha and Pierre Zweigenbaum. 2015. MEANS: A medical question-answering system combining NLP techniques and semantic web technologies. *Inf. Process. Manag.* 51, 5 (2015), 570–594.

[2] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *VLDB J.* 28, 5 (2019), 793–819.

[3] Rodrigo Agerri and German Rigau. 2016. Robust multilingual named entity recognition with shallow semi-supervised features. *Artif. Intell.* 238 (2016), 63–82. DOI: https://doi.org/10.1016/j.artint.2016.05.003

[4] Rubayyi Alghamdi and Khalid Alfalqi. 2015. A survey of topic modeling in text mining. *Int. J. Adv. Comput. Sci. Appl.* 6, 1 (2015).

[5] Alessia Amelio and Clara Pizzuti. 2014. Overlapping community discovery methods: A survey. In *Social Networks: Analysis and Case Studies*, Sule Gündüz Ögüdücü and A. Sima Etaner-Uyar (Eds.). Springer, 105–125. DOI: https://doi.org/10.1007/978-3-7091-1797-2_6

[6] Siddhant Arora and Srikanta Bedathur. 2020. On embeddings in relational databases. *CoRR* abs/2005.06437 (2020).

[7] David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. 2007. Approximating betweenness centrality. In *5th International Workshop on Algorithms and Models for the Web-Graph (Lecture Notes in Computer Science*, Vol. 4863). Springer, 124–137. DOI: https://doi.org/10.1007/978-3-540-77004-6_10

[8] Maciej Besta and Torsten Hoefler. 2018. Survey and taxonomy of lossless graph compression and space-efficient graph representations. *CoRR* abs/1806.01799 (2018).

[9] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data* 1, 1 (2007), 5.

[10] Indrajit Bhattacharya, Lise Getoor, and Yoshua Bengio. 2004. Unsupervised sense disambiguation using bilingual probabilistic models. In *Annual Meeting of the Association for Computational Linguistics.* ACL, 287–294.

[11] Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. 2012. SODA: Generating SQL for business users. *Proc. VLDB Endow.* 5, 10 (2012), 932–943.

[12] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *International Conference on Management of Data (SIGMOD'08).* ACM, 1247–1250. DOI: https://doi.org/10.1145/1376616.1376746

[13] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *J. Math. Sociol.* 25, 2 (2001), 163–177.

[14] Ursin Brunner and Kurt Stockinger. 2020. Entity matching with transformer architectures—A step forward in data integration. In *International Conference on Extending Database Technology.* OpenProceedings.org, 463–473.

