Fantastic Tables and Where to Find Them: Table Search in Semantic Data Lakes

Martin Pekár Christensen
Aalborg University
mpch@cs.aau.dk

Aristotelis Leventidis
Northeastern University
leventidis.a@northeastern.edu

Matteo Lissandrini
University of Verona
matteo.lissandrini@univr.it

Laura Di Rocco
Northeastern University
la.dirocco@northeastern.edu

Renée J. Miller
Northeastern University
miller@northeastern.edu

Katja Hose
Technische Universität Wien
katja.hose@tuwien.ac.at

ABSTRACT

In data lakes, one of the core challenges remains finding relevant tables. We introduce the notion of *semantic data lakes*, i.e., repositories where datasets are linked to concepts and entities described in a knowledge graph (KG). We formalize the problem of *semantic table search*, i.e., retrieving tables containing information semantically related to a given set of entities, and provide the first formal definition of semantic relatedness of a dataset to tuples of entities. Our solution offers the first general framework to compute the *semantic relevance* of the contents of a table w.r.t. entity tuples, as well as efficient algorithms (exploiting semantic signals, such as entity types and embeddings) to scale the semantic search to repositories with hundreds of thousands of distinct tables. Our extensive experiments on both real-world and synthetic benchmarks show that our approach is able to retrieve more relevant tables (up to 5.4 times higher recall) in comparison to existing methods while ensuring fast response times (up to 17 times faster with LSH).

1 INTRODUCTION

Data lakes are the state-of-the-art technology to collect heterogeneous datasets in large organizations in a flexible manner [48]. However, the high level of complexity and diversity in terms of data formats, schemas, and contents poses many challenges [16, 59][1]. This includes data discovery as used, for instance, within data science workflows, where the task is to find the "right" data to solve a given data science problem [15, 46].

For example, data scientists looking into analyzing a particular phenomenon frequently need to identify all tables containing relevant data. This requires focusing specifically on entity-centric search, i.e., user queries that are composed of example entities of interest [43, 47]. To support these and similar use cases, we need to offer data discovery solutions that are able to track information about entities, their relationships, and concepts of interest across multiple tables and across the entire data lake. To tackle this problem of finding the most relevant tables for a given set of entities of interest, one can use (open or enterprise) knowledge graphs (KGs) [50, 59, 60] as a key technology for modeling entities, relationships, and their occurrences in different datasets – Figure 1a shows a simple example.

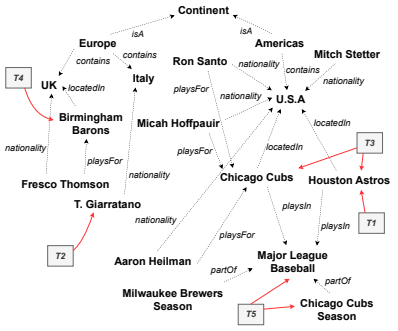
[1]While data lakes can contain more than tables, e.g., JSON files, in this work, we focus specifically on table repositories with no predefined schemas.

Semantic relevance search consists of ranking a set of tables by semantic relevance given an example set of tuples containing entities as input. This is different from keyword search, i.e., where tables are retrieved based only on the syntactic presence of specific keywords. Specifically, a data discovery system should be able to complement exact (keyword) matching by also returning tables that are semantically relevant to the query without necessarily containing any exact matches. This is also different from query-by-table approaches that attempt to find relevant or unionable/joinable tables with a possibly large query table [49, 58, 71]. Perhaps more relevant are query-by-example (QBE) or query-by-target approaches which do not require large input tables, rather only a few example tuples [9, 22, 31, 52, 65, 68, 69]. These approaches do not leverage the full knowledge graph as we propose to do. Representation learning-based table search methods [61] are also not suitable as they are not entity centric and thus are limited in their understanding of the semantics of the entities in a table. Figure 2 illustrates the relationships between the output of all existing methods, and shows that tuple search by semantic relevance is a generalization that includes subsets of other table search task outputs.

Example 1.1. Consider a data lake $\mathcal{D}$ storing tables $\{T1, T2, T3, T4, T5\}$ (Figure 1b) and their links to a KG (Figure 1a). Assume a betting company is analyzing baseball teams and players to cross-reference their performance. Given some baseball teams of interest, an initial query would express an interest in baseball players from these baseball teams, as in Figure 1c, to retrieve tables to cross-reference their results. A search engine over $\mathcal{D}$ then retrieves and ranks tables by semantic relevance that record similar data w.r.t. the query. These tables describe baseball teams or players, as well as player transfers between teams and results in different games as context. The engine should also recognize when information is less relevant or likely irrelevant, e.g., a list of teams and player names but from different sports, even if the teams are from the same cities as those in the query, is less relevant. Figure 1b contains tables relevant to the query in Figure 1c. Note that in exact matching, only tables containing keyword matches are returned ($T3$, $T4$, and $T5$). Furthermore, note that the tables may not be unionable or joinable with the query.

Despite the importance of this data discovery task, existing solutions for finding tables are mostly content-based [39, 67, 71] or metadata-based [10, 28, 32, 68] (see Section 3 for more details).

In this paper, we propose a new way of searching tables in data lakes: Our approach leverages *knowledge graphs*, which are now ubiquitous, especially within companies in the form of Enterprise Knowledge graphs (EKG) [35, 50, 59, 60]. We focus on *semantic data lakes*, which augment a data lake with links between the contents of each dataset and the concepts in a KG. Specifically, we identify entity mentions in the datasets



(a) Semantic data lake with tables linked to KG entities

T1

Date	Opponent	Score	Win
April 6	Astros	4-2	Zambrano
April 7	Astros	3-1	Cotts
April 8	Astros	4-1	Franco

T2

Player	G	AB	R
Tony Giarratano	6	18	0
Jason Bartlett	16	59	11
Pat Burrell	16	53	4

T3

Team	BAL	AST	CWS
Baltimore	2-16	5-4	2-5
Astros	16-2	4-4	7-2
Chicago	4-5	4-4	8-10

T4

Level	Team	League	Player
AA	Los Angeles Angels	Pacific Coast League	Truck Hannah
A1	Birmingham Barons	Southern Association	Fresco Thompson
B	Portsmouth Cubs	Piedmont League	Dick Luckey

T5

Season	League	Winner
2008	Major League baseball	Chicago Cubs
2009	Pacific Coast League	Memphis Redbirds
2010	Pacific Coast League	Tacoma Rainiers

(b) A data lake sample of baseball players, teams, and results

Query

Milwaukee Brewers season	Mitch Stetter	Aaron Heilman
Chicago Cubs season	Micah Hoffpauir	Ron Santo

(c) Query to retrieve baseball players in different seasons

Figure 1: Semantic data lake

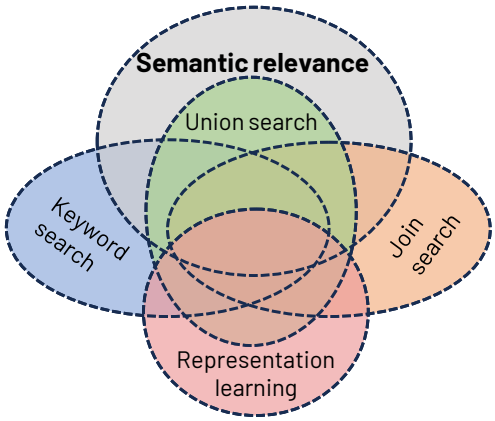


Figure 2: Relationship between table search methods' output

and link only those mentions to the entities in the KG. *Hence, our approach considers KGs with only partial coverage over a data lake, and we also show that we can retrieve relevant results even with low entity linking coverage in a purely content-based manner, making it applicable in data lakes with incomplete or inconsistent metadata.* Furthermore, our approach is useful in data discovery and exploration, where the user, for example, wants to figure out what data is available, even with small query tables as input.

Contributions. We formalize the components of a *semantic data lake* and the task of *table search* in semantic data lakes (Section 2), and discuss the gap in the state of the art (Section 3). We describe a system called Thetis having the following contributions:

- To address the data discovery problem in semantic data lakes, we introduce the first formalization of the table search task as an example-driven query paradigm and we propose a complete axiomatization of the *semantic table relevance score* to derive a principled content-based dataset discovery solution (Section 4);
- We present a search framework to perform semantic table search by proposing two concrete instantiations of a relevance score, exploiting both taxonomic information as well as learned semantic similarities as entity embeddings (Section 5);
- To handle scalability, we present a suite of optimizations to our table search algorithm that exploits variations of locality-sensitive hashing (LSH) to pre-filter irrelevant tables in the search space (Section 6);
- We evaluate our semantic search algorithm against a novel, real-world benchmark [40] of two corpora of Wikipedia tables [6, 8] to demonstrate the ability of our approach to support information discovery tasks that go beyond traditional search methods (Section 7). Furthermore, we evaluate the scalability of our algorithm over GitTables [33] containing 864,478 tables, two benchmarks created from Wikipedia Tables [40], and a synthetic corpus of more than 1.7M tables. We show that our semantic table search algorithm efficiently finds *relevant* tables that *complement* keyword-based search.

2 PROBLEM FORMULATION

We present three core concepts: data lakes, knowledge graphs, and semantic data lakes, i.e., data lakes linked to a reference knowledge graph. Furthermore, we define semantic table search.

2.1 Data Lake

We define a data lake $\mathcal{D}$ as a set of data files each in the form of a table, i.e., $\mathcal{D}=\{T_1, T_2, ..., T_n\}$ (see Figure 1b). Each table $T_i \in \mathcal{D}$ contains a set of tuples $T_i=\{t_1, t_2, ..., t_k\}$ (presented as rows) sharing the same schema, i.e., described by the same set of attributes $\mathcal{A}_i$.

Each attribute of a tuple $t_j \in T_i$ is assigned a value from an infinite countable set of values $\mathcal{V}$ (containing both numbers and strings, plus a special value $\perp$ for the null value). For instance, considering t_1 shown as the first row in table T_2 (Figure 1b), the value of the attribute Player for t_1 is "Tony Giarratano". Finally, for simplicity, a specific attribute value in a tuple is also called a cell and identified by the attribute (the column) and the tuple identifier (the row).

In data lakes, it is commonly assumed that no information is given w.r.t. referential constraint, such as foreign keys, between attributes in two distinct tables [48]. *Therefore, there is no explicit semantic connection among values contained in distinct tables [48].*

2.2 Knowledge Graphs

Knowledge Graphs (KGs) may model heterogeneous information in many domains [41, 51, 59, 60, 60]. A KG is a labeled directed graph $G = \langle \mathcal{N}, \mathcal{E}, \lambda \rangle$ [18, 42, 45, 50], where entities, concepts, and their attributes are modeled as nodes $\mathcal{N}$, and the relationships among them are modeled as edges $\mathcal{E}$. Moreover, nodes and edges are usually annotated with a set of labels, i.e., human readable literals $\mathcal{L}$ by a mapping function $\lambda : \mathcal{N} \cup \mathcal{E} \mapsto \mathcal{L}$.

