SANTOS: Relationship-based Semantic Table Union Search

AAMOD KHATIWADA*, Northeastern University, USA
GRACE FAN*, Northeastern University, USA
ROEE SHRAGA, Northeastern University, USA
ZIXUAN CHEN, Northeastern University, USA
WOLFGANG GATTERBAUER, Northeastern University, USA
RENÉE J. MILLER, Northeastern University, USA
MIREK RIEDEWALD, Northeastern University, USA

Existing techniques for unionable table search define unionability using metadata (tables must have the same or similar schemas) or column-based metrics (for example, the values in a table should be drawn from the same domain). In this work, we introduce the use of semantic relationships between pairs of columns in a table to improve the accuracy of the union search. Consequently, we introduce a new notion of unionability that considers relationships between columns, together with the semantics of columns, in a principled way. To do so, we present two new methods to discover the semantic relationships between pairs of columns. The first uses an existing knowledge base (KB), and the second (which we call a "synthesized KB") uses knowledge from the data lake itself. We adopt an existing Table Union Search benchmark and present new (open) benchmarks that represent small and large real data lakes. We show that our new unionability search algorithm, called SANTOS, outperforms a state-of-the-art union search that uses a wide variety of column-based semantics, including word embeddings and regular expressions. We show empirically that our synthesized KB improves the accuracy of union search by representing relationship semantics that may not be contained in an available KB. This result hints at a promising future of creating synthesized KBs from data lakes with limited KB coverage and using them for union search.

CCS Concepts: • **Information systems → Information integration**.

Additional Key Words and Phrases: Data lakes, Table Discovery, Knowledge Graphs

1 INTRODUCTION

Table search is of growing practical relevance [7, 23, 30, 49]. It allows data scientists to find datasets needed for analysis or training of machine learning algorithms [14, 33]. Google's dataset search, for example, uses keyword search over metadata which Google incentivizes data owners to use in

---

* Khatiwada and Fan are lead authors.

Authors' addresses: Aamod Khatiwada*, Northeastern University, USA, khatiwada.a@northeastern.edu; Grace Fan*, Northeastern University, USA, fan.gr@northeastern.edu; Roee Shraga, Northeastern University, USA, r.shraga@northeastern.edu; Zixuan Chen, Northeastern University, USA, chen.zixu@northeastern.edu; Wolfgang Gatterbauer, Northeastern University, USA, w.gatterbauer@northeastern.edu; Renée J. Miller, Northeastern University, USA, miller@northeastern.edu; Mirek Riedewald, Northeastern University, USA, m.riedewald@northeastern.edu.
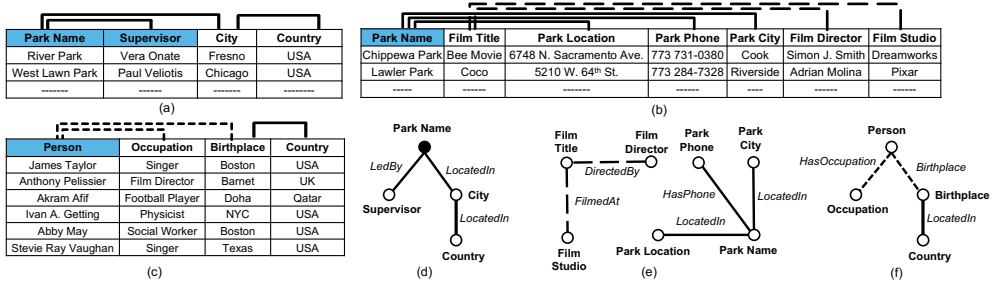

Fig. 1. (a) A table about parks. Relationships between `Park Name` and `Supervisor` (`ledBy`) and between `Park Name` and `City` (`locatedIn`) are found and depicted by the solid lines above the table. We have also found a relationship between `Country` and `City` (`locatedIn`). (b) A table about parks and films shown in the park. Some of the relationships we found are shown by solid and dashed lines above the table. (c) A table about famous people. (d) A semantic graph for Table (a) with the relationships found related to the root `Park Name`. (e), (f) Semantic graphs for Tables (b), (c), respectively.

a standard way [4]. However, when dealing with enterprise data or open data, keyword search via textual queries or over metadata is challenging because metadata may be missing, inconsistent, or incomplete [1, 10, 32, 47]. Hence, table search for those domains must rely on the data available within given tables instead of relying on well-curated metadata [7, 23].

In this work, we focus on *table union search*, which is the problem of finding tables that can union with a query table and possibly extend it with more rows. Prior work by Nargesian et al. [33] considers two tables to be *unionable* if "they share attributes from the same domain". We believe that this is a necessary, but not sufficient condition. In particular, we argue that the relationships modeled by pairs of attributes in a table should share a common semantic meaning. We illustrate this with an example.

> EXAMPLE 1. *Consider Tables (a), (b) and (c) in Fig. 1 describing (a) parks, (b) films shown in different parks, and (c) the birth places of famous people, respectively. Suppose the user supplies Query Table (a) and Tables (b), (c) reside in a data lake. The user wants to find data lake tables that union with the Query Table (a). Typically, union search has been defined to permit either the full query table or a projection of it to union with the full data lake table or a projection of it [3]. Now, if we only consider attribute unionability, Table (c) may be considered the most unionable table with three unionable attributes (*Person, Birthplace, Country*) having high attribute unionability scores with the attributes (*Supervisor, City and Country*) in the query table. This may be true if we are considering data values [33] or if we consider the attribute names and values [3, 21, 24]. However, if we consider the relationships in the tables, then we would notice that in Query Table (a),* City *is where the* Park *is located (and the person in* Supervisor *works), but in Table (c),* Birthplace *contains cities where the people in the* Person *column were born. Deciding unionability based only on attribute unionability, without considering the semantics of the relationships between attributes, can lead to false positives. Even worse, it can lead to the omission of good answers like Table (b) which has information about parks and the city in which they are located, but might be considered less unionable than (c) based solely on column semantics.*

We present a new definition of unionability based on both the attribute semantics and the *relationship semantics between attributes*. Relationships have been used to a limited extent in web table search. Their tables are often assumed to be "entity tables" (i.e., all attributes are properties of a known common entity represented by a single attribute) [7, 8, 28] or they use a subject column and a single relationship from that column to search for tables [43]. However, we do not assume

that every table has a single subject attribute that all other attributes describe, nor that we can automatically detect such an attribute. While the entity-table assumption may be mostly true in web tables, data lake tables (in both open data and enterprise lakes) tend to be much wider and may describe the interplay between more than one entity.

In this work, we provide a new definition of table union search. Given a query table Q and a set of data lake tables $\mathcal{T}$, the top-k table union search problem is to find the best k tables from $\mathcal{T}$ that can be unioned with the query table by taking the column and relationship semantics of all involved tables into account.

> EXAMPLE 2. *The columns* City *and* Country *in Table (a) are unionable with* Birthplace *and* Country *in Table (c) because the columns are unionable and their relationships are the same (*locatedIn*). If a user is interested in amassing a collection of cities and their properties like location, then these tables are unionable. However, if a user wants to collect a set of park supervisors and their properties (the city and country in which they work), then Tables (a) and (c) are not unionable. This is because the relationship between* Supervisor *and* City *(*worksIn*) is not the same as the relationship between* Person *and* Birthplace *(*bornIn*). A user may consider Table (b) to be unionable with Table (a), but only on* π(*Park Name, City, Country*)*. If we only consider attribute semantics, we might mistakenly say* Park Name, Supervisor, City *is unionable with* Park Name, Film Director, Park City*. But the relationship semantics reveal that* Supervisor *has a different relationship to* Park Name *than* Film Director *has to* Park Name*.

1.1 Contributions

Our contributions can be summarized as follows.

• *Relationship-based Semantic Table Union Search.* We present a new definition of table union search that comprehensively integrates column and relationship semantics.

• *KB Solution.* We present a solution to the relationship-based semantic table union search problem that uses a knowledge base (KB) to determine column and relationship semantics. Our solution creates semantic graphs for tables whose nodes are columns and edges are binary relationships found in the KB (depicted in Fig. 1(d)-(f) and explained in Sec. 3). We present a scoring function to match the semantic graph of a data lake table to that of a query table.

• *Synthesized KB Solution.* We present a second, novel solution to the search problem that uses the data lake itself to determine if a set of columns have the same or similar semantics. Our solution determines if the binary relationship between two columns has similar semantics with other binary relationships in the data lake. We show how this *synthesized KB* can be used to create semantic graphs and solve the table union search problem.

• *Empirical Evaluation.* Our two solutions were designed to be used in concert with the Synthesized KB compensating for a curated KB that has only partial coverage over a data lake (which is typical when using both open KBs and proprietary enterprise KBs). We show that when used together, our solutions outperform a SOTA table union baseline. We also experiment with using each solution independently. Finally, we use the Synthesized KB together with different sized samples of the curated KB to understand how accurate our solution is as the coverage of the curated KB decreases.

• *Benchmarks.* We develop (and openly share) three new relationship union benchmarks. In the first, we reuse part of the TUS benchmark [33] and label it with relationship semantics. In the second, we use real open data tables and queries, and label a groundtruth. The third is a larger data lake with open data tables and queries, where we only label the query results that either our technique or a baseline technique returns. Hence, this larger benchmark can be used to evaluate precision and efficiency, but not recall.

2 RELATED WORK

There is a rich literature on table union search that started with work on web tables. We begin by describing the state-of-the-art in table union search (Sec. 2.1), after which we discuss related work on attribute annotation (Sec. 2.2) and other related work (Sec. 2.3).

2.1 Searching for Unionable Tables

Cafarella et al. [5] uses search-style keyword queries and returns a ranked list of relevant web tables. They mainly use tf-idf score and mean string length difference to find the similarity between the columns and combine them to infer the table similarity.

Sarma et al. [7] defines the problem of finding unionable web tables as an entity complement problem. Two tables are deemed unionable if they share similar schemas and a "subject" column, which is *a single column that contains the entities the table is about*. This work assumes that each table has a single subject column, a common trait of web tables, but as we argued in Sec. 1 is a limiting assumption in data lakes and open data. Also, this approach relies solely on an existing KB.

Lehmberg et al. [20] uses attribute labels and value overlap between attributes to determine table matching. They build on work from Ling et al. [24] that relies on web tables having identical or similar schemas and uses value overlap. However, attribute labels can be ambiguous or missing in data tables [32], so we cannot rely on such metadata to be complete and consistent.