[15] Ursin Brunner and Kurt Stockinger. 2021. ValueNet: A natural language-to-SQL system that learns from database information. In *IEEE International Conference on Data Engineering*. IEEE, 2177–2182.
[16] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *International Conference on Management of Data (SIGMOD'20)*. 1335–1349.
[17] Ümit V. Çatalyürek, Kamer Kaya, Ahmet Erdem Sariyüce, and Erik Saule. 2013. Shattering and compressing networks for betweenness centrality. In *13th SIAM International Conference on Data Mining*. SIAM, 686–694. DOI: https://doi.org/10.1137/1.9781611972832.76
[18] Tanmoy Chakraborty, Ayushi Dalmia, Animesh Mukherjee, and Niloy Ganguly. 2017. Metrics for community analysis: A survey. *ACM Comput. Surv.* 50, 4 (2017), 54:1–54:37. DOI: https://doi.org/10.1145/3091106
[19] Mostafa Haghir Chehreghani. 2014. An efficient algorithm for approximate betweenness centrality computation. *Comput. J.* 57, 9 (2014), 1371–1382. DOI: https://doi.org/10.1093/comjnl/bxu003
[20] Ali Choumane, Ali Awada, and Ali Harkous. 2020. Core expansion: A new community detection algorithm based on neighborhood overlap. *Soc. Netw. Anal. Min.* 10, 1 (2020), 30. DOI: https://doi.org/10.1007/s13278-020-00647-6
[21] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.* 53, 6, Article 127 (2020), 42 pages. DOI: https://doi.org/10.1145/3418896
[22] Diane J. Cook and Lawrence B. Holder. 1994. Substructure discovery using minimum description length and background knowledge. *J. Artif. Intell. Res.* 1 (1994), 231–255. DOI: https://doi.org/10.1613/jair.43
[23] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. 2017. KBQA: Learning question answering over QA corpora and knowledge bases. *Proc. VLDB Endow.* 10, 5 (2017), 565–576.
[24] Vinícius da Fonseca Vieira, Carolina Ribeiro Xavier, and Alexandre Gonçalves Evsukoff. 2020. A comparative study of overlapping community detection methods from the perspective of the structural properties. *Appl. Netw. Sci.* 5, 1 (2020), 51. DOI: https://doi.org/10.1007/s41109-020-00289-9
[25] Danica Damljanovic, Valentin Tablan, and Kalina Bontcheva. 2008. A text-based query interface to OWL ontologies. In *Language Resources and Evaluation Conference*. European Language Resources Association.
[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 4171–4186. DOI: https://doi.org/10.18653/v1/N19-1423
[27] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2020. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. *CoRR* abs/2010.13273 (2020).
[28] Mohnish Dubey, Sourish Dasgupta, Ankit Sharma, Konrad Höffner, and Jens Lehmann. 2016. AskNow: A framework for natural language query formalization in SPARQL. In *Extended Semantic Web Conference (Lecture Notes in Computer Science*, Vol. 9678). Springer, 300–316.
[29] Julian Eberius, Patrick Damme, Katrin Braunschweig, Maik Thiele, and Wolfgang Lehner. 2013. Publish-time data integration for open data platforms. In *2nd International Workshop on Open Data (WOD'13)*. 1:1–1:6. DOI: https://doi.org/10.1145/2500410.2500413
[30] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. 2015. Top-k entity augmentation using consistent set covering. In *International Conference on Scientific and Statistical Database Management*. ACM, 8:1–8:12. DOI: https://doi.org/10.1145/2791347.2791353
[31] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231. Retrieved from http://www.aaai.org/Library/KDD/1996/kdd96-037.php
[32] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. 2019. Lazo: A cardinality-based method for coupled estimation of Jaccard similarity and containment. In *International Conference on Data Engineering*. IEEE, 1190–1201.
[33] Anderson A. Ferreira, Marcos André Gonçalves, and Alberto H. F. Laender. 2012. A brief survey of automatic methods for author name disambiguation. *SIGMOD Rec.* 41, 2 (2012), 15–26. DOI: https://doi.org/10.1145/2350036.2350040
[34] Marcus Fontoura, Maxim Gurevich, Vanja Josifovski, and Sergei Vassilvitskii. 2011. Efficiently encoding term co-occurrences in inverted indexes. In *Conference on Information and Knowledge Management*. ACM, 307–316.
[35] Linton C. Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* 40, 1 (1977), 35–41.
[36] Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better approximation of betweenness centrality. In *Algorithm Engineering and Experiments Conference*. SIAM, 90–100. DOI: https://doi.org/10.1137/1.9781611972887.9
[37] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: Theory, practice & open challenges. *PVLDB* 5, 12 (2012), 2018–2019.
[38] Aman Goel, Craig A. Knoblock, and Kristina Lerman. 2012. Exploiting structure within data for accurate labeling using conditional random fields. In *14th International Conference on Artificial Intelligence (ICAI'12)*.

[39] Oktie Hassanzadeh, Michael Jeffrey Ward, Mariano Rodriguez-Muro, and Kavitha Srinivas. 2015. Understanding a large corpus of web tables through matching with knowledge bases: An empirical study. In *10th International Workshop on Ontology Matching (CEUR Workshop Proceedings)*. CEUR-WS.org, 25–34. Retrieved from http://ceur-ws.org/Vol-1545/om2015_TLpaper3.pdf

[40] Keith Henderson, Tina Eliassi-Rad, Spiros Papadimitriou, and Christos Faloutsos. 2010. HCDF: A Hybrid Community Discovery Framework. In *SIAM International Conference on Data Mining*. SIAM, 754–765.

[41] Kevin Zeng Hu, Snehalkumar (Neil) S. Gaikwad, Madelon Hulsebos, Michiel A. Bakker, Emanuel Zgraggen, César A. Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çagatay Demiralp. 2019. VizNet: Towards a large-scale visualization learning and benchmarking repository. In *ACM Conference on Human Factors in Computing Systems*. ACM, 662. DOI: https://doi.org/10.1145/3290605.3300892

[42] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *J. Classif.* 2, 1 (1985), 193–218.