Note that a KG describes in a unified model both the entities of interest (e.g., people) along with relevant attributes (e.g., age), as well as their categorizations (e.g., types of profession, also called a

taxonomy). This allows for the modeling of complex connections between many different types, entities, and concepts, e.g., a multi-step path connecting a baseball player from one baseball team to another player in another team.

2.3 Semantic Data Lake

There have been some proposals to integrate the tables in a data lake to match the structure and content of reference KGs [3, 20, 23, 59]. Unfortunately, these solutions require both entity and schema alignment, i.e., a reliable mapping between the schema of the KG and the schema of each table in the data lake. To date, existing solutions to perform this alignment automatically are still in their infancy [67], and thus obtaining a semantic data lake that applies this type of mapping requires a substantial manual effort [60]. On the other hand, entity linking, i.e., the task of matching entity mentions in tables to entities in KGs, is an easier task with multiple effective automatic solutions [6, 41, 55, 64]. Hence, to allow effective data discovery, without the need for costly and error-prone schema alignment techniques, we propose to exploit only entity linking between values in tables and entities in KGs, e.g., values like Tony Giarratano in $T2$ is linked to entity T. Giarratano in a KG.

The resulting integration of a data lake via a KG by only entity linking constitutes a novel definition of a *Semantic Data Lake*:

Definition 2.1 (Semantic Data Lake). A Semantic Data Lake is defined as (1) a data lake repository $\mathcal{D}=\{T_1, T_2, ..., T_n\}$, (2) a knowledge graph $G = \langle \mathcal{N}, \mathcal{E}, \lambda \rangle$, and a partial mapping function $\Phi : \mathcal{D}\times\mathcal{A}\times\mathcal{V}\mapsto\mathcal{N}$ and its inverse to the power set of data lake values $\Phi^{-1} : \mathcal{N}\mapsto\mathcal{P}(\mathcal{D}\times\mathcal{A}\times\mathcal{V})$.

In particular, we note that the two mapping functions map only some of the values and entities in the KG and the data lake tables. This allows our system to be robust and flexible even in the case that the KG does not describe all the information contained in the data lake, e.g., the KG may not contain some entities that appear in some of the tables. This is particularly important since requiring complete and exact mapping would greatly hinder the ingestion of new tables in a dynamic data lake.

2.4 Problem Definition

We define semantic table search as the task of retrieving a ranked list of tables that are semantically relevant to a given input query. We assume as input a query consisting of a set of entity tuples Q:$\{t_1, t_2, \ldots, t_k\}$ where each entity tuple t_i is a list of entities from G (the KG) of the form $\langle e_1, e_2, \ldots, e_n \rangle$, with $e_i \in \mathcal{N}$. Query entities not in the KG are ignored. The *semantic relevance score* ($\textsc{SemRel}_G(Q, T) \in [0, 1]$), given a user query Q, assigns a relevance score to each table T in the data lake depending on the relatedness (computed within G, between the entities in the query and the entities in the table). For instance, such a function should be able to provide a higher relevance score to tables describing baseball players than to tables describing volleyball players when comparing them to the query entity Ron Santo (Figure 1b).

Problem 2.2 (Semantic Table Search). *Given a semantic data lake $\langle \mathcal{D}, G, \Phi \rangle$, and a set of entity tuples Q as input, extract from $\mathcal{D}$ a ranked list of relevant tables describing entities semantically related to Q according to their semantic relevance score $\textsc{SemRel}_G(Q, T)$, s.t., $\forall T \in \mathcal{D}$: $\textsc{SemRel}_G(Q, T) > 0$.*

Note that, given a query Q and a table T, if the aggregate relevance score $\textsc{SemRel}(Q, T)$ (in the following, SemRel for simplicity) is zero, then we conclude the specific table is irrelevant and should not be returned. It is possible that returning all tables where $\textsc{SemRel}(Q, T) > 0$ may result in a set that is too large. Hence, without loss of generality, we return the top-k tables according to SemRel. More specifically, we tackle the challenge of defining an appropriate SemRel relevance score and devising efficient algorithms to establish tables with the highest semantic relevance score without computing scores for all tables in the data lake at runtime.

3 RELATED WORK

We consider the literature on exploratory settings where an analyst needs to retrieve datasets potentially relevant to their project from a data lake. Here, we focus on example-based approaches, in which the user circumvents query languages using examples of the information they require as input [43]. We distinguish between (a) *matching* where the content of the target datasets matches the content of the query; (b) *augmenting*, where the target datasets provide additional attributes for the query tuples or additional tuples with the same semantics; and (c) *related* where the target data provides additional contextual information w.r.t. the query. Among existing data discovery systems, we *identify semantic data lakes as an unexplored area*, and scalable semantic-aware search has not yet received sufficient attention.

3.1 Table Search in Data Lakes

Often, table search methods are based on standard information retrieval techniques. The most common approach exploits keyword search [4, 10, 12, 13, 56], where the user query is matched against captions, file names, and metadata annotations, an approach also used in the Google Dataset Search portal [4, 10]. *Unfortunately, relying on high-quality descriptive metadata represents a restrictive assumption.* Other approaches perform query-by-table, where an example table is given as the input query [9, 14, 17, 49, 52, 53, 57, 71]. With example tables as queries, the table search task becomes a matter of *relevance* that may be expressed in terms of (semantic) overlap in their contents. For instance, approaches like LSH Ensemble [71], JOSIE [70], MATE [24], and BareTQL [52] measure the overlap between column values as one of many signals, where higher overlap signifies a higher relevance.

A related task is dataset augmentation, which aims to add features, labels, or instances to a dataset [69]. ARDA [17] is a proposed framework to evaluate the quality of the information obtained through augmentation. Hence, given a specific predictive model, it takes a dataset and a data repository as input and outputs an augmented dataset such that training the predictive model on this augmented dataset results in improved performance. However, *these approaches focus mainly on data lake values* and only tangentially take into account external information w.r.t. the semantics of the values mostly in terms of data types.

To account for heterogeneity in the representation of values, *many approaches [27, 36, 49, 52] go beyond simple value overlap* and relevance between tables using KG mappings and value, tuple, or column embeddings. Some table search approaches [36, 49] also use some taxonomic information (WebIsA, YAGO, and Freebase types) and KG properties as reference knowledge to determine the relatedness of two sets of entities appearing in table columns or the relatedness of two relationships (pair of columns). For web tables, some machine learning approaches [19, 67, 68] propose to exploit contextual information extracted from the web pages in which they appear, e.g., text and heading in the page to

Table 1: Dataset search systems features: in terms of fulfilled (✔), partial (*), and missing (✗) properties to support semantic table search

		Output Tables			Searching technique		
Method	**Query Type**	**Augmenting**	**Matching**	**Related**	**Metadata**	**Content**	**Semantic**
Aurum [27]	Keyword	✔	✔	✔	✔	✔	(*)
Auctus [13]	Keyword	✔	✔	✗	✔	✗	✗
BM25 [56]	Keyword	✗	✔	✗	(*)	✔	✗
Google Dataset Search [4, 10]	Keyword	✗	✔	✗	✔	✗	✗
OCTOPUS [12]	Keyword	✗	✔	✗	✔	✔	✗
ARDA [17]	Generic table	✔	✗	✗	✗	✔	✗
BareTQL [52]	Generic table	✔	✔	✔	✔	✔	✗
D^3L [9]	Generic table	✔	✔	(*)	✔	✔	(*)
DICE [27, 53]	Generic table	✔	✔	✔	✔	✔	✗
JOSIE [70]	Generic table	✔	✔	✗	✗	✔	✗
JUNEAU [69]	Generic table	✔	✗	(*)	✔	✔	✗
MATE [24]	Generic table	✔	✗	✔	✗	✔	✗
Proximity [2]	Generic table	✗	✔	✗	✔	✔	✗
QCR [57]	Generic table	✔	✔	✗	✗	✔	✗
SANTOS [36]	Generic table	✔	✔	(*)	✗	✔	(*)
SEMPROP [14]	Generic table	✔	✔	✔	✔	✔	(*)
Starmie [25]	Generic table	✔	✔	(*)	✗	✔	(*)
Table Union [49]	Generic table	✔	✔	✗	✗	✔	✗
DS4DM/RapidMiner [30, 37, 38]	Entity table	✔	✔	✗	✔	✔	✗
S3D [29]	Entity table	✔	✔	✗	✔	✔	(*)
TURL [19]	Generic table	✔	✗	✗	✗	✔	✗
Thetis (ours)	Entity Tuples	✔	✔	✔	✗	✔	✔

obtain a better high-level understanding of the contents of a table. These approaches heavily rely on this contextual information and on training complex machine learning models. Moreover, some require the query to be compared to all instances in the repository, making them inapplicable in a scalable data lake system.

TURL [19] is a representation learning approach designed for table understanding through vectorized input queries and data lake tables. It was not directly designed for table search, but it can be employed for this task by computing vector similarities. However, tables must be large enough to achieve high-quality vector representations, limiting the effectiveness of small queries.

Many systems aim to find tables that are joinable with a query table [9, 12, 24, 57, 70, 71]. This task requires ranking candidate tables based on the syntactic overlap in the content of one or more columns or rows but does not account for any notion of semantic similarity or topical relevance. Others do table union search [9, 25, 36, 49] which does not necessarily require any syntactic overlap and can exploit semantic similarity in the data or in some cases the metadata [9] to establish when two columns describe the same domain of values. However, for table union search, the ranking is designed to favor tables that are more structurally similar having more columns and more relationships (or context) that are shared with the query table. In semantic search, however, we are not only interested in tables that union with our query table (where the goal is to find additional tuples that expand a query table by matching their schema). Hence, our semantic relevance considers the strength of the semantic similarity rather than structural similarity (having more columns or relationships in common). Another approach for related table search also exploits contextual data derived by static analysis of the usage of specific tables in different programs (e.g., python notebooks) [69]. Hence, such approaches point towards *the importance of integrating contextual information in the search task, but are limited in the type of contextual information they adopt.*

To best exploit the taxonomic information present in KGs, some approaches explicitly assume that the data lake contains only tables describing a set of entities (*entity tables*), so that one column contains the entity identifiers and the remaining columns are all interpreted as attributes of those entities [58]. Using this paradigm, systems like RapidMiner [30, 37, 38] and S3D [29] can first perform a form of schema matching between the columns of the table and the schema of the KG, and then use this information to propose set of attributes that can explain joins with other tables. *Yet, these methods either do not fully exploit these semantic resources, e.g., the structure and connectivity of the KGs, or limit themselves to explicit one-hop-matching with attributes expressed in the table.*

In our work, we establish a new entity-centric semantic relatedness measure for generic entity tuples that do not require matching any structural condition between tables (that they be joinable or unionable). We summarize the features of existing approaches in Table 1. Our method is the only one that accepts generic *entity tuples* that considers both *content* and *semantic relationships* to produce a *relevance-based ranking of tables* that goes beyond *structural matching* retrieving also tables that contain *related* entities without the need for any schema matching or accurate metadata.