Nargesian et al. [33] considers two tables to be unionable if they have a bipartite matching between a subset of their columns. This approach generates attribute unionability scores using three different statistical tests, one of which is semantic unionability that makes use of an existing KB. Since they also look at set overlap and word embeddings, they do not rely exclusively on a KB. However, they do not consider the semantics of relationships between columns as discussed in Ex. 1. This in turn, may yield false positive results when searching for unionable tables.

Fernandez et al. [11] introduces SemProp, which links related tables by creating DAGs of table elements and external sources such as ontologies and embeddings, connected by semantic and syntactic links. Although the goals are similar, we solely leverage the data values in tables whereas SemProp mainly uses the tables' schemas and names to find related tables.

Bogatu et al. [3] proposes an attribute-unionability framework similar to Nargesian et al. [33] that adds attribute name similarity, regular expression similarity, and distribution similarity to determine the relatedness between tables. We focus on attribute values, and also include relationship and type hierarchy semantics.

2.2 Annotating Attributes and Relationships

Supervised Approaches. Sherlock is a supervised technique to annotate attributes with 78 semantic types [19]. Zhang et al. extended Sherlock with SATO, a hybrid machine learning model that uses topic modeling and structured learning [46]. Recently, Suhara et al. introduced a multi-task learning approach called Doduo [40] that fine-tunes a BERT-base language model [9] and predicts the column types and the relationship types between columns using column values. These approaches, along with other supervised approaches [23, 42], require training over annotated data. Due to the diversity of data lakes (and the large number of possible column and relationship types) and the lack of comprehensive training data, it is difficult to employ a supervised approach to solve the table union problem.

Unsupervised Approaches. There are several unsupervised approaches that annotate columns using KB's. Mulwad et al. [31] and Syed et al. [41] base their work on KBs to find type labels for column headers. Mazumdar and Zhang propose TableMiner+ [28, 48], which detects a subject column and incrementally finds attribute values in a KB to classify the columns. Venetis et al. [43]

extracts data from the web and constructs an isA database for types and a relation database for properties (relationships) to provide a wide coverage of types and relationships. These unsupervised approaches rely on information that is sparse in a data lake, such as column headers. Our approach uses both an existing and a synthesized KB to find column and relation semantics from only attribute values. Also, we make use of a KB type hierarchy to expand the range of union search.

2.3 Other Related Work

Embeddings. Nargesian et al. [33] introduced the use of word embeddings for table union search. Cappuzzo et al. [6] proposed to create relational embeddings for each table based on neighboring values in rows and columns. They consider a few data integration tasks, but not unionability and require significant training data.

Domain Discovery. Ota et al. [34] introduce D4, an unsupervised algorithm that uses co-occurrence of values in different tables to discover the set of values that belong to the same domain. Our approach also makes use of co-occurrence of values when creating our synthesized KB, and we not only use co-occurrence of values in a column, but also consider the co-occurrence of values in binary relationships to discover relationship domains.

Semantic Search over Structured Data. Galhotra et al. [14] proposes S3D, a system that finds related tables, rows, and columns from KBs and datasets. S3D's search method is similar to ours, but they do not account for low KB coverage.

3 RELATIONSHIP UNIONABILITY

We now introduce the building blocks of SANTOS (SemANtic Table uniOn Search). SANTOS determines unionability based on not just the semantics of columns (*column semantics*), but also the semantics of binary relationships between columns (*relationship semantics*).

Column Semantics. Like previous approaches to semantic annotation or column-type discovery in data lakes [19, 34, 46], we associate each column with a set of semantic annotations. An annotation may be a type in a KB or, when using unsupervised techniques [33, 34], we may determine that a set of columns have the same, but unknown semantics (for example, they are *attribute unionable* [33]). We call this *column semantics*. As an example, in Tables (a) and (b) of Fig. 1, even if no known semantics can be found for Park information, we can still determine that the first columns share the same (but unknown) semantics. As the discovery process is uncertain, each annotation or type in the column semantics is associated with a confidence score.

DEFINITION 3 (COLUMN SEMANTICS). *Each column c in a table T has a* column semantics *(denoted $CS(c)$) which is a set of annotations each defining a conceptual domain to which the values in the column may belong. Each annotation $a \in CS(c)$ has a confidence score between 0 and 1 that reflects the confidence in the inclusion of the annotation a in $CS(c)$.*

EXAMPLE 4. *In Table (b) of Fig. 1, CS(*Park Name*) may have a set of types {*Place, Tourist Attraction, Park*} and CS(*Film Title*) may have {*Creative Work, Movie*}. The confidence score will reflect our confidence in each type, for example if, hypothetically, more values in column* Park Name *map to type* Tourist Attraction *and only a few to the more specialized type* Park, *then a method may assign a lower confidence score to* Park. *There may be other columns for which the column semantics is the empty set.*

Relationship Semantics. In addition to column semantics used in previous table union search approaches [3, 33], we use relationship semantics to guide the search. Note that a relationship is any binary relation between column values (in particular, it does not need to be a function). In SANTOS, each pair of columns is associated with a set of relationships, each with a confidence

score. Using a KB, we may assign a column pair with a known relationship (for example, a property) in the KB [18, 36], or we may use unsupervised techniques to determine that a set of column pairs have the same (but unknown) relationship.

Definition 5 (Relationship Semantics). *Each pair of columns c_1, c_2 in a table T has a relationship semantics (denoted $RS(c_1, c_2)$) which is a set of annotations, each defining a conceptual relationship to which the tuples in $\pi_{c_1,c_2}(T)$ may belong. Each annotation $a \in RS(c_1, c_2)$ has a confidence score between 0 and 1 that reflects the confidence in the inclusion of the annotation a in $RS(c_1, c_2)$.*

> Example 6. *Relationship semantics in Table (b) of Fig. 1 include* DirectedBy *between* Film Title *and* Film Director, *and* HasPhone *between* Park Name *and* Park Phone. *There may be no relationship discovered between* Film Title *and* Park Name, *in which case* RS(Film Title, Park Name) $= \emptyset$.

Semantic Graph. A relationship can be represented as a (Subject, Predicate, Object) triple or an edge in a graph whose nodes are the columns of a table and edges connect pairs of columns. For now, we assume we have an oracle (which may be a KB or any relationship discovery tool) that provides column and relationship semantics. We can form a *semantic graph* for each table T that contains a node for each column and an edge between two columns if they have non-empty relationship semantics.

Definition 7 (Semantic Graph). *Given a set of columns $C = \{c_1, c_2 \ldots c_m\}$ in a table T, the semantic graph of T is a graph $SG(T) = (V, E)$ with a distinct vertex v_i for each column c_i, labeled with $CS(c_i)$. For each column pair (c_i, c_j), $i \neq j$, with non-empty relationship semantics $RS(c_i, c_j) \neq \emptyset$, there is an undirected edge e_{ij} between v_i and v_j labeled with $RS(c_i, c_j)$.*

> Example 8. *Semantic graphs for data lake Table (b) about parks and films and Table (c) about people are shown in Fig. 1 (e), (f) respectively (with column semantics omitted). As seen in (e), semantic graphs of data lake tables may not be connected, depending on the relationship discovery. Some edges are dashed only to make the connection with Tables (b) and (c) more intuitive.*

Unionability Search over Semantic Graphs. In SANTOS, a user (a data scientist) provides a Query Table (denoted by Q) and specifies its *intent column* (denoted by I), which is a column of most interest to the user that forms relationships of interest with other columns. SANTOS creates a semantic graph for the query table that is restricted to being a tree rooted at the intent column, called the *Query Semantic Tree*. We find relationships from the intent column to other columns in the query table, then transitively search for relationships from these columns. This process allows a user to direct the search to include their intent column. We search for data lake tables that contain a column with a similar (matching) semantics to the intent column and with similar relationships. More specifically, we will look for a tree within each semantic graph of data lake tables that matches a subtree of the query tree rooted at the intent column. We do not require that the full query semantic tree be covered by a data lake table. We assume a scoring function that considers the strength of column and relationship matching as well as how much of the query tree is matched. We define a precise scoring function in Sec. 6, after we describe how we compute column and relationship semantics. The scoring function captures how closely the Semantic Graph of a data lake table matches with the Query Semantic Tree.

> Example 9. *Consider Table (a) about parks in Fig. 1 as a query table. Suppose a user's intent column is* Park Name, *which has two relationships (1)* Park Name–LedBy–Supervisor *and (2)* Park Name–LocatedIn–City. City *forms a third relationship:* City–LocatedIn–Country. *The semantic*

graph is depicted in Fig. 1 (d). For brevity, we omit the CS in the figure and include only one RS for each edge. With the intent column I, we can determine if the semantic graph of Table (b) (Fig. 1(e)) contains a subtree that maps to the query semantic tree rooted at I. If the CS for Park Name *and* City *match in (a) and (b) and RS of* locatedIn *matches* locatedIn, *then (b) is a candidate unionable table that unions on* $\Pi_{\text{Park Name, Park City}}$. *Notice however the semantic graph Fig. 1(f) of Fig. 1(c) about people. Although there is a possible matching relationship involving* Birthplace *and* City *(with* City–LocatedIn–Country*), (c) is not a candidate unionable table as there is no good match with the user's intent column.*

DEFINITION 10 (SANTOS TOP-*k* UNION SEARCH SOLUTION). *Given a set of Data Lake Tables $\mathcal{T}$, a query table Q with a specified intent column I, a semantic graph for Q, SG(Q), which forms a tree rooted at I. We assume a semantic graph matching or scoring function S that for each $T \in \mathcal{T}$ returns the highest scoring subtree of SG(T) that matches a subtree of SG(Q) rooted at I along with its score. The SANTOS union search solution is the top-k data lake tables having the highest score modeled by S.*

We introduce two methods to create semantic graphs. Sec. 4 describes one that uses an existing KB, and Sec. 5 introduces another that is designed to be used when there is no or partial KB coverage over the data lake (as commonly seen with enterprise and open data lakes). We discuss how we leverage these methods to create semantic graphs in Sec. 6 and walk through our implementation in Sec. 7. Finally, we present experiments (Sec. 8) using each method independently and together to provide better accuracy.

4 SEMANTIC GRAPH CREATION WITH KB

In this section, we present a method for creating semantic graphs using an existing KB. Recall that metadata (like column headers, table name, etc.) may be missing, inconsistent or incomplete in data lake tables [1, 10, 32, 47]. Therefore, we create the semantic graphs by using the cell values only.