[43] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *International Conference on Knowledge Discovery and Data Mining*. ACM, 1500–1508. DOI: https://doi.org/10.1145/3292500.3330993

[44] Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. 2016. Embeddings for word sense disambiguation: An evaluation study. In *Annual Meeting of the Association for Computational Linguistics*. ACL. DOI: https://doi.org/10.18653/v1/p16-1085

[45] Muhammad Aqib Javed, Muhammad Shahzad Younis, Siddique Latif, Junaid Qadir, and Adeel Baig. 2018. Community detection in networks: A multidisciplinary review. *J. Netw. Comput. Appl.* 108 (2018), 87–111. DOI: https://doi.org/10.1016/j.jnca.2018.02.011

[46] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. 2012. Gbase: An efficient analysis platform for large graphs. *VLDB J.* 21, 5 (2012), 637–650. DOI: https://doi.org/10.1007/s00778-012-0283-9

[47] Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer. 2007. NLP-reduce: A naive but domain independent natural language interface for querying ontologies. In *4th European Semantic Web Conference*. Springer, 1–2.

[48] Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. 2006. Querix: A natural language interface to query ontologies based on clarification dialogs. In *5th International Semantic Web Conference (ISWC'06)*. Citeseer, 980–981.

[49] Maurice G. Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.

[50] Anurag Khandelwal, Zongheng Yang, Evan Ye, Rachit Agarwal, and Ion Stoica. 2017. ZipG: A memory-efficient graph store for interactive queries. In *ACM International Conference on Management of Data (SIGMOD'17)*. ACM, 1149–1164. DOI: https://doi.org/10.1145/3035918.3064012

[51] Angelika Kimmig, Alex Memory, Renée J. Miller, and Lise Getoor. 2019. A collective, probabilistic approach to schema mapping using diverse noisy evidence. *IEEE Trans. Knowl. Data Eng.* 31, 8 (2019), 1426–1439.

[52] Pigi Kouki, Jay Pujara, Christopher Marcum, Laura M. Koehly, and Lise Getoor. 2019. Collective entity resolution in multi-relational familial networks. *Knowl. Inf. Syst.* 61, 3 (2019), 1547–1581. DOI: https://doi.org/10.1007/s10115-018-1246-2

[53] Christos Koutras, Marios Fragkoulis, Asterios Katsifodimos, and Christoph Lofi. 2020. REMA: Graph embeddings-based relational schema matching. In *International Conference on Extending Database Technology*.

[54] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia. *Semant. Web* 6, 2 (2015), 167–195. DOI: https://doi.org/10.3233/SW-140134

[55] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A large public corpus of web tables containing time and context metadata. In *World Wide Web Conference*. 75–76. DOI: https://doi.org/10.1145/2872518.2889386

[56] Aristotelis Leventidis, Laura Di Rocco, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2021. DomainNet: Homograph detection for data lake disambiguation. In *24th International Conference on Extending Database Technology*. 13–24. DOI: https://doi.org/10.5441/002/edbt.2021.03

[57] Felipe Hoppe Levin and Carlos A. Heuser. 2010. Evaluating the use of social networks in author name disambiguation in digital libraries. *J. Inf. Data Manag.* 1, 2 (2010), 183–198. Retrieved from http://seer.lcc.ufmg.br/index.php/jidm/article/view/35

[58] Fei Li and H. V. Jagadish. 2016. Understanding natural language queries over relational databases. *SIGMOD Rec.* 45, 1 (2016), 6–13.