3.2 Semantic Data Lakes

Thanks to their flexibility and expressiveness, KGs have been adopted in many organizations [35, 50]. A recent trend is that of exploiting KGs in data lakes [3, 5, 23, 32, 44, 59]. This leads to sophisticated approaches where the schema and contents of the datasets in a data lake are linked to the entities and relationships contained in a knowledge graph. Hence, in these approaches, every piece of information is virtually mapped to statements in the KG. This allows heterogeneous data to be accessed uniformly through semantic queries (usually SPARQL) via schema mapping and integration. While this approach achieves a high level of data

integration, it requires extensive manual mapping as well as data cleaning efforts [60], which is in contrast with the principle that a data lake should allow effortless addition of new datasets.

In contrast, our proposal is more flexible. We propose to identify within each data lake dataset all mentions of entities from a reference KG, and hence identify which pieces of the data contain those mentions. This linking is performed automatically [6, 55]. Our approach does not require mapping the complete dataset schema to KG relationships. Thus, we integrate a data lake with a KG to obtain a *Semantic Data Lake* without manual curation.

Contrary to approaches like S3D [29], we allow for a wide range of semantic similarities between entities, and we integrate them transparently in a principled scoring function, while still providing a fast search algorithm. This also sets apart our proposal from approaches for web tables, like STR [66], where embeddings of tables are aggregated by complex machine learning models requiring the computation of many-to-many matches in a brute-force fashion.

3.3 Graph Relevance

Our system infers table relevance from a given entity relevance metric provided, therefore it is explicitly designed to accommodate any relevance measures that takes advantage of any information encoded in the KG. There are many approaches that compute the relevance within a graph depending on the task. Here, we focus on semantic similarity metrics that can be converted to vector operators, allowing for fast indexing and search. Hence, instantiations of relevance measures can exploit entity attributes, such as the sets of entity types and predicates, or vector representation of entity embeddings.

We focus on the similarities that are most widely adopted: similarity of types and learned entity similarity. The former ranks entities that share similar types higher, the second extracts a learned vector representation of all entities so that entities can be compared in a learned high-dimensional space, also called an entity embedding. Thus, we compare entities based on the Jaccard similarity of their types [63] as well as the cosine similarity of their embeddings [54]. In this paper, we experiment with Jaccard of entity types and RDF2Vec embeddings constructed on the entire KG structure.

4 SEMANTIC TABLE SEARCH

We formalize the semantic relevance of a table given the semantic relevance score for entities and provide an axiomatization of the properties that a semantic relevance score should satisfy.

4.1 Aggregating Semantic Relevance

The task of semantic table search relies on the definition of a semantic relevance score, $\textsc{SemRel}(Q, T)$, between a query Q and a table T. The query is composed of one or more entity tuples, and the table is composed of one or more rows containing entity mentions. Since the relevance score is measured through G, given a table row, we only consider the entity mentions in it, i.e., extracted by the mapping function Φ (see Definition 2.1). Therefore, rows in a table are also treated as entity tuples. Consequently, to define SemRel, we need to first define a *semantic relevance score* between pairs of entity tuples $t_i, t_j \in \mathcal{P}(\mathcal{N})$. Given the set of all possible tuples as the set $\mathcal{P}(\mathcal{N})$, i.e, the set of all the subsets of arbitrary size of nodes $\mathcal{N}$ from the graph G, we require the instantiation of a semantic relevance function $\textsc{SemRel}:\mathcal{P}(\mathcal{N})\times\mathcal{P}(\mathcal{N})\mapsto[0,1]$. For instance, given tuples t_1:⟨Mitch Stetter, Milwaukee Brewers⟩, t_2:⟨Ron Santo, Chicago Cubs⟩, and t_3:⟨M. Streep, Actor⟩, this function should be able to return a higher relevance score for the pair (t_1, t_2) than for the pair (t_1, t_3). Note also that, since two tuples can be of different sizes, i.e., contain a different number of entities, SemRel may be asymmetric. Nonetheless, for consistency, given the tuples t_1 and t_2, with $t_2 \subset t_1$ and $|t_2| < |t_1|$, it must always hold that $\textsc{SemRel}(t_1, t_2) \leq \textsc{SemRel}(t_2, t_1)$. As an example, given the entity tuples t_1:⟨Mitch Stetter, Milwaukee Brewers⟩ and t_2:⟨Chicago Cubs⟩, when t_1 is considered as a query, we want the relevance score to signal that t_2 only partially matches the type of information mentioned by t_1, on the other hand, when t_2 is considered as a query, t_1 may be considered a perfect match.

It follows that, to define the semantic relevance between two entity tuples t_1, t_2, SemRel needs to compare each entity in t_1 with each entity in t_2, e.g., to recognize that both tuples mention companies. Therefore, we say that the concept of semantic relevance between tuples relies on the notion of *semantic similarity* between any two entities in G. Consider for instance the simplest case where we compare two tuples each containing just one entity, i.e., $t_i : \langle e_i \rangle$ and $t_j : \langle e_j \rangle$. Here, we naturally consider that the highest relevance score is necessarily that of an entity compared to itself, while for non-identical entities, we assume that through G it is possible to determine how similar is the semantic role in the database played by two entities. Given a query tuple t_Q, a target tuple t_T is relevant if for every entity in t_Q it contains exactly that entity or an entity that is semantically similar to that. Therefore, we require the definition of the concept of *semantic similarity score* $\sigma:\mathcal{N}\times\mathcal{N}\mapsto[0,1]$, with $\sigma(e, e)=1$. The semantic similarity score between pairs of entities should satisfy all properties of a metric. In the next section (Section 4.2), we formalize the defining characteristics of such relevance score and then how SemRel relates to σ.

Finally, given the semantic relevance score defined between pairs of tuples, we define the SemRel score between a query Q and a table T as an aggregate score of the combined relevance for each tuple in the query, $t_i \in Q$, and each tuple in the target table, $t_j \in T$, this is required since we need to compute a single table score allowing ranking of the tables to satisfy Problem 2.2. While there can be different instantiations for SemRel – and more can be studied in the future – we study the final score either as the average of the score within each tuple-to-tuple comparison or as the average of the best match between query tuples and tuples in the table. We show in our experimental evaluation (Section 7) how the second interpretation leads to better results, so we use:

$$\textsc{SemRel}_{MAX}(Q, T): \frac{\sum_{t_i \in Q} \max_{t_j \in T} \textsc{SemRel}(t_i, t_j)}{|Q|} \quad (1)$$

4.2 Axiomatization of Relevance

In the previous section, we described how the problem of semantic table search is different from classical table search, e.g., used for table augmentation, and requires the definition of an appropriate relevance score between entity tuples. Here, we follow a principled approach and provide an axiomatization of the properties that such a score needs to satisfy. Based on those axioms, we design our score.

Given a tuple $t_Q:\langle e_Q^1, ..., e_Q^m \rangle$ from the input query Q (i.e., $t_Q \in Q$), and tuple $t_T:\langle e_T^1, ..., e_T^n \rangle$ from the table $T \in \mathcal{D}$ (i.e., $t_T \in T$), we define the concept of a *relevant mapping* from t_Q to t_T as

a partial injective function[2] $\mu_{t_Q,t_T}: t_Q \hookrightarrow t_T$ s.t. $\mu_{t_Q,t_T}(e_Q^i)=e_T^j$ iff $\sigma(e_Q^i, e_T^j)>0$. Therefore, we identify four cases, namely:

(1) All entities in t_Q appear (separately) in t_T, i.e., $\forall e_Q^i \in t_Q$. $\mu(e_Q^i)\equiv e_Q^i$, this is called a *total exact mapping* denoted $t_Q \overset{\text{TE}}{\approx} t_T$.
(2) Some, but not all, entities in t_Q appear in t_T, i.e., $\exists t'_Q \subset t_Q$ s.t. $\forall e_Q^i \in t'_Q$. $\mu(e_Q^i)\equiv e_Q^i$, this is called a *partial exact mapping* denoted $t_Q \overset{\text{PE}}{\approx} t_T$.
(3) For each entity $e_Q^i \in t_Q$ there exist a mapping entity $e_T^j \in t_T$, i.e., $\forall e_Q^i \in t_Q$. $\exists e_T^j \in t_T$ s.t. $\mu(e_Q^i)=e_t^j$ and $\sigma(e_Q^i, e_T^j)>0$, those are called related entities. Since μ is injective, no two entities in t_Q are mapped to the same entity in t_T, this is called a *total related mapping* denoted $t_Q \overset{\text{TR}}{\approx} t_T$.
(4) There exists a related mapping as defined above but just of a subset of the entities in t_Q, this is called a *partial related mapping* denoted $t_Q \overset{\text{PR}}{\approx} t_T$.

Consider the following tuples: t_1:⟨Mitch Stetter, Milwaukee Brewers⟩, t_2:⟨Mitch Stetter, Milwaukee Brewers, Milwaukee⟩, t_3:⟨Ron Santo, Chicago Cubs⟩, t_4:⟨Ron Santo, Chicago⟩, t_5:⟨Milwaukee⟩. The following holds: $t_1 \overset{\text{TE}}{\approx} t_2$, $t_2 \overset{\text{PE}}{\approx} t_1$, $t_1 \overset{\text{TR}}{\approx} t_3$, $t_2 \overset{\text{PR}}{\approx} t_4$, and $t_1 \overset{\text{PR}}{\approx} t_5$.

When none of the above holds for a target tuple, we say that that the target tuple t_T *is irrelevant* to the query tuple t_Q and it should not be returned. Additionally, we note that the definition of total *related mapping* holds also for the case in which some of the mapped entities in t_T are exact mappings for the entities in t_Q. That is, if there exists a mapping such that all entities in t_Q are mapped (either through an exact or related mapping) to at least one entity in t_T, then we effectively consider this case as a total related mapping (e.g., $t_4 \overset{\text{TR}}{\approx} t_2$ from above). Based on the four cases above, we identify a set of foundational axioms that the semantic relevance score should satisfy given the type of mapping between two tuples. Hence, given a query tuple t_Q and two distinct target tuples t_{T1} and t_{T2}, the following must hold:

Axiom 1. *if it holds that* $t_Q \overset{TE}{\approx} t_{T1}$ *and* $t_Q \overset{TE}{\not\approx} t_{T2}$ *then it must also hold that* $\textsc{SemRel}(t_Q, t_{T1}) > \textsc{SemRel}(t_Q, t_{T2})$*, i.e., total exact mappings are the most relevant.*

Axiom 2. *if it holds that* $t_Q \overset{PE}{\approx} t_{T1}$ *and* $t_Q \overset{PE}{\approx} t_{T2} \vee t_Q \overset{PR}{\approx} t_{T2}$ *for the same set or a subset of entities in* t_Q*, i.e.,* $dom(\mu_{T2}) \subseteq dom(\mu_{T1})$*, then it must also hold that* $\textsc{SemRel}(t_Q, t_{T1}) \geq \textsc{SemRel}(t_Q, t_{T2})$*, i.e., larger partial exact mapping are more relevant then mappings that involve only a subset of the entities.*

Axiom 3. *if it holds that* $\forall e^i \in t_Q\ \sigma(e^i, \mu_{T1}(e^i)) > \sigma(e^i, \mu_{T2}(e^i))$*, then it must also hold that* $\textsc{SemRel}(t_Q, t_{T1}) > \textsc{SemRel}(t_Q, t_{T2})$*, i.e., tuples with more related entities have a higher relevance score.*

Next, we define the SemRel relevance score that satisfies all the above axioms and an algorithm to compute such a score. Later, in Section 6, we introduce an optimized algorithm able to compute an approximate solution with performance guarantees.