4.1 Column Semantics

To create column semantics (CS), we associate a set of KB types (called annotations in Definition 3) to each column. Associating columns with types from a KB is a well-studied problem [19, 38, 43, 46, 48]. Like previous work on unionability [33] and column type detection [34, 40, 46], we only use values within a column to determine the associated KB types. Like others [33, 43], we use both the KB types and type hierarchy to define CS, since there may be tables that match with a query table at a different granular level. Therefore, instead of annotating each column with a single type, we annotate with a set of types. This provides flexibility in matching a data lake table with query tables of different granularity.

EXAMPLE 11. *Consider the* Birthplace *column in Table(c) of Fig. 1 that describes where people were born. We might assign a more specific type* city *to this column because the majority of values are cities. However, we might also assign a broader type* Place *as this column also contains information on places that are not cities (e.g.,* Texas *and* Barnet*). Consequently, using any single type for the columns in the data lake tables can impact the effectiveness of union search as it may differ from the detected type in the query table. Therefore, we keep a set of types as the CS and select one to use at query time depending on the query table CS. This allows us to match the same data lake table column with the columns from different query tables—some with just cities and some with places.*

We assign CS based on *semantic consistency*. For example, if a column is assigned a type *place*, it can also be assigned another granular type *city* but not *music album*, which is semantically

inconsistent. We make use of the KB type hierarchy to ensure semantic consistency by only assigning types to a column that are in a ISA relationship in the KB type hierarchy.

SANTOS can be used with any open, enterprise-level, or domain-specific KB. We use YAGO 4 [36] (referred to as YAGO hereafter) in our setup which has a single root. The KB root is a generic type and is uninformative to use as part of CS. As YAGO has a large and rich set of types that are direct descendants of the root, we choose to use all direct descendants of this root as the top level types denoted by A_{Top}. These are types that can form the root of CS and are assumed to be semantically disjoint.[1] To define the CS(c), we map each value in c to the KB. If the value appears in the KB in a type a, we add a and all its ancestors up to a top level type $\in A_{\text{Top}}$ to the column semantics candidate set $CS_{\text{candidate}}(c)$. Note that a single value may map to multiple types (for example in YAGO, Boston maps to City and Music Album[2]), and different values may map to different or identical types. After going through all values, if $CS_{\text{candidate}}(c)$ contains multiple top level types, then we keep only the top level type (and all its descendants) in $CS_{\text{candidate}}(c)$ mapped to by the majority of values, which ensures semantic consistency.[3]

> EXAMPLE 12. *Consider* Birthplace *column in Table (c) of Fig. 1 that has 5 unique values. The value* Boston *is associated with* place *and* creative work, *both of which are in* A_{Top}. *The KB returns both types for the* Birthplace *column, as well as descendants of* place *to which at least one value is mapped:* {administrative area, city, state} *and the descendant of* creative work *to which at least one value is mapped:* {music album}. *Notice the majority of the values are associated with* place. *Therefore, we select* place *as the top level type for the* Birthplace *column and discard* creative work *and its descendants. Therefore, CS(*Birthplace*) is the set of types:* {place, administrative area, city, state}.

4.2 Column Semantics Confidence

We assign a confidence score to each type by taking the product of the *frequency score* (*fs*) and *granularity score* (*gs*). This score captures the same intuition as the well known measure TF-IDF [2, 37]. TF-IDF measures the importance of a word in a document considering its occurrence and specificity. Here we want to measure the importance of each type. In our case, TF, which is the occurrence of a term in the document [25], aligns with *fs*; and IDF, which captures the specificity and rareness of the term [39], aligns with *gs*.

Concretely, the *frequency score fs*(a) of type $a \in CS(c)$ in column c is the fraction of unique values in c that are mapped to a ($|c_a|$) out of all unique values in c mapped to the KB ($|c_{\text{KB}}|$):

$$fs(a) = \frac{|c_a|}{|c_{\text{KB}}|} \tag{1}$$

> EXAMPLE 13. *Consider the CS for the column* Birthplace *of Fig. 1(c) which includes the annotations* city *and* state. *These annotations are siblings in the KB, each with an ISA relationship to a parent* place. *Notice however, only 1 unique data value (*Texas*) out of 5 is associated with* state *while 3 are associated with* city (Barnet *is mapped to neither* city *nor* state *but does map to* place). *To model this difference, we assign frequency scores fs(*state*) = 0.2 and fs(*city*) = 0.6. As all values are mapped to the top level type* place, *fs(*place*) = 1.0.*

A set of entities of a descendant type is always a subset of entities belonging to its ancestor types in the type hierarchy. So, the higher level types are always mapped to by the same or more entities

---

[1]In a different KB, a different choice may be made.
[2]https://en.wikipedia.org/wiki/Boston_(album)
[3]In the rare case of ties, we keep the rarer top level type having the fewest entities.

than children types. This is reflected in the frequency score. From the perspective of information theory, the less probable outcome has greater information [26, 39]. In our case, a type that appears more frequently in the KB is less informative. As mentioned earlier, inspired by the well-known tf–idf measure [2], we penalize the frequent ancestor types by using a frequency-based strategy utilizing KB statistics and assign a *granularity score (gs)* to each type. The basic idea is to count the frequency of each type a by counting the entities that map to type a (i.e., *a.count*) and penalize the frequent types. The granularity score is computed in Eq. (2) which uses a log function.

$$gs(a) = \frac{1}{max(1, \log(a.count))} \tag{2}$$

To keep *gs* consistent with *fs*, *gs* also ranges from 0 to 1. Note that for rare types with less than 10 entities, the log function returns a value less than 1. Thus, we use the *max* function in the denominator.

EXAMPLE 14. *Consider the entity* Boston, *which belongs to the types* city *and* place. *In YAGO, over 6 million entities have type* place, *and ~42,000 entities have type* city. *Then,* city *is a more informative type than* place, *which is an ancestor type. Hence, the granularity scores are* $gs(\text{place}) \approx 0.14$ *and* $gs(\text{city}) \approx 0.22$.

We assume that the computations of *granularity score* and *frequency score* are independent of one another as the former is based on KB statistics and the latter is based on the semantics of values in a column. Our objective is to prioritize those types that are more specific (i.e. higher *granularity score*) and also capture the semantics of the column values better (i.e. high *frequency score*). Thus, given a column c with annotation a, we define the KB Column Semantics Confidence Score by aggregating *fs* and *gs* of a:

$$CS_{\text{CONF}}(c, a) = \begin{cases} fs(a) \cdot gs(a) & \text{if } c \in \text{data-lake table } T \\ fs(a) & \text{if } c \in \text{query table } Q \end{cases} \tag{3}$$

Note that to avoid double penalization, we only penalize the top-level types in the data lake tables.

EXAMPLE 15. Birthplace *in data lake table* Fig. 1(c) *has CS with respective* CS_{CONF} {place:1.0 · 0.14, administrative area:1.0 · 0.17, city:0.60 · 0.22, state:0.20 · 0.35}.

4.3 Relationship Semantics

We compute relationship semantics (RS) for every pair of string (non-numeric) columns within the query table and within data lake tables. Note that many pairs of columns (like Park Name and Film Director) may not have a semantic relationship represented in the KB. Intuitively, there is at least an indirect relationship between all columns in a table (in this case, the director of a film shown in the park). But in this section, we are only interested in relationships found in the KB. Suppose columns c_1 and c_2 in table T have non-empty CS (meaning they are annotated with at least one KB type). To determine RS, we determine if a pair of values in $\pi_{c_1,c_2}(T)$ is associated with entities (e_1, e_2) that have a KB relationship p. The RS Confidence Score $RS_{\text{CONF}}(c_i, p, c_j)$ for a binary relationship p between columns c_i, c_j ($\pi_{c_i,c_j}(T)$) in T is:

$$RS_{\text{CONF}}(c_i, p, c_j) = \frac{|(c_i, c_j)_p|}{|(c_i, c_j)_{\text{KB}}|} \tag{4}$$

such that $|(c_i, c_j)_p|$ is the number of unique value-pairs with predicate p from KB, and $|(c_i, c_j)_{\text{KB}}|$ is the total number of unique value pairs mapped to the KB. Note that only the relationship semantics

with the maximum score is included in the semantic graph.[4] Unlike column semantics, we do not penalize the relationship semantic types because the relationships do not have a type hierarchy.

> EXAMPLE 16. *In Fig. 1(c), RS(*Person,Birthplace*) contains the annotation* birthplace *with confidence score* RS_{CONF} *= 1.0.*
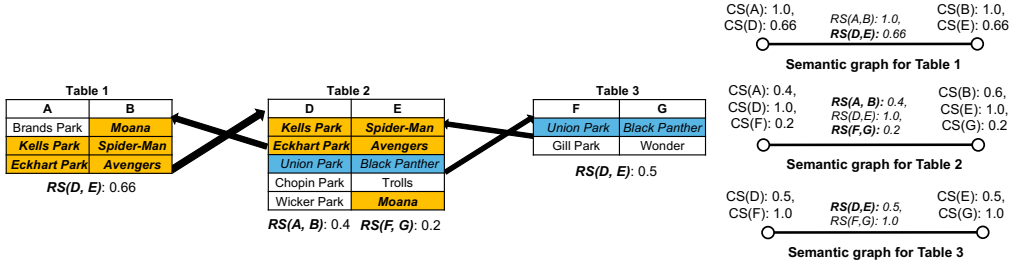
5 SYNTHESIZED KB SEMANTIC GRAPH

Fig. 2. Synthesized relationship semantics of data lake tables and their respective semantic graphs. Value-pairs bolded and highlighted in orange appear in Tables 1 and 2, and the value-pair italicized and highlighted in blue appear in Tables 2 and 3.

KBs may have limited coverage over real data lakes. Hence, using only an existing KB (even a set of KBs) to determine CS and RS can lead to low coverage. Our experimental study indicates that YAGO [36], a well-known and maintained KB, covers only 42% of the string cell values in UK open data and 34% in Canada open data. To solve this problem, we propose a novel data-driven approach using the data lake itself, creating a *synthesized KB*. Our key insight is that we can *replace the role of an existing KB in finding CS and RS with a KB that captures co-occurrence information across data lake tables*. To determine CS, instead of mapping values to an existing KB, we now use a mapping to other columns with the same values. Rather than finding actual semantic types, we leverage type co-occurrence across columns to decide their semantic compatibility. To do so, we annotate all values that co-occur in a column or a meaningful relationship with a "synthesized type." We then determine CS and RS by also considering other column and column pairs (thus their synthesized types) with overlapping values.