[59] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.* 14, 1 (Sep. 2020), 50–60. DOI: https://doi.org/10.14778/3421424.3421431

[60] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and searching web tables using entities, types and relationships. *PVLDB* 3, 1 (2010), 1338–1347. Retrieved from http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R118.pdf

[61] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR* abs/1907.11692 (2019).
[62] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *ACM Comput. Surv.* 51, 3 (2018), 62:1–62:34. DOI: https://doi.org/10.1145/3186727
[63] Meilian Lu, Zhenglin Zhang, Zhihe Qu, and Yu Kang. 2019. LPANNI: Overlapping community detection using label propagation in large-scale complex networks. *IEEE Trans. Knowl. Data Eng.* 31, 9 (2019), 1736–1749. DOI: https://doi.org/10.1109/TKDE.2018.2866424
[64] Antonio Maccioni and Daniel J. Abadi. 2016. Scalable pattern matching over compressed graphs via dedensification. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1755–1764. DOI: https://doi.org/10.1145/2939672.2939856
[65] Sebastian Maneth and Fabian Peternek. 2015. A survey on methods and systems for graph compression. *CoRR* abs/1504.00616 (2015).
[66] Renée J. Miller. 2018. Open data integration. *PVLDB* 11, 12 (2018), 2130–2139. DOI: https://doi.org/10.14778/3229863.3240491
[67] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data lake management: Challenges and opportunities. *PVLDB* 12, 12 (2019), 1986–1989. DOI: https://doi.org/10.14778/3352063.3352116
[68] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table union search on open data. *PVLDB* 11, 7 (2018), 813–825. DOI: https://doi.org/10.14778/3192965.3192973
[69] Roberto Navigli. 2009. Word sense disambiguation: A survey. *ACM Comput. Surv.* 41, 2 (2009), 10:1–10:69. DOI: https://doi.org/10.1145/1459352.1459355
[70] Mark E. J. Newman. 2010. *Networks: An Introduction.* Oxford University Press. DOI: https://doi.org/10.1093/ACPROF:OSO/9780199206650.001.0001
[71] Masayo Ota, Heiko Mueller, Juliana Freire, and Divesh Srivastava. 2020. Data-driven domain discovery for structured datasets. *PVLDB* 13, 7 (2020), 953–965.
[72] Krishna Kumar P., Paul Langton, and Wolfgang Gatterbauer. 2020. Factorized graph representations for semi-supervised learning from sparse data. In *International Conference on Management of Data (SIGMOD'20)*. ACM, 1383–1398. DOI: https://doi.org/10.1145/3318464.3380577
[73] George Papadakis, Georgios M. Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. 2020. Three-dimensional entity resolution with JedAI. *Inf. Syst.* 93 (2020), 101565. DOI: https://doi.org/10.1016/j.is.2020.101565
[74] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB* 9, 9 (2016), 684–695. DOI: https://doi.org/10.14778/2947618.2947624
[75] Mohammad Taher Pilehvar and Roberto Navigli. 2014. A large-scale pseudoword-based evaluation framework for state-of-the-art word sense disambiguation. *Comput. Ling.* 40, 4 (2014), 837–881. DOI: https://doi.org/10.1162/COLI_a_00202
[76] William M. Rand. 1971. Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.* 66, 336 (1971), 846–850.
[77] Matteo Riondato, David García-Soriano, and Francesco Bonchi. 2017. Graph summarization with quality guarantees. *Data Min. Knowl. Discov.* 31, 2 (2017), 314–349. DOI: https://doi.org/10.1007/s10618-016-0468-8
[78] Matteo Riondato and Evgenios M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Min. Knowl. Discov.* 30, 2 (2016), 438–475. DOI: https://doi.org/10.1007/s10618-015-0423-0
[79] Dominique Ritze, Oliver Lehmberg, Yaser Oulabi, and Christian Bizer. 2016. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *World Wide Web Conference*. ACM, 251–261.
[80] Giulio Rossetti, Letizia Milli, and Rémy Cazabet. 2019. CDLIB: A Python library to extract, compare and evaluate communities from complex networks. *Appl. Netw. Sci.* 4, 1 (2019), 52:1–52:26. DOI: https://doi.org/10.1007/s41109-019-0165-9
[81] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *CoRR* abs/1910.01108 (2019).
[82] Alkis Simitsis, Georgia Koutrika, and Yannis E. Ioannidis. 2008. Précis: From unstructured keywords as queries to structured databases as answers. *VLDB J.* 17, 1 (2008), 117–149.
[83] Giovanni Simonini, Sonia Bergamaschi, and H. V. Jagadish. 2016. BLAST: A loosely schema-aware meta-blocking approach for entity resolution. *PVLDB* 9, 12 (2016), 1173–1184. DOI: https://doi.org/10.14778/2994509.2994533
[84] Bradley Skaggs and Lise Getoor. 2014. Topic modeling for Wikipedia link disambiguation. *ACM Trans. Inf. Syst.* 32, 3 (2014), 10:1–10:24.
[85] Neil R. Smalheiser and Vetle I. Torvik. 2009. Author name disambiguation. *Ann. Rev. Inf. Sci. Technol.* 43, 1 (2009), 1–43. DOI: https://doi.org/10.1002/aris.2009.1440430113