5 SEMANTIC SEARCH ALGORITHM

Given the above axioms and the semantic similarity score σ, here we first describe how to produce a mapping between a query and the tuples in a table, then we describe how to effectively compute the SemRel score for a pair of tuples, and finally, we describe an exact algorithm to solve the problem of semantic table search. The algorithm is illustrated in Figure 3.

5.1 Mapping the Query Tuple to a Table

As seen above, to identify the relevance score between a query entity tuple t_Q and a target entity tuple t_T (from a target table T), the first step is that of identifying a mapping $\mu_{Q,T}$ between the two. Ideally, given the semantic similarity score σ, we define a mapping $\mu_{Q,T}$ such that the cumulative score $\sum_{e_i \in t_Q} \sigma(e_i, \mu_{Q,T}(e_i))$ is maximized. Furthermore, given that the semantic relevance will need to be aggregated over all tuples of the target table, assuming two tuples t_T^1, and t_T^2 from table T, if we map entity e_i from t_Q to entity e_j from t_T^1, corresponding to attribute $A \in \mathcal{A}$, then when comparing t_Q to t_T^2, we should also map e_i to the entity e_l corresponding to A for t_T^2. That is, we map an entity from the query to entities from the same column for all tuples in the target table. Therefore, we need a mapping to a column in a table that maximizes the total score SemRel for all entities in that column. Moreover, we must ensure that each entity in the query tuple is assigned to a different column in the target table.

Given table T with cell values organized across columns $C_1, \ldots, C_n$ and rows $R_1, \ldots, R_m$. Let t_Q denote a query tuple composed of entities $e_1, \ldots, e_k$. We define a column-relevance score between a query entity $e_i \in t_Q$, and column $C_j \in T$ as:

$$score(e_i, C_j) = \sum_{\bar{e} \in C_j} \sigma(e_i, \bar{e})$$

Since we want the final score, summed across all query entities, to be maximized, we compute the relevance score for each pair of query entity and table column to find the best mapping function $\tau: t_Q \mapsto \{C_1, ..., C_n\}$. The information about the mapping score from query entities to columns is represented in a matrix, S, as follows:

$$S = \begin{pmatrix} score(e_1, C_1) & \cdots & score(e_1, C_n) \\ \vdots & \ddots & \vdots \\ score(e_k, C_1) & \cdots & score(e_k, C_n) \end{pmatrix}$$

Our goal is to find an assignment for each query entity to a unique column such that the column relevance score is maximized under the constraint that each entity must map to a different column. The formalization of the optimization problem is the following

$$\underset{X}{\text{argmax}} \sum_{i=1}^{k} \sum_{j=1}^{n} S_{ij} X_{ij}$$

where the assignment is specified by X which is a boolean matrix, where $X_{ij}=1$ if and only if row i is assigned to column j, and where each row is assigned to exactly one column, and each column is assigned to at most one row (i.e., in the matrix X, there are exactly k ones). This assignment problem can be solved by the Hungarian Method [21]. This can be expressed as permuting the rows and columns of a cost matrix C to maximize its trace $\min_{L,R} Tr(LCR)$ where L and R are permutation matrices. The solution to the assignment problem will provide us with the mapping function $\tau_{Q,T}: t_Q \mapsto \{C_1, ..., C_n\}$. Therefore, to compute $\mu_{Q,T}$, given the query tuple $t_Q \in Q$ and target row $t_T \in T$, we map each entity $e_j \in t_Q$ to the entity $e_k \in t_T$ where $e_k = C_k(t_T)$, i.e., the entity corresponding to column C_k in t_T, with $C_k = \tau(e_j)$.

The result of $\mu_{Q,T}$ is computed independently for each query tuple. Thus, it is not required that all query tuples $t_Q \in Q$ follow

[2] We will write μ_T or μ instead of μ_{t_Q,t_T} to simplify notation.

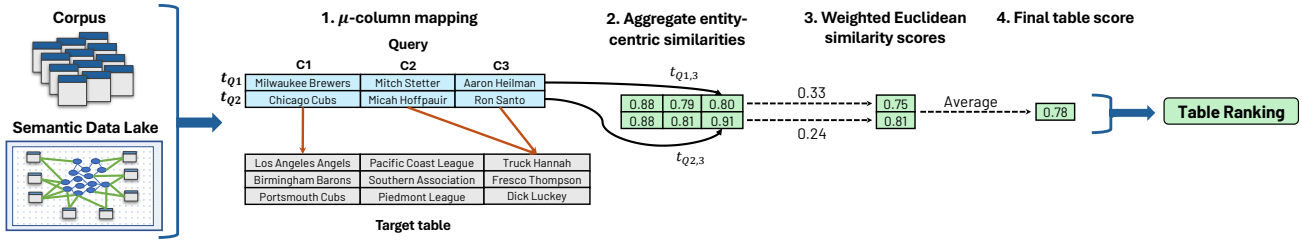


Figure 3: Semantic table search algorithm

the same schema, although this could be easily enforced. This process is depicted in Region 1 of Figure 3, though for simplicity both query entity tuples have the same column mapping.

5.2 The definition of the SemRel score

Here, we define the similarity measure that satisfies all the properties required by the SemRel score as defined above. Given the query tuple t_Q:$\langle e_Q^1, ..., e_Q^m \rangle$ (with $|t_Q|$=m) from the input query Q (i.e., $t_Q \in Q$), and the target tuple t_T:$\langle e_T^1, ..., e_T^n \rangle$ from the table $T \in \mathcal{D}$, we map the target tuple to an Euclidean space with the number of dimensions equal to the number of entities in the query tuple, i.e., $\mathbb{R}^m$. In this space, given a relevant mapping μ:$t_Q \mapsto t_T$, each target tuple t_T is mapped to a point p_T:$\langle x_1, ..., x_m \rangle$ with coordinate x_i=$\sigma(e_Q^i, \mu(e_Q^i))$. For the cases where there is no relevant mapping in t_T for an entity $e_Q^i \in t_Q$, i.e., $\mu(e_Q^i)$ is undefined, it follows that x_i=0. This is depicted in Region 2 of Figure 3. Given the mapping of each tuple in our space, we can then compute the semantic relevance of each tuple as their *euclidean distance* D from the perfect match, i.e., from the point corresponding to t_Q which will have coordinates x_i=1 $i \in [1, m]$ (Region 3 of Figure 3).

Our observation is that the most relevant tuples are those containing all the entities in the user query, i.e., those for which it holds $t_Q \overset{\text{TE}}{\approx} t_T$ (Axiom 1). These tuples will have scores $\sigma(e_Q^i, e_T^j)$ = 1 for e_T^j=$\mu(e_Q^i)$. Then, combining Axiom 2 and 3, we notice that the second best set of tuples is the set where the majority of the tuples in t_T are exact mapping of tuples in t_Q.

In the above definition, all the entities in a tuple contribute for the final relevance score in the same way. However, the entities in a query have different roles, i.e., some entities will be more important than others in determining the relevance of a given target tuple. For example, in a query like ⟨Mitch Stetter, Milwaukee Brewers⟩, we can intuitively assume that the interest of the user is primarily about baseball players, and the team where they play to a lesser degree. Therefore, a tuple containing only Milwaukee Brewers is intuitively less relevant than a tuple containing only Mitch Stetter. To cope with this observation, we propose an instantiation of SemRel able to capture the need for differentiating the discriminate power of the entities in the user query. To do so, we introduce the concept of informativeness of an entity, i.e., I:$\mathcal{N} \mapsto [0, 1]$ that automatically assigns a weight to the entities in the query based on entity frequency in the corpus. Thus, we compute SemRel as a the *weighted euclidean distance* as the following

$$\mathrm{D_I}(p_Q, p_T) = \sqrt{\sum_{i \in [1,m]} \mathrm{I}(e_Q^i)\,(1 - x_i)^2} \tag{2}$$

We convert the *weighted Euclidean distance* into a similarity score, such that a score closer to 1 means a smaller Euclidean distance.

Algorithm 1: Table Search

Input: $\mathcal{D}$=$\{T_1, T_2, ..., T_n\}$; Q:$\{tq_1, tq_2, ..., tq_m\}$
Input: I:$\mathcal{N} \mapsto [0, 1]$; σ:$\mathcal{N} \times \mathcal{N} \mapsto [0, 1]$
Output: *Table Relevance scores* $\langle T_i, \alpha_i \rangle \forall T_i \in \mathcal{D}$

```
tableScores ← [ ]
forall T ∈ D do
    qScores ← [ ]
    forall tq ∈ Q do
        columnMapping ← hungarianMapping(tq, T)
        rScores ← [ ]
        forall row ∈ T do
            eScores ← [ ]
            forall e ∈ tq do
                mappedToColumn ← columnMapping[e]
                eScores[e] ← σ(e, row[mappedToColumn])
            rScores[row] ← eScores
        aggScores ← aggRowScores(rScores)
        qScores[tq] ← 1/(sqrt(Σ_{e∈tq} I(e) (1 − aggScores[e])^2) + 1)
    tableScores [T] ← (Σ_{tq∈Q} qScores[tq])/|Q|
```

$$\mathrm{SemRel}(t_Q, t_T) = \frac{1}{\mathrm{D_I}(p_Q, p_T) + 1} \tag{3}$$

The final table score is the average of the weighted euclidean distances (Region 4 of Figure 3).

5.3 Computing Semantic Relevance

To compute the final semantic relevance score of all tables in the data lake, we follow the procedure described by Algorithm 1. The algorithm receives as input the set of tables $T_i \in \mathcal{D}$, a query Q consisting of a set of tuples $t_{Q_i} \in Q$, an entity weighing function I, and an entity semantic similarity scoring function σ. The output is then the value of SemRel for each data lake table.

In this work, we experiment with two alternative similarity functions for entities. The first is derived from the Jaccard similarity of the sets of entity types of two entities, the second is the cosine similarity between pairs of entity embeddings. For the former case, we assume two entities are similar if they share the same set of entity types. In rich KGs, it is common for entities to be annotated with multiple types at different levels of granularity, e.g., in DBpedia, `Milwaukee Brewers` is annotated both as a sports team and as an organization. Hence, we adjust the Jaccard similarity score such that we return a similarity of 1.0 when comparing an entity to itself, and otherwise, we return the Jaccard similarity between the two sets of entity types capped at 0.95. Specifically, given $\mathcal{T}_1$ as the set of entity types for entity e_i, we define our adjusted Jaccard similarity as:

$$Jaccard^*(e_1, e_2) = \begin{cases} 1 & \text{if } e_1 \equiv e_2 \\ \min(0.95, \frac{|\mathcal{T}_1 \cap \mathcal{T}_2|}{|\mathcal{T}_1 \cup \mathcal{T}_2|}) & \text{otherwise} \end{cases} \quad (4)$$

The similarity based on entity types relies on the quality and completeness of the ontology in the KG, but not on other connections with other entities. An alternative similarity function is based on embeddings. Entity embeddings, instead, are obtained through a self-supervised machine learning approach that learns a vector representation for every entity in the KG based on their higher-order connections to other entities [7, 54]. Hence, each entity is associated with a vector, and two entity vectors are similar when the two corresponding entities have similar semantic connections within the graph. In this model, given $\mathbf{V}_i$ as the entity embedding vector for e_i, the semantic similarity $\sigma(e_1, e_2)$ is computed as the cosine similarity between their vectors.