5.1 Synthesized Column Semantics
Generally, values within the same column share the same semantic types. For example, in Fig. 1(c), all values in the Person column have type person and values in Birthplace share type place. Considering this property and assuming we do not have homographs [22], we start by assigning to each column in the data lake a unique synthesized column semantics with a confidence score equal to 1. For example, considering Table 1 about parks and movies in Fig. 2, we assign synthesized column semantics A, B to the columns. Similarly, we assign D, E and F, G as column semantics to the columns in Tables 2, 3 respectively.

There can also be different columns that share the same column semantics. For instance, consider Tables 1 and 2 in Fig. 2. Notice that columns A and D are both about parks while columns B and E are about movies. We hypothesize that columns with common semantics have overlapping values that share the same types. We want to determine how likely it is for a column *c* to share semantics

---

[4]If there is a tie, we pick the annotation for *p* that has the smallest number of entity pairs in the KB, thus preferring the rarer predicate.

with column c_j, and thus inherit $CS(c_j)$. Formally, along with its own column semantics, like in Sec. 4.2, we also assign synthesized type a from column c_j to column c with a confidence score. We define a synthesized column semantics confidence score for column c where CS_{CONF} of a is the fraction of unique values in c that are also in c_j (i.e., $|c \cap c_j|$) over the total unique values in c ($|c|$). If $c = c_j$, then CS_{CONF} of a is 1.

$$CS_{\text{CONF}}(c, a \in CS(c_j)) = \begin{cases} 1 & \text{if } c = c_j \\ \dfrac{|c \cap c_j|}{|c|} & \text{otherwise} \end{cases} \tag{5}$$

Note that as we do not have type hierarchy information for the synthesized column semantics, we assume all synthesized CS are of the same granularity level and set $gs(a) = 1$ for each type.

5.2 Synthesized Relationship Semantics

For column pairs in a table that share a *"meaningful relationship"*, we assign a synthesized relationship semantics (RS). For example, consider Table 1 of Fig. 2 about films shown at parks, where columns A and B have a meaningful relationship, annotated as $RS(A, B)$. Following the same logic as annotating synthesized CS, we consider two column pairs to have the same RS if their value pairs overlap. For instance, column pairs (A, B) and (D, E) in Table 2 of Fig. 2 have overlapping value pairs (depicted in bold text), indicating that they likely share a meaningful relationship. Therefore, along with its own synthesized RS, we also assign a synthesized RS $p \in RS(d_i, d_j)$ of column pair (d_i, d_j) to the column pair (c_i, c_j) $(\pi_{c_i,c_j}(T))$ of table T with synthesized RS_{CONF} given by:

$$RS_{\text{CONF}}(c_i, p, c_j) = \begin{cases} 1 & \text{if } c_i = d_i, c_j = d_j \\ \dfrac{|(c_i, c_j) \cap (d_i, d_j)|}{|(c_i, c_j)|} & \text{otherwise} \end{cases} \tag{6}$$

where $|(c_i, c_j) \cap (d_i, d_j)|$ is the number of unique overlapping value pairs between distinct column pairs and $|(c_i, c_j)|$ is the total unique value-pairs in (c_i, c_j).

Note that although we may have multiple types in relationship semantics for each column pair (seen in Semantic Graphs in Fig. 2), we use only the type that best matches with the query table during the query phase (Sec. 6). We discuss implementation details for efficient indexing of the synthesized KB in Sec. 7.

6 SANTOS UNION SEARCH

In the previous sections, we presented two methods for creating semantic graphs: one relying on a high-quality existing KB (YAGO), the other on our proposed synthesized KB. We now discuss how to compute a unionability score from both semantic graphs.

Scoring function. Recall that SANTOS takes a query table as input from the user and the objective is to generate a ranked list of top-k unionable tables. Generating a ranked list of relevant documents for a given query is a well-studied problem in the literature [13, 15]. Recently, Ho et al. formulated a ranking function to rank a list of relevant text documents for a given sentence query [16]. They extract a set of tokens representing the context of the query sentence and each candidate document. Each token is assigned a confidence score. The (normalized) summation of the confidence scores of matching tokens is used to rank the documents. This function is modified and used to rank the relevant web tables for the given quantity queries [17]. We use the same concept to motivate our scoring function without normalization as this does not change the ranking. In our work, the context is represented by matching query table's column pairs with data lake table's column pairs and the confidence is represented by their matching quality. Specifically, we match a connected subtree of Q's semantic query tree, rooted at the intent column, into the semantic graph

of data lake tables. Intuitively, the larger the number of matching column pairs and the higher the confidence that the corresponding semantic types agree, the higher the match score. Note that in the score computation, nodes and edges may each be annotated with multiple possible CS and RS (Sec. 4.1, Sec. 5.2), each with respective confidence scores.

EXAMPLE 17. *Consider table (a) of Fig. 1 about parks. Possible semantics for column pair (*Park Name, City*) are* park-locatedin-place, park-locatedin-city, *etc. based on the given KB. When value pairs from these columns are not covered by the given KB, we also obtain semantics from the synthesized KB such as* CS(W)-RS(W,X)-CS(X) *and* CS(Y)-RS(Y,Z)-CS(Z)*, where* CS(W)*,* CS(X)*,* CS(Y) *and* CS(Z) *are the synthesized CS and* RS(W,X)*,* RS(Y,Z) *the synthesized RS.*

With semantics from both the KB and synthesized KB for the same column pair, we use a 2-step approach for computing the unionability score. First, an "intra-method" technique selects the best semantic match between query table and data lake table for each source (KB and synthesized KB) separately. Then an "inter-method" comparison selects the semantic match that maximizes the overall unionability score.

Let $a_1, a_2 \ldots a_x = (CS(Q_c) \cap CS(T_c))$ be the intersecting CS between a query table Q's column Q_c and a data lake table T's column T_c given by a semantic graph creation method G (i.e., either KB or the synthesized KB). For each i, let the corresponding confidence scores be $CS_{\text{CONF}}(Q_c, a_i)$ and $CS_{\text{CONF}}(T_c, a_i)$, respectively. Since the column semantics assigned to the query table and data lake table are independent of each other, we take a product of confidence scores for each token to get their match score. Then, we select the match that maximizes the score. Intuitively, the match score between Q_c and T_c is given by the column semantics with the greatest product of confidence scores:

$$colMatch_G(Q_c, T_c) = \max_i \ CS_{\text{CONF}}(Q_c, a_i) \cdot CS_{\text{CONF}}(T_c, a_i). \tag{7}$$

We determine the relationship match score for column pairs (Q_{c1}, Q_{c2}) in Q and (T_{c1}, T_{c2}) in T analogously as:

$$relMatch_G((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2})) = \max_i \ RS_{\text{CONF}}(Q_{c1}, p_i, Q_{c2}) \cdot RS_{\text{CONF}}(T_{c1}, p_i, T_{c2}). \tag{8}$$

Here $p_1, p_2, \ldots, p_x = (RS(Q_{c1}, Q_{c2}) \cap RS(T_{c1}, T_{c2}))$ are the intersecting relationship semantics between the column pairs. Note that depending on the KB, $RS(T_{c1}, T_{c2})$ may differ from $RS(T_{c2}, T_{c1})$. Therefore, the KB may return $RS(T_{c1}, T_{c2})$ for the data lake table and $RS(Q_{c2}, Q_{c1})$ for the query table. To model this, we preserve both $RS(T_{c1}, T_{c2})$ and $RS(T_{c2}, T_{c1})$ for the data lake table. Once we get RS for the query table, we match the data lake table's RS that maximizes the score with respect to the query table's RS in Eq. (8).

The overall match score between Q's column pair (Q_{c1}, Q_{c2}) and T's column pair (T_{c1}, T_{c2}) based on method G is then computed as:

$$pairMatch_G((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2})) = colMatch_G(Q_{c1}, T_{c1}) \cdot relMatch_G((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2}))$$
$$\cdot colMatch_G(Q_{c2}, T_{c2}) \quad (9)$$

Let KB and $Synth$ represent the semantic graph creation methods based on the given and synthesized KB, respectively. Recall that CS_{CONF} based on the given KB is assigned a granularity score for penalizing the top-level types (Eq. (3)). However, due to the absence of a type hierarchy in the synthesized KB, we consider all types to be of same level and assign a granularity score of 1 to each synthesized column semantics (Eq. (5)). To avoid bias, we compare the "inter-method" pair matches ignoring the granularity score.

Formally, let a_1 and a_2 be the column semantics selected in $pairMatch_{KB}((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2}))$. With f denoting a flag such that $f = 1$ iff $\frac{pairMatch_{KB}((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2}))}{gs(a_1).gs(a_2)} \geq$
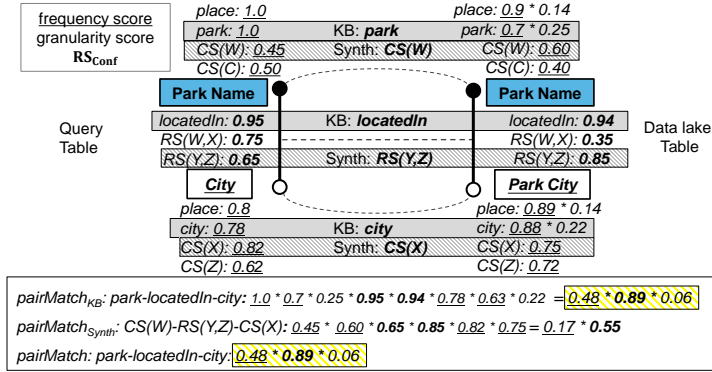
Fig. 3. Computation of pairMatch score between the semantic tree of the parks table Fig. 1(a) as query table (left) and the parks-and-movies table Fig. 1(b) as data lake table (right). From the data lake table's semantic graph, we extract the semantic tree rooted at Park Name. The dotted arcs connect the matching columns and relationships.