[86] Dezhao Song, Frank Schilder, Charese Smiley, Chris Brew, Tom Zielund, Hiroko Bretz, Robert Martin, Chris Dale, John Duprey, Tim Miller, and Johanna Harrison. 2015. TR discover: A natural language interface for querying and analyzing interlinked datasets. In *International Semantic Web Conference (Lecture Notes in Computer Science*, Vol. 9367). Springer, 21–37.

[87] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2016. NetworKit: A tool suite for large-scale complex network analysis. *Netw. Sci.* 4, 4 (2016), 508–530. DOI: https://doi.org/10.1017/nws.2016.20

[88] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *World Wide Web Conference*. 697–706. DOI: https://doi.org/10.1145/1242572.1242667

[89] Kunihiro Takeoka, Masafumi Oyamada, Shinji Nakadai, and Takeshi Okadome. 2019. Meimei: An efficient probabilistic approach for semantically annotating tables. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 281–288. DOI: https://doi.org/10.1609/aaai.v33i01.3301281

[90] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 7567–7578.

[91] Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of "small-world" networks. *Nature* 393, 6684 (1998), 440–442.

[92] Jierui Xie, Stephen Kelley, and Boleslaw K. Szymanski. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.* 45, 4 (2013), 43:1–43:35. DOI: https://doi.org/10.1145/2501654.2501657

[93] Vikas Yadav and Steven Bethard. 2018. A survey on recent advances in named entity recognition from deep learning models. In *International Conference on Computational Linguistics*. ACL, 2145–2158. Retrieved from https://www.aclweb.org/anthology/C18-1182/

[94] Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *6th ACM International Conference on Web Search and Data Mining*, Stefano Leonardi, Alessandro Panconesi, Paolo Ferragina, and Aristides Gionis (Eds.). ACM, 587–596. DOI: https://doi.org/10.1145/2433396.2433471

[95] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. *CoRR* abs/1906.08237 (2019).

[96] Fanghua Ye, Chuan Chen, and Zibin Zheng. 2018. Deep autoencoder-like nonnegative matrix factorization for community detection. In *27th ACM International Conference on Information and Knowledge Management*, Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang (Eds.). ACM, 1393–1402. DOI: https://doi.org/10.1145/3269206.3271697

[97] Mohamed Amir Yosef, Johannes Hoffart, Ilaria Bordino, Marc Spaniol, and Gerhard Weikum. 2011. AIDA: An online tool for accurate disambiguation of named entities in text and tables. *PVLDB* 4, 12 (2011), 1450–1453.

[98] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 3911–3921.

[99] Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. 2009. From keywords to semantic queries—Incremental query construction on the semantic web. *J. Web Semant.* 7, 3 (2009), 166–176.

[100] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. SATO: Contextual semantic type detection in tables. *PVLDB* 13, 11 (2020), 1835–1848. Retrieved from http://www.vldb.org/pvldb/vol13/p1835-zhang.pdf

[101] Ning Zhang, Yuanyuan Tian, and Jignesh M. Patel. 2010. Discovery-driven graph summarization. In *26th International Conference on Data Engineering*. IEEE Computer Society, 880–891. DOI: https://doi.org/10.1109/ICDE.2010.5447830

[102] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR* abs/1709.00103 (2017).

[103] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap set similarity search for finding joinable tables in data lakes. In *International Conference on Management of Data (SIGMOD'19)*. ACM, 847–864. DOI: https://doi.org/10.1145/3299869.3300065

[104] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH ensemble: Internet-scale domain search. *PVLDB* 9, 12 (2016), 1185–1196. Retrieved from http://www.vldb.org/pvldb/vol9/p1185-zhu.pdf

Received 25 September 2021; revised 15 November 2022; accepted 2 July 2023