Note that our search framework and the optimization algorithm are designed to generalize also to other similarity scores based either on set similarities or vector similarities. For instance, one can also compute the similarity between two entities based on the set of predicates around them [47] or replace RDF2Vec with other entity embedding approaches designed specifically for node classification [1, 7]. We leave the exploration of alternative similarities as future work.

The algorithm then computes the relevance score for all tables as follows. For a table in the data lake, we compute a mapping $\tau: t_Q \mapsto \{C_1, ..., C_n\}$ from each query entity to a table column using the Hungarian Method, such that the summed similarity score across all query entities is maximized (line 3). Then, we generate a mapping from each query tuple to each table row (lines 7-12). Here, we compute the semantic similarity score between each entity in the query tuple $e \in t_Q$ and each entity in the target table row $e' \in t_T$. This will result in a score for every row in the table. Then, we aggregate all these scores to obtain the final score for the entire table (line 13). We consider two types of aggregation: maximal and average score. The maximal score extracts the maximal semantic similarity score among all entities mapped to the same query entity, the average score computes the average similarity instead. Given the aggregated tuple relevance score for t_Q, we compute the Euclidean distance from t_Q to the point $x_i = 1$ $i \in [1, |t_Q|]$ which is converted to a distance similarity score $0 \leq s \leq 1$, where s closer to 1 means higher similarity (line 14). As described in the previous section, this allows weighing the similarity based on the relative frequency of query entities in the entire corpus. The final table relevance score is the average of query tuple relevance scores (line 15).

The time complexity of Algorithm 1 is determined by the mapping function $\mu_{T,Q}$. Given the number of table rows R_T, the number of table columns C_T, the number of query tuples R_Q, and the number of query entities per query tuple C_Q, the time complexity is defined as $O(R_T C_T R_Q C_Q)$. However, as the query is usually small, the time complexity is in practice $O(R_T C_T)$.

6 SEMANTIC SEARCH PREFILTERING

We now present a pre-filtering technique to reduce the set of tables over which we need to compute the relevance score, given that we are usually interested in only the top-K results, i.e., the most similar tables. To achieve this, we employ a Locality-Sensitive Hashing (LSH) scheme, which groups similar items into the same bucket. Specifically, we use the set of entity types or entity embeddings as input to LSH such that similar entities are hashed into the same buckets. Then, we retrieve the subset of tables to rank based on the entities they contain. This results in our *Locality-Sensitive Entity-Index* (LSEI) for tables.

6.1 Building Locality-Sensitive Indexes

The goal is to compute signatures of each entity. LSH requires as input a vector representation for the entity to compute a reduced signature for that vector. The signature is further divided into bands. Each band is further hashed to find the corresponding bucket for similar entities, hence each distinct band corresponds to a distinct group of buckets, and within each group of buckets, an entity appears only in one of them.

LSEI for Entity Types. When comparing entities based on their types, we represent each type in the KG by a numeric index. We mimic the idea of shingling in min-hashing for documents by creating a bit vector of size $|\mathcal{T}| \times |\mathcal{T}|$ for an entity, where $|\mathcal{T}|$ is the number of types, and flip bits to 1s in positions corresponding to pairs of types, e.g., a pair of types with indices 24 and 48 have index 2448 in the bit vector. We hash this bit vector by LSH into an entity signature with a dimension equal to the number of permutation vectors. Since some types are extremely frequent, e.g., every entity has the type *owl:Thing* in DBpedia, we filter away those types that appear in more than 50% of all tables in the corpus, the idea being that a type that describes more than half of the entities cannot be really informative. This percentage is chosen based on observations from smaller experiments. Decreasing this percentage leads to a decrease in prefiltering efficacy.

LSEI for Entity Embeddings. While traditional LSH schemes for sets are based on the concept of random permutations, LSH schemes for embeddings are based on the concept of *random projections* [11]. Therefore, the signature dimension equals the number of projection vectors, each dividing the space in a positive and negative sub-space. Then, the entity signature is a bit vector, where each 1 means that the dot product between the entity embeddings vector and a given projection vector is positive.

Index Structure. An LSH index has several groups of buckets. When inserting an entity, we split the signature into multiple bands, one for each group of buckets. The size of the bands and the number of permutation/projection vectors determine the number of bucket groups, e.g., an LSH index using 32 permutation/projection vectors and a band size of 8 has 4 bands and thus 4 bucket groups. The number of buckets in each bucket group is 2^B, where B is the band size. Therefore, large band sizes result in a large number of buckets, each more likely to contain only a few entities. This corresponds to a larger search space reduction. However, a higher search space reduction also risks loss of accuracy. Finally, for each entity in the LSEI, we maintain a list of all the table identifiers in which that entity appears.

6.2 Locality-Sensitive Entity-Index Prefiltering

Before executing our table search algorithm (Algorithm 1), we search our LSH index using the entities in the input query to reduce the search space by prefiltering tables. All entities in the query are individually used to search our LSH index, and the resulting set of tables per query entity are merged into the new, reduced search space. When searching in the LSH index with a single entity, all entities in each matching bucket are merged into one set of entities, and the tables these entities are linked to are returned. Some tables may be found multiple times when merging the found buckets of entities and their linked tables. This

Table 2: Benchmark statistics: # of tables (T), mean # of rows (R), mean # of columns (C), and mean entity link coverage (Cov)

	Queries		Data Lake Tables			
	T	C	T	R	C	Cov
WT 2015	100	3.4	238,038	35.1	5.8	27.7%
WT 2019	100	2.4	457,714	23.9	6.3	18.2%
GitTables	100	3.4	864,478	142.0	12.0	29.6%
Synthetic	100	3.4	1,732,328	9.6	5.8	34.8%

gives the opportunity to further restrict the tables returned for a single entity by implementing a voting strategy. That is, we treat this intermediate result set as a bag of tables, i.e., maintaining duplicates, and count the number of occurrences of a table in the results set, so that only those tables that appear a certain amount of times are returned.

Column aggregation. An alternative approach to compute entity signatures that also saves space is by aggregating vector representations of entities in the same column into one single vector representation. When constructing this kind of LSH index using types, we merge all entity types from the entities in a single table column into one unique set of types and compute a signature of that column using this merged set of types. When using embeddings, we compute the average embeddings vector of all entities in a column.

We also note that the higher the number of query entities, the higher the number of LSH lookups, which results in a larger result set. Thus, we further optimize the LSH lookup by applying aggregation by column on the input query level in the same way as described above for tables. This introduces a further approximation with the benefit of reducing the search cost since it effectively treats queries composed of multiple entity tuples as if they were 1-tuple queries.

7 EVALUATION

We evaluate the output quality and the scalability of our semantic table search algorithm by implementing our approaches within a prototype system: Thetis. We show that keyword search can only find a limited number of tables that contain exact matches and is not adequate for discovering relevant tables without matches. Similarly, union- and join-based techniques do not address adequately this task. We show that our prefiltering techniques with LSH ensure scalability without sacrificing search quality. To this end, we compare multiple system configurations and recommend the most appropriate. We evaluate Thetis on queries of different sizes and show that, although runtime is affected by larger queries, Thetis can retrieve high-quality tables efficiently.

7.1 Experimental Setup

We compare Thetis to BM25 [56], a well-established keyword search algorithm that has been used for table search [62, 66, 68]. We further compare to state-of-the-art unionability and joinability approaches SANTOS [36], Starmie [25] and D^{3}L [9], respectively. We also compare against a deep-learning Table Representation, TURL [19], by adapting it for our task. Using TURL's pre-trained model, we aggregate all contextualized vector representations in each table to construct an embedding for each table and query. Cosine similarity between the table and query representations is used to rank the table search output. We do not compare to other methods like Aurum [27] which rely on value equality between similar attributes, something already covered by BM25, or heuristics that have already been proven less effective than D^{3}L and SANTOS. We also do not compare to SemProp [14], as it is an extension to Aurum, and its repository is outdated and non-operational. Furthermore, we also do not compare against S3D [29], as the code is not publicly available and RapidMiner [30, 37, 38] because the code is not working, and the authors could not offer any help.

All of the components of our experimental setup are available online as open-source [3]. We use a snapshot of DBpedia from 2021 as our reference KG containing ~31M nodes, ~89M edges, 763 distinct types, and 10,051 distinct predicates, however, Thetis works with any KG. Importantly, DBpedia has a range of coverage on the data lakes we consider (see below) making it particularly suitable for our evaluations. Evaluating Thetis on other public KGs is beyond the scope of this work. Nonetheless, a typical alternative would be WikiData, which showcases a rich vocabulary of entity and relationship types that is slightly richer than DBpedia's, and with which we would expect a slightly higher accuracy overall. We use RDF2Vec [54] to generate embeddings on our reference KG. However, Thetis can accommodate any set of entity embeddings. Experiments are conducted on a server with 2TB of RAM and a 64-core CPU.

Data Lake Benchmarks. We evaluate Thetis over a real-world, data discovery benchmark composed of two datasets from Wikipedia tables (WT) in Wikipedia pages (WP) [40]. We denote the two snapshots of Wikipedia tables as WT2015 and WT2019 from 2015 and 2019, respectively, which come with tuple queries consisting of various numbers of tuples (see characteristics in Table 2). In our evaluation, we have extracted a heterogeneous set of 50 1- and 5-tuples queries of width of at least 3, where the 1-tuple queries are contained in the 5-tuples queries. Note that, of the eighteen approaches mentioned in Table 1, 5 of them evaluate on less than 50K tables and 8 on less than 500K tables. Moreover, the WT benchmarks provide entity links to DBpedia and the problem of improving entity linking from tables to a KG is orthogonal to our work, where there are already numerous works tackling this problem [1, 6, 26]. These entity links can also be provided for other public KGs through `owl:sameAs` relations. The benchmarks come with ground truth rankings of tables constructed based on Wikipedia categories and navigational links. In our experiments, unless specified otherwise, we focus first on results over WT2015, which given the smaller size and the highest coverage, i.e., the number of cells linked to a KG entity, allows us to test more efficiently different settings. We use recall to evaluate the correctness of the returned result set and Normalized Discounted Cumulative Gain (NDCG) to evaluate the result set rankings. We compute recall as the number of retrieved tables that are in the top-k ground truth relevant tables according to their Jaccard similarity to the query.