$pairMatch_{Synth}((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2}))$, the match score between (Q_{c1}, Q_{c2}) and (T_{c1}, T_{c2}) is computed as:

$$pairMatch((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2})) = \begin{cases} pairMatch_{KB}((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2})) & \text{if } f = 1 \\ pairMatch_{Synth}((Q_{c1}, Q_{c2}), (T_{c1}, T_{c2})) & \text{otherwise} \end{cases} \quad (10)$$

> EXAMPLE 18. *Let Table (a) in Fig. 1 about parks be the query table (Q) with* Park Name *as the intent column. Consider Table (b) in Fig. 1 about parks and movies as the data lake table (T). There is a possible match between these two tables such that* Park Name *matches with* Park Name *and* City *matches with* Park City*. We compute the corresponding pair match score as follows. Fig. 3 shows the semantic graphs and the scores involved in selecting the matching pairs. The query table semantic tree is shown on the left and the semantic tree extracted from the data lake semantic graph rooted at* Park Name *is shown on the right. The semantic graph before extracting the tree is shown in Fig. 1(e). The dotted arcs connect the matching nodes and edges. First, we use Eq. (7) to find the matching column semantics at each node between the tables using existing and synthesized KBs. For* Park Name *(Query Table) and* Park Name *(Data Lake Table),* park *and* CS(W) *are selected from the existing and synthesized KB, respectively. Also for* City *and* Park City*,* city *and* CS(X) *are selected using the same formula (Eq. (7)). Furthermore, as we have only one relationship semantics for KB, i.e.,* locatedIn*, it is selected as the winner. For the synthesized KB method,* RS(Y,Z) *wins over* RS(W,X) *according to Eq. (8). Finally, we use Eq. (10) to perform the inter-method comparison between* park–locatedIn–city *and* CS(W)–RS(Y,Z)–CS(X)*. As shown in the figure, as* $max(0.48 \cdot 0.89, 0.17 \cdot 0.55) = 0.48 \cdot 0.89$*, we select* park–locatedIn–city *as the pairMatch between the column pairs.*

If a branch is selected from the existing KB, we include the granularity score in the pairMatch score so that when we compare the score between the query table and different data lake tables, the data lake tables matching in the granular types are prioritized over the tables that match just on the top level types. Our *pairMatch* score may be low if there is no match at the most granular level. One can design a different penalization scheme to penalize the frequent types to change the trend of scoring. However, our objective is to introduce relative difference in the match between the query table and each data lake table rather than the absolute unionability score.

Let there be m matching column pairs between Q's semantic tree $SG(Q)$ rooted at intent column I and T's subtree $SG(T)$ rooted at c. We compute the unionability score (S) between Q and T using

$$S(Q,T) = \sum_{i=1}^{m} pairMatch((Q.I, Q.c_i), (T.c, T.c_i)). \tag{11}$$

Intuitively, Eq. (11) favors the data lake table that has more matching columns, finer granularity of matching types, and greater probability of matching types.[5] The latter two are captured by the individual matching scores while the former is addressed by the summation of such scores. Note that unionability between two tables is often viewed as a binary decision, i.e., either the tables are unionable or not. In a traditional database setting, two tables are unionable iff they have compatible schema.[6] In the data-lake setting, the constraint of aligning a unionable table's schema exactly with that of a query table is not practical. Existing works [3, 33] have relaxed this in their unionability definitions. However, they solely consider column unionability without considering the relationships and the preservation of the query table's topics. Thus, by introducing SANTOS, we aim to show the importance of considering relationships when searching for unionable tables that share the same intent as the query table.
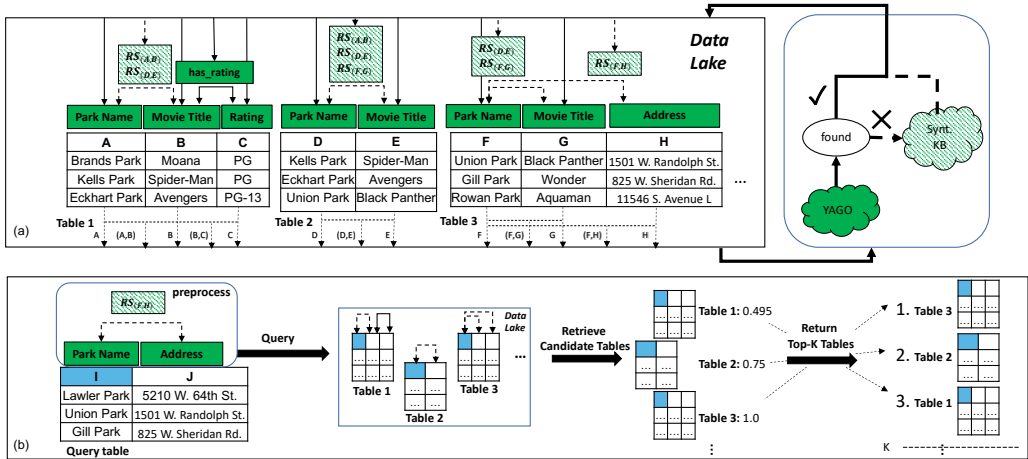
7 SANTOS IMPLEMENTATION

Fig. 4. Pipeline of SANTOS. (a) Preprocessing phase: data-lake tables are labeled with types from YAGO or *synthesized KB*. (b) Query phase: the query table is annotated, and we query the data lake to retrieve and rank unionable tables. Columns highlighted in blue represent the (matching) intent column. The type hierarchy is omitted for simplicity.

We now describe SANTOS implementation, namely, the KB use, the synthesized KB creation, and the pre-processing and query phases.

7.1 Knowledge Base Implementation

When creating CS and RS using YAGO types, we build four dictionaries for efficient access. First, we store labels and alternate names (synonyms) that describe the corresponding entity, derived

---

[5]The score is normalized between 0 and 1 in the literature [16, 17].
[6]One might also pre-process the tables for compatibility, for example, project out columns or rename attribute names.

from YAGO, mapped to URIs[7], in an *entity dictionary*. [8] We connect tables to the KB by mapping column values to KB labels and alternate names. Note that YAGO permits multiple entities to have the same label or alternate name, so a value may be mapped to entities of different types (e.g. homographs [22]). Second, we build an *inheritance dictionary* that stores each Top level type and their children types. Next, we use a *type dictionary* to lookup entities as keys, which are also values in the *entity dictionary*, with their set of (types, granularity score) as values (e.g. Boston: {(place, 0.14), (city, 0.22),...}). Finally, we use a *relationship dictionary* to store the set of binary relationships (properties) for each value-pair in the KB, similar to the (proprietary) relation database used to recover semantics of webtables [43].

As we index only the necessary KB triples, the total space taken by these dictionaries in main memory is 3.75 GB. We need 965.18 MB to store them persistently. [9] Note that creating these dictionaries is one-time task and takes less than 20 minutes in our experimental setup (see Sec. 8.1).

7.2 Synthesized KB Implementation

For the synthesized KB, we create a *Synthesized Type Dictionary* and *Synthesized Relationship Dictionary*. As we directly use cell values as entities, an *entity dictionary* is not required. Also, the inheritance dictionary is not needed as we consider all synthesized types to have the same granularity. We will only discuss the creation of the *Synthesized Relationship Dictionary* for column pairs, since the *Synthesized Type Dictionary* is created in the same way.

Relationship	RS: Type Score
(Brands Park, Moana)	RS(A, B): 1.0, **RS(D, E)**: 0.66
(Kells Park, Spider-Man)	RS(A, B): 1.0, RS(D, E): 1.0, **RS(F, G)**: 0.2
(Union Park, Black Panther)	**RS(A, B)**: 0.4, RS(D, E): 1.0, RS(F, G): 1.0
(Chopin Park, Trolls)	**RS(A, B)**: 0.4, RS(D, E): 1.0, **RS(F, G)**: 0.2
(Gill Park, Wonder)	**RS(D, E)**: 0.5, RS(F, G): 1.0

Fig. 5. *Synthesized Relationship Dictionary* for Fig. 2.

Synthesized Relationship Dictionary. In Sec. 4, we built a *relationship dictionary* using an existing KB to create annotation to add to the RS. Here, we follow an inverse process, i.e., we first assign a new binary RS to each column pair, then populate a synthesized relationship dictionary with each *value-pair* in the column pair that is not in any existing KB. For each such value-pair, we assign a binary *synthesized relationship semantic (RS)*, which is a distribution of column pairs that exhibit some meaningful relationships among value-pairs. A similar idea exists in topic modeling [35], where some latent topic of a document (value-pair) is represented by a distribution over other documents (column-pairs).

We first assign all value pairs in a column pair with the same synthesized RS. For instance, value-pairs (Brands Park, Moana), (Kells Park, Spider-Man) and (Eckhart Park, Avengers) in Table 1 of Fig. 2 are all assigned $RS(A,B)$. As we have discussed however (Ex. 6), some binary relationships between columns in a data lake are not meaningful and may be indirect. For example, in Fig. 1 Table (c), value-pairs in $\Pi_{FilmTitle,ParkLocation}$ like (Bee Movie, 6748 N. Sacramento Ave.) and (Coco, 5210 W. 64th St.), have a tenuous relationship at best (film shown in a park with this address). Thus, we aim to capture only *meaningful* binary relationships in which one column functionally determines another. For example, in Fig. 1(b), Park Name functionally determines Park Location, indicating a possible meaningful relationship. We hypothesize that column pairs in a

---

[7]We only index labels and alternate names in English.
[8] https://www.w3.org/2004/12/q/doc/rdf-labels.html
[9] We report main memory space returned by sys.getsizeof() and indexes are stored as compressed pickle files (https://docs.python.org/2/library/pickle.html).

functional dependency are more likely to contain a meaningful semantic relationship that also exists in other tables and may be useful in union search. This also has a benefit of reducing the synthesized relationship dictionary size. We use an existing *functional dependency* (FD) discovery algorithm called FDEP [12] to find unary FDs (FDs with a single column determinant in binary relationships), and run their bottom-up variant that first considers all pairwise relationships, then checks if a dependency satisfies an FD. FDEP time complexity is quadratic but we only use it offline.

To generate our synthesized relationship dictionary, we first iterate over the data lake tables and store value-pairs in column pairs (not found in the existing KB) that form an FD, and their synthesized RS with a type score of 1, in a *lookup dictionary*. We then find the type score of each RS to capture the likelihood for the associated value-pair to have that semantics. Type scores, also seen as confidence scores for value-pairs, are also calculated based on the overlap of value pairs in different tables (consistent with Eq. (6)). However, we now consider value-pairs rather than column-pairs, so we calculate the overlap score for each value-pair in a column pair. Thus, we fill the *Synthesized Relationship Dictionary (S)* with RS from the lookup dictionary and their associated type scores. We clarify the computation using the following example.