To test scalability, we experiment with the GitTables dataset, since it contains 864,478 tables with an average number of columns and rows much larger than WTs (Table 2). Note that GitTables does not come with ground truth relevance annotations; hence, we cannot use GitTables to evaluate NDCG or recall. We additionally generate a synthetic dataset from WT2015 (called Synthetic in Table 2). For each table, we randomly select some rows and insert them into a new synthetic table in random order. We used this method to generate 1,494,290 new tables which we split into three corpora of different sizes: 500K, 1M, and all

[3] https://github.com/EDAO-Project/TableSearch

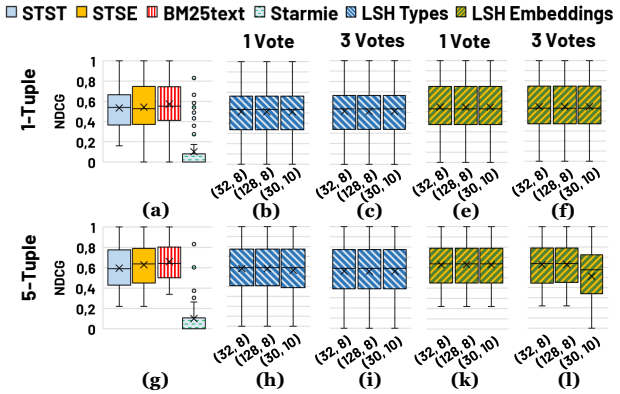


Figure 4: NDCG at top-10 on WT2015. Comparing brute-force approaches for types/embeddings (STST/STSE), different configurations of LSH prefiltering: (number of permutation/projection vectors, band size), BM25 on text queries, and Starmie union search.

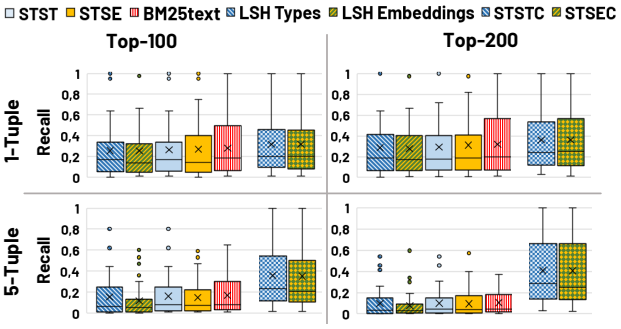


Figure 5: Recall at top-100 and top-200, including BM25 complemented with STS. STSTC/STSEC: Semantic tuple Search using Types/Embeddings Complemented with BM25.

1,494,290 tables. In each corpus, we include the original WTs, resulting in corpora of 738,038, 1,238,038, and 1,732,328 tables.

We convert the query tuples into keyword queries for BM25 which we refer to as *text* queries. We extract the entire text contents in each cell in a query and let those be keywords.

7.2 Semantic Table Search Quality

We evaluate Thetis against BM25, SANTOS, D^{3}L, and TURL on 50 1-tuple and 50 5-tuple queries on WT2015. We focus on the exact (i.e., without prefiltering) approaches: Semantic Table Search using Types (STST) and using Embeddings (STSE). We compare the NDCG scores (Figure 4) as well as the improvement in recall separately and when using both techniques jointly (Figure 5).

Ranking Quality and Recall.

Comparing NDCG scores at top-10 (Figure 4: a and g), Thetis shows a similar ranking quality, i.e., precision, to BM25 text queries on both 1- and 5-tuple queries. This is due to BM25 effectively finding the ground truth relevant tables containing exact matches with the text queries, whereas Thetis finds a different set of ground truth relevant tables that do not necessarily contain exact matches. Using a larger KG, such as WikiData, would expectedly result in a better performance of Thetis, as WikiData is more detailed and descriptive than DBpedia for many entities. Union search with Starmie achieves worse performance as relevant tables are often not unionable. The performance of SANTOS and D^{3}L as representative union and join search approaches, respectively, is even worse with NDCG scores ~1000x lower than Thetis and is therefore not plotted. Specifically, SANTOS achieves an average NDCG score of 0.0001 for both 1- and 5-tuple queries, and D^{3}L achieves 0.00006 and 0 for 1- and 5-tuple queries, respectively. This is due to these methods not being designed to take into account topical relevance, as described in Section 3.1. Therefore, these methods are not able to rank tables according to semantic relevance, and thereby they fail to properly rank relevant tables, as experiments show. However, the improved performance of Starmie over SANTOS is due to its ability to capture rich contextual semantic information within tables using trained column encoders. Similarly, TURL achieves average NDCG scores of 0.004 and 0.005 for the same queries, respectively. However, TURL's performance can reach 0.488 using entire source tables. This reflects that TURL is not designed for semantic search, but rather for table understanding.

Given the data discovery use case, it is important to evaluate the recall to see how the methods allow for retrieval of tables from the long tail. Thus, we compute top-100 and top-200 results (Figure 5) and measure the recall achieved by the methods. Here, we see once again similar results across all methods. BM25 achieves a higher third quartile in NDCG scores, but the median and mean are similar to our approach. It is important to note that the results for 5-tuple queries have lower recall compared to those for 1-tuple queries despite the increased informativeness in 5-tuple queries. This is due to the 5-tuple queries becoming easily over-specialized. The performance of STST and STSE in NDCG and recall are similar. STST is beneficial when the set of entity types is sufficiently fine-grained, whereas STSE depends on the embedding quality. Moreover, STSE works well when the taxonomy is less detailed, however, the embeddings are at times not able to distinguish entities of the same type or domain, e.g., two countries or a country and its capital.

Despite the similar performance in NDCG and recall, *the two approaches find remarkably different subsets of relevant tables.* Specifically, we compute the difference between the top-100 returned tables by BM25 and Thetis. The median size of the results set difference is 66 and 80 tables for STST on 1- and 5-tuples queries, respectively. The median is 100 for STSE on both 1- and 5-tuple queries. Hence, our semantic table search algorithm finds a disjoint set of tables from BM25. We therefore study the benefit of a Semantic Data Lake when combining Semantic Table Search and BM25. We extracted the top 50% from each method, merged the two result sets, and measured recall at top-100 and 200. We refer to these results as Semantic Table Search using Types/Embeddings Complemented with BM25 (STSTC/STSEC). Shown in Figure 5, STSTC/STSEC achieve a much higher recall. Specifically, for top-100 on 1-tuple queries, the median recall is improved by 9.1% compared to BM25text for both STSTC/STSEC. On 5-tuple queries, recall is improved by 187.8% and 156.6% compared to BM25text for STSTC/STSEC, respectively. For top-200, recall improves by 18.5% and 25.0% on 1-tuple queries and by 536.9% and 459.5% on 5-tuple queries for STSTC/STSEC, respectively. Hence, complementing tables found by exact matching with tables found by semantic relevance combines the best of both worlds. There are many other methods to complement the two approaches, such as using learning to rank, but we leave this as future work. Furthermore, exact matching over metadata can also be incorporated as a third signal, but only when metadata is informative and consistent between tables.

Aggregating row scores. As mentioned in Section 5.3, in Algorithm 1 line 13, we need to aggregate the SemRel scores across

Table 3: Runtime in seconds with LSH prefiltering by each LSH configuration on 1- and 5-tuples queries and 1 and 3 votes

			1 Vote						3 Votes					
	STST	**STSE**	**T(32, 8)**	**T(128, 8)**	**T(30, 10)**	**E(32, 8)**	**E(128, 8)**	**E(30, 10)**	**T(32, 8)**	**T(128, 8)**	**T(30, 10)**	**E(32, 8)**	**E(128, 8)**	**E(30, 10)**
1-Tuple	73.0	74.6	18.3	36.9	12.5	66.6	71.9	56.5	11.3	11.3	11.1	23.0	71.2	4.4
5-Tuples	242.0	337.9	1.2	1.8	1.5	85.2	83.4	64.9	1.1	1.2	1.1	36.1	80.5	6.4

Table 4: Search space reduction with LSH prefiltering by each LSH configuration on 1- and 5-tuples queries and 1 and 3 votes

	1 Vote						3 Votes					
	T(32, 8)	**T(128, 8)**	**T(30, 10)**	**E(32, 8)**	**E(128, 8)**	**E(30, 10)**	**T(32, 8)**	**T(128, 8)**	**T(30, 10)**	**E(32, 8)**	**E(128, 8)**	**E(30, 10)**
1-Tuple	83.0%	71.1%	88.6%	12.4%	0.01%	34.9%	89.4%	89.2%	89.4%	83.8%	8.3%	98.0%
5-Tuples	82.8%	61.4%	88.9%	11.8%	0.2%	33.1%	90.2%	89.6%	90.2%	82.5%	10.2%	97.6%

rows for a given query tuple. We experiment with either picking the maximum or the average of these scores across all table rows per query tuple. Results on NDCG at top-10 (not reported due to space limitations) show that aggregation via maximum provides the best results, with up to 5x better NDCG scores on average, as it better amplifies the relevance signal from the matching tuples.

7.3 Runtime Evaluations

We evaluate runtime on a Dell R7425 with 2TB or RAM, 64 AMD 7551 cores, and a 10TB HDD.

Table scoring. We study the computation cost of the mapping function $\mu_{T,Q}$ (Section 5.3) and compare it to the total cost required to score a single table. We find that the total, average runtimes of scoring a WT2015 table using 1- and 5-tuple queries are 2.2ms and 8.6ms, respectively. Similarly, the average runtimes of scoring a GitTable on the same queries are 3.8ms and 16.6ms, respectively. On WT2015, for 1-tuple quieres, 63.7% and 58.6% of runtime is spent computing $\mu_{T,Q}$ using types and embeddings, respectively. On 5-tuple queries, these fractions are 74.5% and 67.3%. Similarly, on GitTables, 68.0% and 62.4% are spent on this computation on 1-tuple queries, and 78.1% and 75.7% on 5-tuple queries. Hence, the overall time spent for Algorithm 1 and the cost of $\mu_{T,Q}$ are limited, even when dealing with larger tables.

LSH Configuration. We compare the effect of 6 different LSH configurations when using entity types and embeddings. This will inform the selection of our best configuration regarding the tradeoff between runtime reduction and output quality. These configurations are denoted by (X, Y), where X is the number of permutation/projection vectors and Y is the band size. These LSH configurations have been selected after testing various configurations on a smaller subset of the corpus. We apply a voting threshold, which indicates the frequency an entity must appear in the result set of an LSH lookup. We compare the NDCG scores obtained with prefiltering with these configurations (Figure 4 b, c, h, i, e, f, k, and l) to our semantic table search without prefiltering (Figure 4 a and g). All LSH configurations achieve equivalent NDCG scores as our semantic table search algorithm without prefiltering. Experimenting with table column aggregation, as described in Section 6.2, did not provide any NDCG scores above those in Figure 4.

Thetis's search runtime is correlated with the number of query tuples (Table 3), as each query tuple is compared to all table rows. The runtime is also correlated to the search space reduction achieved by each LSH configuration (Table 4): higher reduction corresponds to faster response time. By comparing Thetis's runtime with LSH prefiltering with the 3 chosen LSH configurations (Table 3), we see that the (30, 10)-configuration slightly outperforms the other two configurations, as this configuration has 4 times more buckets per bucket group than the other two configurations. Therefore, each bucket contains fewer tables. Furthermore, this configuration also has the lowest number of bucket groups leading to a lower number of buckets to be merged into the final LSH result set. Finally, requiring 3 table votes induces even faster runtime without decreasing NDCG. *In summary, our experiments show that the (30, 10)-configuration is the best-performing LSH configuration on our dataset, as it provides the highest search space reduction while achieving similar NDCG scores as our semantic table search algorithm without prefiltering, although the other configurations also achieve similarly good performances.* Therefore, for the remainder of this section, we will only experiment with the (30, 10)-configuration, as this configuration also has the best performance on other datasets.

We also evaluate a naive prefiltering technique using BM25 keyword search instead. On 1-tuple queries, NDCG decreased by 18.1% compared to LSH using types and by 29.9% compared to LSH using embeddings. On 5-tuple queries, NDCG decreased by 13.3% and 13.4%, respectively. Hence, BM25 filters out many relevant tables and is not a valid prefiltering method.

7.4 Experiments on Other Datasets

Synthetic Dataset. Using 0.7M tables, the runtimes on 1-tuple queries are on average 20.7s using types and 56.2s using embeddings. These runtimes increase to 27.4s and 71.8s on 1.2M tables and 29.9s and 85.1s on 1.7M tables. The same linear runtime increase is observed using 5-tuple queries: 54.4s and 282.8s on 0.7M tables, 73.4s and 343.4s on 1.2M tables, and 77.3s and 369.4s on 1.7M tables. The linear increase in runtime for each corpus size is because the search space reduction percentage is stable on all synthetic corpora. Runtimes using types are faster than using embeddings due to LSH using types filtering out a larger fraction of the search space: on average 91% prefiltered using types and 74% using embeddings.

Experiments on WT2019. We also evaluate Thetis on the WT2019 dataset, a dataset with more tables and a significantly lower entity link coverage. Thetis achieves average NDCG scores of 0.55 using both types and embeddings on 1-tuple queries and 0.61 and 0.62 on 5-tuple queries. These scores are very similar to those presented in Figure 4, showing that the accuracy of Thetis is not greatly influence by a drop in coverage from 27.7% to 18.2% (see Table 2). Since WT2019 is a larger dataset and the search space reduction percentages are similar to those on the WT2015 dataset, the runtimes are slower compared to WT2015. The runtimes are at 26.2s and 35.1s on average using types and embeddings on 1-tuple queries, respectively, and 95.4s and 189.0s on 5-tuple queries.

Experiments on GitTables. We evaluate the runtime of Thetis on GitTables consisting of larger tables, i.e, more rows and columns (see Table 2). We do not evaluate the ranking quality on

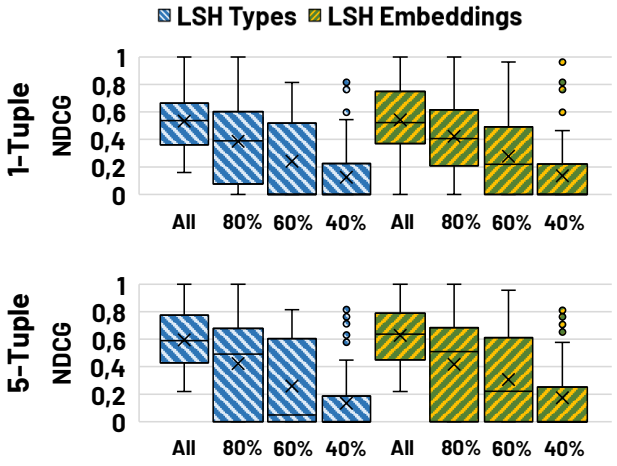


Figure 6: NDCG at top-10 when decreasing coverage

this dataset, as it does not come with ground truth. Furthermore, GitTables do not come with entity link annotations. Therefore, we construct Lucene indexes using the KG entity labels and perform keyword search to link mentions to KG entities. The resulting runtimes are 1.7s and 3.7s on 1-tuple queries and 2.2s and 14.9s on 5-tuple queries. Therefore, the runtimes on GitTables are comparable to those on the corpora containing smaller tables, i.e., WT2015 and WT2019. This is due to the LSH prefiltering reducing the GitTables corpus by more than 98% for all queries, as the table entities are more evenly distributed across the LSH buckets, and hence, the LSH lookups are more selective. Thus, Thetis effectively has to rank fewer tables than on WT2015 and WT2019.

7.5 Varying Entity Linking Coverage

Thetis is designed to exploit the contextual information offered by a KG when entities in the tables are linked to it, while it does not require a table to be fully linked. Here, we refer to *link coverage* as the percentage of linked entities among all its cells. Intuitively, the higher the coverage the higher the information we can infer about the contents of a table. To maximize the coverage, it is important that the target KG can describe all the important entities in the data lake. Nonetheless, it is very common to test open-domain KGs when trying to integrate data in data lakes [34]. In the experiments shown above, we have seen that even with less than 30% of cells being linked to KG entities on average, our approach is competitive in precision to BM25 while being *able to retrieve a large set of tables that were not retrieved otherwise.* Thus, even with partial overlap between the KG domain and the tables, our approach is effective in retrieving relevant data.

We further experiment with different levels of table link coverage (Figure 6). Specifically, we retrieve top-1000 tables and keep only tables with at most a given link coverage, e.g., only tables with link coverage up to 60%. We then evaluate NDCG on the top-10 of those retrieved tables. As expected, the tables become increasingly more difficult to retrieve as entity link coverage decreases, e.g., we see a drop in performance with less than 40% of entities linked to the KG. Yet, Thetis still retrieves relevant tables that BM25 cannot retrieve. For 40% link coverage, the median top-10 result set difference is 3 on 1-tuple queries using types and 2 using embeddings. On 5-tuple queries, these numbers are 4 and 3, respectively. Further, there are still cases where, even when not many entities are linked, the method can achieve up to 0.8 of NDCG as it can capitalize on the semantic information of the few entities linked.

We also perform an experiment (not reported in the figures due to space limitations), where we substitute the ground truth entity links in WT15 with the predicted entity links retrieved with a state-of-the-art entity linker (EMBLOOKUP [1]). Using this entity linker, the mean linking coverage is only 20.3% (compared to 27.7% on WT15) and the F1-score of the employed entity linker is only 0.21. Despite this, Thetis achieves an NDCG score of 0.14 using types and $k = 10$ and 0.20 when using embeddings for 1-tuple queries and 0.26 and 0.29 for 5-tuples queries using types and embeddings, respectively. This performance is better than the performance when decreasing the percentage of ground truth entity links to at most 40% per table in Figure 6. Note that the 20.3% is the mean coverage, whereas the 40% in Figure 6 is an upper bound. Therefore, Thetis is still able to retrieve meaningful results even when faced with poor entity linking quality. Moreover, Thetis will directly benefit from future developments in entity linking methods.

8 CONCLUSION AND FUTURE WORK

We have defined *semantic data lakes* and the *semantic table search* task for which we have presented a solution Thetis. Thetis exploits a reference KG to facilitate entity-centric, exemplar querying for semantically related tables. Thetis includes a search space pre-filtering method that uses LSH to improve the runtime by up to 17 times. Our experiments show that complementing keyword search with Thetis allows one to find more relevant tables improving recall by up to 5.4 times. In the future, we plan to explore the impact of alternative embeddings and more advanced structural graph embeddings. We will also experiment with alternative similarity metrics to improve the results for the case of over-specialized queries. We will also explore using a combination of similarity measures in Thetis, including complementing BM25 with Thetis using both types and embeddings in a unified manner. Finally, incorporating available metadata as a third signal in our relevance ranking is also a possibility to explore.

ACKNOWLEDGMENT

This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-8048-00051B, the EU's H2020 research and innovation programme under grant agreement No. 838216, and the Poul Due Jensen Fond. Additional funding from NSF awards IIS-2107248, IIS-1956096, and IIS-2325632.

REFERENCES

[1] Ghadeer Abuoda, Saravanan Thirumuruganathan, and Ashraf Aboulnaga. 2022. Accelerating Entity Lookups in Knowledge Graphs Through Embeddings. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1111–1123. https://doi.org/10.1109/ICDE53745.2022.00088

[2] Ayman Alserafi, Alberto Abelló, Oscar Romero, and Toon Calders. 2020. Keeping the Data Lake in Form: Proximity Mining for Pre-filtering Schema Matching. *ACM Transactions on Information Systems (TOIS)* (2020).

[3] Ada Bagozi, Devis Bianchini, Valeria De Antonellis, Massimiliano Garda, and Michele Melchiori. 2019. Personalised Exploration Graphs on Semantic Data Lakes. In *On the Move to Meaningful Internet Systems: OTM 2019 Conferences - Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21-25, 2019, Proceedings (Lecture Notes in Computer Science)*, Vol. 11877. Springer, 22–39. https://doi.org/10.1007/978-3-030-33246-4_2

[4] Omar Benjelloun, Shiyu Chen, and Natasha F. Noy. 2020. Google Dataset Search by the Numbers. In *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Vol. 12507. Springer, 667–682. https://doi.org/10.1007/978-3-030-62466-8_41

[5] David Bernhauer, Martin Nečaský, Petr Škoda, Jakub Klímek, and Tomáš Skopal. 2022. Open dataset discovery using context-enhanced similarity search. *Knowledge and Information Systems* (2022), 1–27.

[6] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *The Semantic Web - ISWC 2015*, Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, Krishnaprasad Thirunarayan, and Steffen Staab (Eds.). Springer International Publishing, Cham, 425–441.

[7] Russa Biswas, Jan Portisch, Heiko Paulheim, Harald Sack, and Mehwish Alam. 2022. Entity Type Prediction Leveraging Graph Walks and Entity Descriptions. In *The Semantic Web – ISWC 2022*, Ulrike Sattler, Aidan Hogan, Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d'Amato (Eds.). Springer International Publishing, Cham, 392–410.

[8] Tobias Bleifuß, Leon Bornemann, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. 2021. Structured Object Matching across Web Page Revisions. In *IEEE International Conference on Data Engineering (ICDE)*. 1284–1295.

[9] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[10] Dan Brickley, Matthew Burgess, and Natasha Noy. 2019. Google Dataset Search: Building a Search Engine for Datasets in an Open Web Ecosystem. In *The World Wide Web Conference (WWW '19)*. Association for Computing Machinery, New York, NY, USA, 1365–1375. https://doi.org/10.1145/3308558.3313685

[11] James Briggs. 2022. *Faiss: The Missing Manual*. Pinecone Systems, Inc. https://www.pinecone.io/learn/faiss/

[12] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. *PVLDB* 2, 1 (2009), 1090–1101. http://www.vldb.org/pvldb/2/vldb09-576.pdf

[13] Sonia Castelo, Rémi Rampin, Aécio Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. 2021. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *Proc. VLDB Endow.* 14, 12 (oct 2021), 2791–2794. https://doi.org/10.14778/3476311.3476346

[14] Raul Castro Fernandez, Essam Mansour, Abdulhakim A. Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 989–1000. https://doi.org/10.1109/ICDE.2018.00093

[15] Chengliang Chai, Jiayi Wang, Yuyu Luo, Zeping Niu, and Guoliang Li. 2022. Data Management for Machine Learning: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2022), 1–1. https://doi.org/10.1109/TKDE.2022.3148237

[16] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. 2020. Dataset search: a survey. *The VLDB Journal* 29, 1 (2020), 251–272.