EXAMPLE 19. *Consider Tables 1, 2, and 3 in Fig. 2 and S in Fig. 5. First, we assign all value-pairs with $RS(A,B), RS(D,E)$ and $RS(F,G)$ with respect to their consisting column pairs with type score 1. Next, consider Table 1 ($RS(A,B)$), which shares two value-pairs (bolded) with Table 2 ($RS(D,E)$) out of three total value-pairs.* (Brands Park, Moana) *in Table 1 is then also assigned $RS(D,E)$ with a type score of $\frac{2}{3}$. Now consider Table 2, which shares two value-pairs with Table 1 (bolded) and one value-pair with Table 3 (italicized), out of five total value-pairs. Then,* (Chopin Park, Trolls) *is assigned $RS(A,B)$ with a type score of $\frac{2}{5} = 0.4$ and $RS(F,G)$ with a type score of $\frac{1}{5} = 0.2$.*

7.3 Pre-Processing Phase

During the offline pre-preprocessing phase, we find CS and RS for data lake tables using an external KB (Sec. 4), and discover FDs in the data lake tables to create a synthesized KB (Sec. 7.2). To reduce query time, we create two inverted indexes. First, a *node inverted index* maps a column to its CS with respective CS_{CONF}. The second is an *edge inverted index* that maps RS to its connected columns in the same table, with RS_{CONF}.

Time Complexity. Consider a set of data lake tables $\mathcal{T}$ and let m and n be the largest number of columns and rows respectively in a data lake table. We make a linear pass over each column and record the count of each candidate CS (include candidate top-level types) in an inverted index. So, the asymptotic time complexity of computing CS is $O(|\mathcal{T}| \cdot m \cdot n)$. Similarly, for computing RS, we only consider columns with non-empty CS, which can then find RS in the KB [36]. Let m_c be the largest number of columns having CS in a data lake table such that $m_c \leq m$. The time taken to compute RS is bounded by $O(|\mathcal{T}| \cdot m_c^2 \cdot n)$.

The time complexity of creating synthesized KB includes the creation of *Synthesized Type Dictionary* and *Synthesized Relationship Dictionary*. *Synthesized Type Dictionary* creation is analogous to CS computation, where we make a linear pass over each column ($O(|\mathcal{T}| \cdot m \cdot n)$ time). Also, the creation of *Synthesized Relationship Dictionary* is analogous to RS computation. But in addition, we also need to mine unary FDs, which is quadratic in the number of columns [12]. So, its creation is linear in the highest number of rows and quadratic only in the highest number of columns across any table i.e., $O(|\mathcal{T}| \cdot m^2 \cdot n)$. Recall that this is a pre-processing task and does not need to be done during the query phase. We report the pre-processing time on different benchmarks in Sec. 8.4.

7.4 Query Phase

At query time, shown in Fig. 4(b), the user provides a query table Q in which the intent column is marked.[10] We then create the semantic tree for Q, rooted at the intent column. For this, we access KB and synthesized KB to find the CS and RS for the query table. Given the annotated semantics of the query table, SANTOS searches for unionable tables. It does so by retrieving a set of candidate tables and their respective confidence scores from the inverted index. SANTOS computes the unionability score between column pairs of Q and candidate tables based on different hierarchy levels, allowing us to match columns of different granularities (see Eq. (7)).

Time complexity. Given a query table Q with m columns and n rows, we compute its CS from both KB and synthesized KB by making a single pass over each column, which takes $O(m \cdot n)$ time. Recall that we only find RS for columns that have non-empty CS. Let m_c be the number of columns having CS such that $m_c \leq m$. RS can thus be computed in $O(m_c^2.n)$ time. After finding CS and RS for the query table, the computation of unionability score depends linearly on the number of semantics found for the query table. With inverted indexes and the number of possible CS and RS as constant during the query phase, it is efficient to compute the unionability score. We report the query time using real data lake tables in Sec. 8.4.

8 EXPERIMENTS

In this section, we compare SANTOS with a state-of-the-art unionability approach D^3L [3] that uses column unionability. Notice that D^3L builds on Table Union Search [33] by adding regular expressions, domain distributions, etc. Hence, we only compare with D^3L to analyze the importance of relationship semantics in union search.

8.1 Experimental Setup

SANTOS is implemented in Python on a server with Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz processor. Our code is publicly available.[11] Our experiments aim to answer the following questions:

(1) How effective is SANTOS in returning top-k unionable tables relative to the baseline given a query table? (Sec. 8.2)
(2) How do each component of SANTOS (use of existing KB, use of synthesized KB, and use of both) influence the quality of the results? (Sec. 8.3)
(3) How well does SANTOS scale over real data lakes, as compared to prior work? (Sec. 8.4)

Evaluation Measures. Since our method, along with other table union search methods, returns a top-k list of unionable tables, we use mean average precision ($MAP@k$) to evaluate the effectiveness of table union search approaches [27]. Following previous works [3, 33], we also report Precision at k ($P@k$) and Recall at k ($R@k$) [27]. When creating our ground truth for evaluations, we assign a binary score $\in \{0, 1\}$ to a data lake table T, given a query table Q to label T as unionable or not-unionable to Q. Formally, let $\mathcal{T}_Q$ be the set of unionable tables based on the ground truth and $\hat{\mathcal{T}}_Q$ be the set of top-k unionable tables based on some method with respect to a query table Q. Then, $P@k$ and $R@k$ with respect to Q are given by:

$$P@k = \frac{\mathcal{T}_Q \cap \hat{\mathcal{T}}_Q}{\hat{\mathcal{T}}_Q}, R@k = \frac{\mathcal{T}_Q \cap \hat{\mathcal{T}}_Q}{\mathcal{T}_Q} \qquad (12)$$

---

[10]The user may also select a set of property columns to consider. In our experiments (Sec. 8), we simply assume that the user is interested in all property columns.
[11]https://github.com/northeastern-datalab/santos

Note that the size of $\hat{\mathcal{T}}_Q$ is set to k, while $\mathcal{T}_Q$ may, in general, be larger (or smaller) than k. To best understand the results, we create benchmarks where the ground truth ($\mathcal{T}_Q$) is at least k. Hence, using $P@k$, if a method returns less than k results, the results not returned are considered incorrect (false). For instance, if $k = 10$ and the ground truth has 20 results, if a method returns only 2 results out of which 1 is correct and the other is incorrect, then $P@10 = \frac{1}{10}$ and $R@10 = \frac{1}{20}$. So when $k < |\mathcal{T}_Q|$, perfect recall is not possible as the $R@k$ can at best be $\frac{k}{|\mathcal{T}_Q|}$.

The mean average precision ($MAP@k$), defined as follows:

$$MAP@k = \frac{1}{|\hat{\mathcal{T}}_Q|} \sum_{k=1}^{|\hat{\mathcal{T}}_Q|} P@k \tag{13}$$

We compute $P@k$, $R@k$, and $MAP@k$ for each query and report the average performance over several queries for a fixed k (e.g. average $P@k$). We also measure the pre-processing and query times for scalability experiments.

Benchmarks. Figure 6 details the statistics of the benchmarks that we use, both for their data lake tables and the query tables. The benchmarks are publicly available. [12]

Table Source	Data lake Tables			Query Tables		
	# Tables	# Columns	# Rows	# Tables	# Columns	# Rows
TUS	1,530	14,810	6.8 M	125	1,610	557 K
SMALL	550	6,322	3.8 M	50	615	1.07 M
LARGE	11,090	123,477	70 M	80	1,017	1.03 M

Fig. 6. Benchmarks used in the experiments.

1. TUS Benchmark (TUS): The TUS benchmark [33] focuses on attribute unionability, ignoring relationships between columns. Specifically, two tables in the benchmark are unionable if they have unionable attributes. To repurpose this benchmark, we labeled tables in the benchmark unionable if they share relationships, not just attributes. Specifically, out of the 10 original seed tables used to produce 1530 tables in the benchmark, we found meaningful relationships in tables that originate from 6 seed tables. From these, we randomly selected 125 tables as Query Tables and marked the intent columns. We use the original data lake from the benchmark.

2. SANTOS Small Benchmark (SMALL): The SANTOS Small Benchmark contains 550 data lake tables from Canada, UK, US, and Australian open data. First, we collected 296 real data lake tables from 35 distinct domains. To further expand the benchmark, we selected 19 large tables among them and manually annotated the relationships between their columns. Using the benchmark creation technique from TUS [33], we then partitioned the annotated tables horizontally and vertically to obtain 254 non-overlapping tables, thereby increasing the total number of tables to 550. Then we randomly selected 50 tables having at least 10 unionable tables as query tables (at most 2 query tables from each domain) and labeled the intent columns and the ground truth. As the tables are taken from real data lakes, they contain nulls, string values, date, numerical values, etc. We will refer to this benchmark as *SMALL*.

3. SANTOS Large Benchmark (LARGE): To evaluate SANTOS in a broader environment, we collected 11,090 real tables from Canada and UK Open Data for the data lake. From these tables, we randomly selected 80 tables as query tables, each having at least 20 unionable tables, and marked their intent columns. For this benchmark, we only report $P@k$ and $MAP@k$ (and runtimes) as reporting $R@k$ requires a laborious annotation of the full data lake. For similar reasoning, we manually verify $P@k$ and $MAP@k$ only up to $k = 20$. Like SMALL Benchmark, these tables also

---

[12]https://zenodo.org/record/7758091

contain nulls, string values, dates, numerical values, etc. We refer to this benchmark as *LARGE* from now on.

Baselines. The table union search problem based on determining if column values are drawn from the same domain was first defined and addressed by Nargesian et al. [33]. Recently, Bogatu et al. proposed D^3L for the broader problem of finding related tables (both joinable and unionable tables) [3]. D^3L adds metrics based on column names, regular expressions and domain distributions to the word-embedding and value overlap-based models of Nargesian et al. [33]. Therefore, we compare SANTOS to D^3L with these extended metrics, by reproducing D^3L using their code.[13]

TURL is a recent method that uses representational learning over web tables [8]. TURL learns table representations that successfully find CS (column type annotation) and RS (relation extraction) in web tables [8]. Although it does not support union search directly, we extended it to create a SANTOS-like technique. Specifically, we treat TURL as a KB and, similar to SANTOS, use it to annotate the CS and RS for each table. Then we index the data lake tables similarly to the method in Sec. 4. This provides an analysis of a learning-based alternative that uses a pre-trained model.

8.2 SANTOS Effectiveness vs. Baselines

To analyze the effectiveness of SANTOS, we compare $P@k$, $R@k$ and $MAP@k$ of SANTOS against our baselines D^3L and TURL. We then analyze variations of SANTOS in an ablation study in Sec. 8.3.

Figure 7 reports the average performance of SANTOS compared to the baseline D^3L on all three benchmarks. We report TURL results only for TUS benchmark, since its performance is similar on the other benchmarks, with its measures significantly less than SANTOS. For consistency with previous research [3, 33], and to ensure that each query table has at least k expected unionable tables, we report results for $k = 60$ on TUS. Considering the number of unionable tables per query table, we report results at $k = 10$ on SMALL and at $k = 20$ on LARGE (Fig. 7). Note that when the ground truth contains more than k results, 100% $R@k$ is not possible. Rather the highest possible $R@60$ is around 62% for TUS and $R@10$ is around 72% for SMALL (Ideal lines in (b) and (d)). In the LARGE data lake the whole corpus is not labeled. Therefore, we cannot report $R@k$.