[17] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *Proc. VLDB Endow.* 13, 9 (May 2020), 1373–1387.

[18] Dario Colazzo, François Goasdoué, Ioana Manolescu, and Alexandra Roatiş. 2014. RDF analytics: lenses over semantic graphs. In *WWW*. ACM, 467–478.

[19] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (Nov. 2020), 307–319.

[20] Henrik Dibowski and Stefan Schmid. 2021. Using Knowledge Graphs to Manage a Data Lake. *INFORMATIK 2020* (2021).

[21] Jack R. Edmonds and Richard M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* 19, 2 (1972), 248–264. https://doi.org/10.1145/321694.321699

[22] Anshul Bhandari El Kindi Rezig, Anna Fariha, Benjamin Price, Allan Vanterpool, Andrew Bowne, Lindsey McEvoy, and Vijay Gadepally. [n.d.]. Examples are All You Need: Iterative Data Discovery by Example in Data Lakes. ([n. d.]).

[23] Kemele M Endris, Philipp D Rohde, Maria-Esther Vidal, and Sören Auer. 2019. Ontario: Federated Query Processing Against a Semantic Data Lake. In *International Conference on Database and Expert Systems Applications*. Springer, 379–395.

[24] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. MATE: Multi-Attribute Table Extraction. *Proc. VLDB Endow.* 15, 8 (apr 2022), 1684–1696. https://doi.org/10.14778/3529337.3529353

[25] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Reneé J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proc. VLDB Endow.* 16, 7 (Aug. 2023), 14.

[26] Wenfei Fan, Liang Geng, Ruochun Jin, Ping Lu, Resul Tugay, and Wenyuan Yu. 2022. Linking Entities across Relations and Graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 634–647. https://doi.org/10.1109/ICDE53745.2022.00052

[27] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.

[28] Raul Castro Fernandez, Nan Tang, Mourad Ouzzani, Michael Stonebraker, and Samuel Madden. 2019. Dataset-On-Demand: Automatic View Search and Presentation for Data Discovery. *arXiv preprint arXiv:1911.11876* (2019).

[29] Sainyam Galhotra and Udayan Khurana. 2020. Semantic Search over Structured Data. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 3381–3384. https://doi.org/10.1145/3340531.3417426

[30] Anna Lisa Gentile, Sabrina Kirstein, Heiko Paulheim, and Christian Bizer. 2016. Extending rapidminer with data search and integration capabilities. In *European Semantic Web Conference*. Springer, 167–171.

[31] Yue Gong, Zhiru Zhu, Sainyam Galhotra, and Raul Castro Fernandez. 2023. Ver: View Discovery in the Wild. In *ICDE*. 503–516.

[32] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An Intelligent Data Lake System. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 2097–2100. https://doi.org/10.1145/2882903.2899389

[33] Madelon Hulsebos, Çağatay Demiralp, and Paul Groth. 2021. GitTables: A Large-Scale Corpus of Relational Tables. *arXiv preprint arXiv:2106.07258* (2021). https://arxiv.org/abs/2106.07258

[34] Ihab F. Ilyas, JP Lacerda, Yunyao Li, Umar Farooq Minhas, Ali Mousavi, Jeffrey Pound, Theodoros Rekatsinas, and Chiraag Sumanth. 2023. Growing and Serving Large Open-Domain Knowledge Graphs. In *Companion of the 2023 International Conference on Management of Data (SIGMOD '23)*. Association for Computing Machinery, New York, NY, USA, 253–259. https://doi.org/10.1145/3555041.3589672

[35] Ihab F Ilyas, Theodoros Rekatsinas, Vishnu Konda, Jeffrey Pound, Xiaoguang Qi, and Mohamed Soliman. 2022. Saga: A Platform for Continuous Construction and Serving of Knowledge At Scale. In *Proceedings of the 2022 International Conference on Management of Data*. 2259–2272.

[36] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2022. SANTOS: Relationship-based Semantic Table Union Search. In *Proceedings of the 2023 International Conference on Management of Data (SIGMOD '23)*. Association for Computing Machinery, New York, NY, USA, 15.

[37] Oliver Lehmberg and Christian Bizer. 2017. Stitching Web Tables for Improving Matching Quality. *PVLDB* 10, 11 (2017), 1502–1513.

[38] Oliver Lehmberg, Christian Bizer, and Alexander Brinkmann. 2017. WInte.r - A Web Data Integration Framework. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017 (CEUR Workshop Proceedings)*, Vol. 1963. CEUR-WS.org. http://ceur-ws.org/Vol-1963/paper506.pdf

[39] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. 2015. The Mannheim Search Join Engine. *Journal of Web Semantics* 35 (2015), 159 – 166. https://doi.org/10.1016/j.websem.2015.05.001 Semantic Web Challenge 2014.

[40] Aristotelis Leventidis, Martin Pekár Christensen, Matteo Lissandrini, Laura Di Rocco, Katja Hose, and Renée Miller. 2024. A Large Scale Test Corpus for Semantic Table Search. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 2681–2690. https://doi.org/10.1145/3626772.3657877

[41] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *PVLDB* 3, 1 (2010), 1338–1347. http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R118.pdf

[42] Matteo Lissandrini, Katja Hose, and Torben Bach Pedersen. 2023. Example-Driven Exploratory Analytics over Knowledge Graphs. In *EDBT*. OpenProceedings.org, 105–117.

[43] Matteo Lissandrini, Davide Mottin, Themis Palpanas, and Yannis Velegrakis. 2018. *Data Exploration Using Example-Based Methods*. Morgan & Claypool Publishers.

[44] Mohamed Nadjib Mami, Damien Graux, Simon Scerri, Hajira Jabeen, Sören Auer, and Jens Lehmann. 2019. Squerall: Virtual ontology-based access to heterogeneous and large data sources. In *International Semantic Web Conference*. Springer, 229–245.

[45] Frank Manola and Eric Miller (Eds.). 2004. *RDF Primer*. World Wide Web Consortium. http://www.w3.org/TR/rdf-primer/

[46] Renée J. Miller. 2018. Open Data Integration. *PVLDB* 11, 12 (2018), 2130–2139. https://doi.org/10.14778/3229863.3240491

[47] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2016. Exemplar queries: a new way of searching. *The VLDB Journal* 25 (2016), 741–765.

[48] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *PVLDB* 12, 12 (Aug. 2019), 1986–1989. https://doi.org/10.14778/3352063.3352116

[49] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825.

[50] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. 2019. Industry-scale knowledge graphs: Lessons and challenges. *ACM Queue* 17, 2 (2019), 48–75.

[51] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. 2008. Linking Data to Ontologies. In *Journal on Data Semantics X*. Springer Berlin Heidelberg, Berlin, Heidelberg, 133–173.

[52] Davood Rafiei, Harrison Fah, Thomas Lafrance, and Arash Dargahi Nobari. 2022. BareTQL: An Interactive System for Searching and Extraction of Open Data Tables. In *27th International Conference on Intelligent User Interfaces (IUI '22 Companion)*. Association for Computing Machinery, New York, NY, USA, 30–33. https://doi.org/10.1145/3490100.3516452

[53] El Kindi Rezig, Anshul Bhandari, Anna Fariha, Benjamin Price, Allan Vanterpool, Vijay Gadepally, and Michael Stonebraker. 2021. DICE: Data Discovery by Example. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2819–2822. https://doi.org/10.14778/3476311.3476353

[54] Petar Ristoski and Heiko Paulheim. 2016. Rdf2vec: Rdf graph embeddings for data mining. In *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15*. Springer, 498–514.

[55] Dominique Ritze and Christian Bizer. 2017. Matching Web Tables To DBpedia - A Feature Utility Study. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, Volker Markl, Salvatore Orlando, Bernhard Mitschang, Periklis Andritsos, Kai-Uwe Sattler, and Sebastian Breß (Eds.). OpenProceedings.org, 210–221. https://doi.org/10.5441/002/edbt.2017.20

[56] Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.

[57] Aécio Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A Sketch-based Index for Correlated Dataset Search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2928–2941. https://doi.org/10.1109/ICDE53745.2022.00264

[58] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *SIGMOD*. 817–828. https://doi.org/10.1145/2213836.2213962

[59] Stefan Schmid, Cory Henson, and Tuan Tran. 2019. Using Knowledge Graphs to Search an Enterprise Data Lake. In *European Semantic Web Conference*. Springer, 262–266.

[60] Juan F Sequeda, Willard J Briggs, Daniel P Miranker, and Wayne P Heideman. 2019. A Pay-as-you-go Methodology to Design and Build Enterprise Knowledge Graphs from Relational Databases. In *International Semantic Web Conference*. Springer, 526–545.

[61] Ibraheem Taha, Matteo Lissandrini, Alkis Simitsis, and Yannis Ioannidis. [n.d.]. A Study on Efficient Indexing for Table Search in Data Lakes. ([n. d.]).

[62] Mohamed Trabelsi, Zhiyu Chen, Shuo Zhang, Brian D. Davison, and Jeff Heflin. 2022. StruBERT: Structure-Aware BERT for Table Search and Matching. In *Proceedings of the ACM Web Conference 2022 (WWW '22)*. Association for Computing Machinery, New York, NY, USA, 442–451. https://doi.org/10.1145/3485447.3511972

[63] Xu Wang, Zhisheng Huang, and Frank van Harmelen. 2020. Evaluating Similarity Measures for Dataset Search. In *International Conference on Web Information Systems Engineering*. Springer, 38–51.

[64] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *SIGMOD*. 97–108. https://doi.org/10.1145/2213836.2213848

[65] Junwen Yang, Yeye He, and Surajit Chaudhuri. 2021. Auto-Pipeline: Synthesize Data Pipelines By-Target Using Reinforcement Learning and Search. *Proc. VLDB Endow.* 14, 11 (2021), 2563–2575.

[66] Shuo Zhang and Krisztian Balog. 2018. Ad Hoc Table Retrieval using Semantic Similarity. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien L. Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 1553–1562. https://doi.org/10.1145/3178876.3186067

[67] Shuo Zhang and Krisztian Balog. 2020. Web Table Extraction, Retrieval, and Augmentation: A Survey. *ACM Transactions on Intelligent Systems and Technology (TIST)* 11, 2 (2020), 1–35.

[68] Shuo Zhang and Krisztian Balog. 2021. Semantic Table Retrieval Using Keyword and Table Queries. *ACM Trans. Web* 15, 3, Article 11 (May 2021), 33 pages. https://doi.org/10.1145/3441690

[69] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1951–1966. https://doi.org/10.1145/3318464.3389726

[70] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 847–864. https://doi.org/10.1145/3299869.3300065

[71] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *PVLDB* 9, 12 (2016), 1185–1196. http://www.vldb.org/pvldb/vol9/p1185-zhu.pdf