Benchmark	Method	MAP@k	P@k	R@k
TUS (k=60)	TURL	0.13	0.16	0.08
	D³L	0.64	0.58	0.31
	SANTOS	**0.80**	**0.70**	**0.37**
SMALL (k=10)	D³L	0.52	0.58	0.42
	SANTOS	**0.93**	**0.90**	**0.68**
LARGE (k=20)	D³L	0.29	0.26	-
	SANTOS	**0.77**	**0.73**	-

Fig. 7. Comparison of P@k, MAP@k and R@k of TURL, D^3L and SANTOS$_{Full}$ on TUS, and D^3L and SANTOS$_{Full}$ on SMALL and LARGE benchmarks.

We first evaluated a version of SANTOS that only considers column semantics, SANTOS$_{Col}$, compared against D^3L. On TUS, SANTOS$_{Col}$ has $MAP@60$ of 65% and $P@60$ of 62%, which is comparable to D^3L results. When we included relationship semantics and evaluated full SANTOS against D^3L, however, SANTOS performs better than D^3L over all measures in all benchmarks. On TUS, SANTOS outperforms D^3L by over 25%, 20% and 19% in terms of $MAP@60$, $P@60$, and $R@60$, respectively. On SMALL, SANTOS outperforms D^3L by over 78%, 56%, and 61% for $MAP@10$, $P@10$, and $R@10$ respectively. In terms of $R@k$, SANTOS is closer to ideal recall than the baseline in both benchmarks. This indicates that relationship semantics is important in union search.
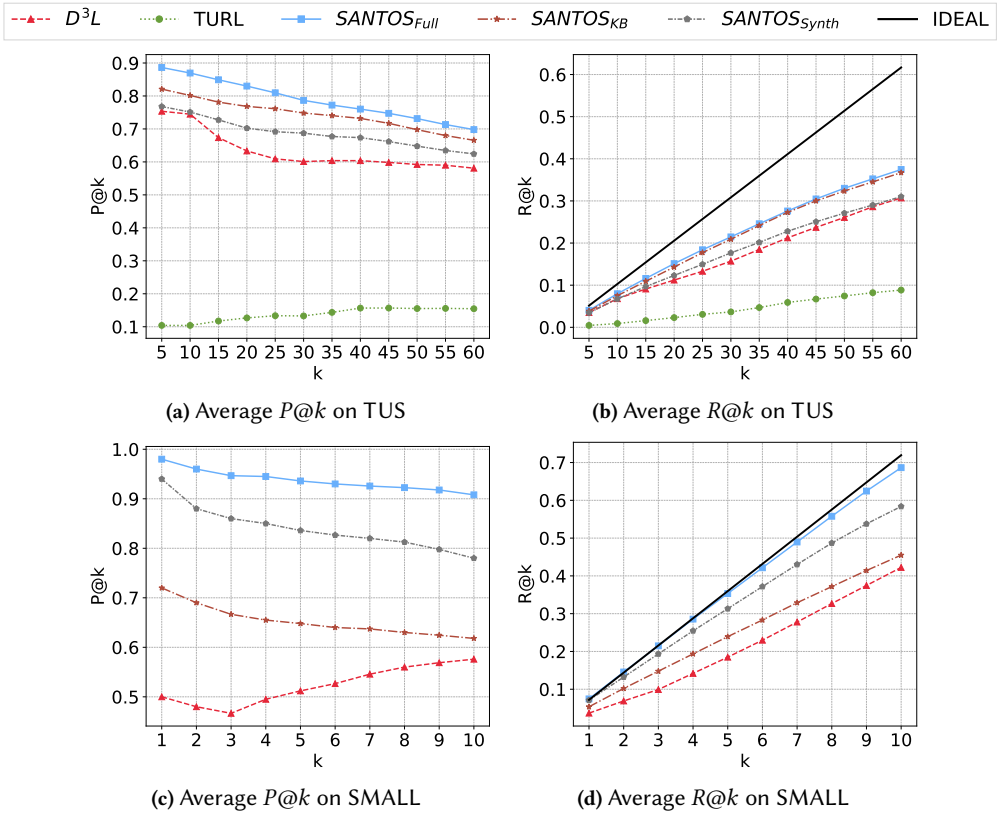
---

[13]https://github.com/alex-bogatu/d3l

Fig. 8. Effectiveness of SANTOS and its variations against baselines in different benchmarks

On a real data lake (LARGE), we observe even further improvements ($MAP@20$ and $P@20$ by over 165% and 180% compared to D^3L respectively), indicating that relationships are even more important in this benchmark. For illustration, consider a query table Q in LARGE benchmark about biodiversity in different counties (with columns like county_name, animal_scientific_name, documented_year, etc.). For this table, the top-k results by SANTOS contain tables about alpine birds, fish, and trees and the places they are found, which seems to be correct. However, as discussed in Ex. 1, although tables returned by D^3L have common (unionable) columns, they are about different topics with different relationships. For instance, they include tables about emergency hospital admissions after accidents because they contain columns like county (unionable with county_name in Q) that describes the place of accident and year (unionable with documented_year) that describes when the accident took place. For both approaches, we manually verify $MAP@k$ and $P@k$ up to $k = 20$. The raw results for each query table on LARGE by SANTOS and D^3L are available in the supplementary materials. [11]

Figure 8 shows a detailed comparison of precision and recall for different values of k in both TUS and SMALL benchmarks. In these graphs, SANTOS is labeled SANTOS$_{Full}$. The other lines labeled SANTOS will be explained below. For each k, SANTOS outperforms competing methods. Notice that perfect $R@k$ is also plotted in Fig. 8(b) (solid black line). TURL-based implementation has the least MAP ($MAP@60 = 0.1$), precision ($P@60 = 0.15$), and Recall ($R@60 = 0.1$) on TUS. A possible reason for the lower performance of TURL is that it is trained over web tables, which have
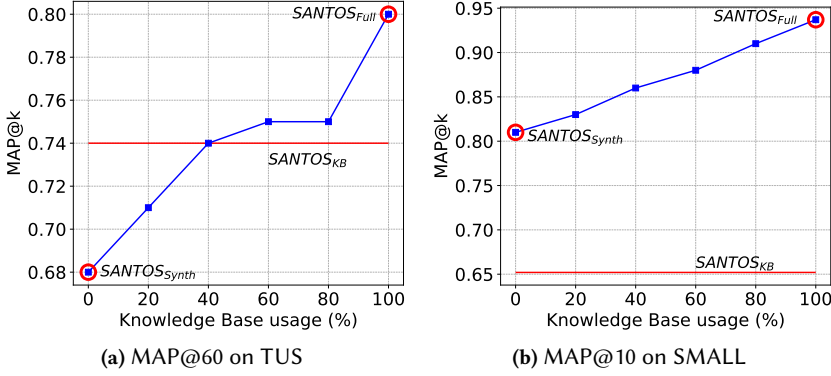
Fig. 9. Average $MAP@k$ of SANTOS$_{\text{Full}}$ ($k = 60$ on TUS, $k = 10$ on SMALL) for different percentages of the existing KB

different characteristics from real open data [29]. We only report TURL performance on TUS as the results in other benchmarks show a similar trend that is well below D^3L and SANTOS.

8.3 SANTOS Effectiveness Ablation Study

We perform an ablation study to understand the impact of one of our key innovations, the use of a synthesized KB, on the accuracy of SANTOS. We compare the full version of SANTOS (SANTOS$_{\text{Full}}$), a version using solely the existing KB (SANTOS$_{\text{KB}}$), and a version only using the synthesized KB (SANTOS$_{\text{Synth}}$) over varying values of k on TUS and SMALL benchmarks. Fig. 8 shows these comparisons in detail. SANTOS$_{\text{Full}}$ obtains the best results over all values of k in both benchmarks. On TUS (Fig. 8 (a) and (b)), SANTOS$_{\text{Synth}}$ (circle/grey line) has lower $P@k$ and $R@k$ than SANTOS$_{\text{KB}}$ because it is not able to retrieve enough results for all the query tables. However, using the synthesized KB with the existing KB together (SANTOS$_{\text{Full}}$) provides the best performance. We see a different trend on SMALL (Fig. 8 (c) and (d)) where SANTOS$_{\text{KB}}$ was not able to return enough results. Specifically, the existing KB had no coverage for 14 of the 50 query tables. However, SANTOS$_{\text{Synth}}$ was able to handle those queries and hence, maintain the overall performance of SANTOS$_{\text{Full}}$. This suggests that the synthesized KB helps SANTOS alleviate the effect from the imperfect coverage of a KB. Note that we ran our experiments using an open KB over open data tables. YAGO may cover less (or more) entities in an enterprise data lake, but enterprises generally maintain their own domain-specific KB's [18, 44, 45]. SANTOS can easily be adapted to such data lakes by augmenting YAGO with the respective enterprise KBs.

To better understand the contribution of the KB and synthesized KB, we compute the average $MAP@k$ of SANTOS$_{\text{Full}}$ by varying the percentage of information from the existing KB randomly. We systematically remove portions of the existing KB entities that are in the data lake tables and evaluate how the synthesized KB compensates for the loss of KB coverage. Fig. 9 shows this analysis on the TUS and SMALL benchmarks for $k = 60$ and $k = 10$, respectively. We first turn off the existing KB and compute $MAP@k$ for SANTOS$_{\text{Synth}}$. Then we gradually increase the existing KB coverage until we reach 100%. In both benchmarks, increasing the usage of the KB increases effectiveness of SANTOS almost linearly, empirically validating that SANTOS performs better with more KB coverage. Furthermore, it shows the significance of both SANTOS$_{\text{KB}}$ and SANTOS$_{\text{Synth}}$ where, SANTOS$_{\text{KB}}$ increases SANTOS$_{\text{Full}}$'s MAP by 18% on TUS and 15% on SMALL and SANTOS$_{\text{Synth}}$ increases it by 8% on TUS and 43% on SMALL. This shows that SANTOS$_{\text{Synth}}$ alone has decent accuracy (68% on TUS and 81% on SMALL), and incrementally adding entries from the KB improves

the accuracy further in a near-linear trend. Thus, leveraging relationship semantics for union search is benefited by the use of both KB's.

Recall that, SANTOS only uses cell values to create the semantic graphs. So, the accuracy of the created graph may depend on the number of rows in the query tables i.e., fewer rows may impact its accuracy. However, even a human expert would need an adequate number of rows to understand the table semantics and so does SANTOS. As reported in Fig. 6, the data lake tables generally contain thousands of rows and hence, SANTOS is fairly effective in understanding their semantics as reported in our experiments.

8.4 SANTOS Scalability

Benchmark	Method	Indexing	Query (sec)
TUS	D^3L	1 hr 21 min	54.1 (20.5 − 97.3)
	SANTOS$_{Full}$	4 hr 26 min	22.9 (1.7 − 48.6)
	SANTOS$_{KB}$	1 hr 38 min	6.1 (0.7 − 13.9)
	SANTOS$_{Synth}$	3 hr 45 min	15.6 (0.7 − 43.2)
SMALL	D^3L	17 min	22.4 (7.4 − 43.3)
	SANTOS$_{Full}$	4 hr 46 min	28.2 (0.8 − 102)
	SANTOS$_{KB}$	1 hr 8 min	10.0 (0.3 − 33.6)
	SANTOS$_{Synth}$	3 hr 41 min	18.2 (0.5 − 98.6)
LARGE	D^3L	7 hr 7 min	177 (13.0 − 325.0)
	SANTOS$_{Full}$	21 hr 59 min	35.8 (0.21 − 57.2)

Fig. 10. Comparison of Indexing and Query times of SANTOS$_{Synth}$, SANTOS$_{KB}$, SANTOS$_{Full}$, and D^3L in different benchmarks. For Query time, we report both the average query time and in parenthesis we include 80% confidence interval (10% to 90% percentile).

As a final analysis, we experimentally validate that SANTOS scales to large data lakes. We report indexing and query times for SANTOS (SANTOS$_{Full}$), the two variations of SANTOS (SANTOS$_{KB}$ and SANTOS$_{Synth}$) and D^3L on all three benchmarks (Fig. 10). For discussion, we focus on LARGE as it is largest in size.

Although D^3L is around 3x faster than SANTOS when indexing data lake tables (Sec. 7.3), the relationship-based approach of SANTOS proves to be significantly more effective than the column-based approach of D^3L. Nevertheless, SANTOS is much faster than D^3L at query time (Sec. 7.4). In LARGE, SANTOS takes 7 sec on average to index each data lake table, and thus is able to scale to tables with thousands of rows and columns (see Fig. 6). The individual index creation time of SANTOS$_{Synth}$, which includes the time taken to discover FDs, is 17 hr 19 min. Recall that SANTOS may be further effective if the tables do not contain homographs. Using the state-of-the-art technique [22], the time taken to detect the homographs is 17 min on the TUS Benchmark. So, we can detect homographs in fairly small and feasible pre-processing time. Notice however, the absence of homographs is not a necessary condition and SANTOS is still more effective than baselines without this pre-processing step (we report the effectiveness results without homograph removal in Sec. 8.2). Similarly, SANTOS$_{KB}$'s index creation time is 12 hr 40 min. One can create these indexes in parallel, so SANTOS$_{Full}$ indexing time can alternatively be max(17 hr 19 min, 12 hr 40 min) = 17 hr 19 min plus few seconds to combine the indexes (rather than 21 hr 59 min reported in Fig. 10). The synthesized KB created over LARGE uses 3588 MB of main memory at query time and occupies 3441 MB in the secondary storage. [9] This further validates SANTOS's scalability. In the future work, we will study other spaces for optimizing the creation of synthesized KB.

We analyze 125 Query Tables on TUS, 50 Query Tables on SMALL and 80 query tables on LARGE, for which we report the average, 10th, and 90th percentile query times. Notice that the query time can vary depending on the complexity of the query table. SANTOS average query times are faster than D^3L query time, since D^3L searches through each of their five indexes when finding unionable tables, whereas we only search through synthesized KB index, YAGO index, or both. Also, the query time of D^3L and SANTOS are comparable on SMALL because D^3L is faster on smaller data lakes. However, SANTOS is faster than D^3L for larger data lakes as suggested by the average query time on TUS (over 3 times faster) and LARGE (almost 6 times faster). All in all, even with the new synthesized KB, our indexing and query times are comparable or even faster than those of a state-of-the-art approach.

9 CONCLUSION

We presented SANTOS, a method for finding unionable tables in data lakes based on both column and relationship semantics. SANTOS discovers and uses relationship semantics between pairs of columns in a table using an existing knowledge base (KB) and a synthesized KB created by exploiting the knowledge of the data lake. We conducted experiments on an adapted version of the existing TUS benchmark as well as our new SMALL and LARGE benchmarks and showed that SANTOS unionability search outperforms a state-of-the-art table union approach. Also, the experimental results showed the robustness of our approach and the importance of our synthesized KB in overcoming curated KBs with limited coverage.

ACKNOWLEDGMENTS

REFERENCES

[1] Marco D. Adelfio and Hanan Samet. 2013. Schema Extraction for Tabular Data on the Web. *Proc. VLDB Endow.* 6, 6 (apr 2013), 421–432. https://doi.org/10.14778/2536336.2536343

[2] Akiko Aizawa. 2003. An information-theoretic perspective of tf–idf measures. *Information Processing and Management* 39, 1 (2003), 45–65. https://doi.org/10.1016/S0306-4573(02)00021-3

[3] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *ICDE 2020*. IEEE, 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[4] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *WWW 2019*. ACM, 1365–1375. https://doi.org/10.1145/3308558.3313685

[5] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. *Proc. VLDB Endow.* 2, 1 (aug 2009), 1090–1101. https://doi.org/10.14778/1687627.1687750

[6] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *SIGMOD 2020*. Association for Computing Machinery, 1335–1349. https://doi.org/10.1145/3318464.3389742

[7] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding Related Tables. In *SIGMOD 2012*. Association for Computing Machinery, 817–828. https://doi.org/10.1145/2213836.2213962

[8] 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. https://doi.org/10.14778/3430915.3430921

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*. ACL, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[10] Mina Farid, Alexandra Roatis, Ihab F. Ilyas, Hella-Franziska Hoffmann, and Xu Chu. 2016. CLAMS: Bringing Quality to Data Lakes. In *SIGMOD 2016*. Association for Computing Machinery, 2089–2092. https://doi.org/10.1145/2882903.2899391

[11] Raul Castro Fernandez, Essam Mansour, Abdulhakim Ali Qahtan, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery. *ICDE* (2018), 989–1000. https://doi.org/10.1109/ICDE.2018.00093

[12] Peter A. Flach and Iztok Savnik. 1999. Database Dependency Discovery: A Machine Learning Approach. *AI Commun.* 12, 3 (1999), 139–160. http://content.iospress.com/articles/ai-communications/aic182

[13] D Frank Hsu and Isak Taksa. 2005. Comparing rank and score combination methods for data fusion in information retrieval. *Information retrieval* 8, 3 (2005), 449–480. https://doi.org/10.1007/s10791-005-6994-4

[14] Sainyam Galhotra and Udayan Khurana. 2020. Semantic Search over Structured Data. In *CIKM 2020*. Association for Computing Machinery, 3381–3384. https://doi.org/10.1145/3340531.3417426

[15] Parantapa Goswami, Eric Gaussier, and Massih-Reza Amini. 2017. Exploring the space of information retrieval term scoring functions. *Information Processing and Management* 53, 2 (2017), 454–472. https://doi.org/10.1016/j.ipm.2016.11.003

[16] Vinh Thinh Ho, Yusra Ibrahim, Koninika Pal, Klaus Berberich, and Gerhard Weikum. 2019. Qsearch: Answering Quantity Queries from Text. In *ISWC*. Springer-Verlag, 237–257. https://doi.org/10.1007/978-3-030-30793-6_14

[17] Vinh Thinh Ho, Koninika Pal, and Gerhard Weikum. 2021. QuTE: Answering Quantity Queries from Web Tables. In *SIGMOD* (Virtual Event, China). Association for Computing Machinery, New York, NY, USA, 2740–2744. https://doi.org/10.1145/3448016.3452763

[18] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *ACM Comput. Surv.* 54, 4, Article 71 (July 2021), 37 pages. https://doi.org/10.1145/3447772

[19] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çağatay Demiralp, and César Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *KDD 2019*. ACM, 1500–1508. https://doi.org/10.1145/3292500.3330993

[20] Oliver Lehmberg and Christian Bizer. 2017. Stitching Web Tables for Improving Matching Quality. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1502–1513. https://doi.org/10.14778/3137628.3137657

[21] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A Large Public Corpus of Web Tables Containing Time and Context Metadata. In *WWW 2016 Companion*. International World Wide Web Conferences Steering Committee, 75–76. https://doi.org/10.1145/2872518.2889386

[22] Aristotelis Leventidis, Laura Di Rocco, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2021. DomainNet: Homograph Detection for Data Lake Disambiguation. In *EDBT 2021*. OpenProceedings.org, 13–24. https://doi.org/10.5441/002/edbt.2021.03

[23] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 1338–1347. https://doi.org/10.14778/1920841.1921005

[24] Xiao Ling, Alon Y. Halevy, Fei Wu, and Cong Yu. 2013. Synthesizing Union Tables from the Web. In *IJCAI 2013*. IJCAI/AAAI, 2677–2683. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6758

[25] H. P. Luhn. 1957. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development* 1, 4 (1957), 309–317. https://doi.org/10.1147/rd.14.0309

[26] David JC MacKay. 2003. *Information theory, inference and learning algorithms.* Cambridge university press. https://books.google.com/books?id=AKuMj4PN_EMC

[27] C.D. Manning, P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press. https://books.google.com/books?id=t1PoSh4uwVcC

[28] Suvodeep Mazumdar and Ziqi Zhang. 2016. Visualizing Semantic Table Annotations with TableMiner+. In *ISWC 2016*, Vol. 1690. CEUR-WS.org. http://ceur-ws.org/Vol-1690/paper88.pdf

[29] Renée J. Miller. 2018. Open Data Integration. *Proc. VLDB Endow.* 11, 12 (aug 2018), 2130–2139. https://doi.org/10.14778/3229863.3240491

[30] Renée J. Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q. Pu, and Periklis Andritsos. 2018. Making Open Data Transparent: Data Discovery on Open Data. *IEEE Data Eng. Bull.* 41, 2 (2018), 59–70. http://sites.computer.org/debull/A18june/p59.pdf

[31] Varish Mulwad, Tim Finin, Zareen Syed, and Joshi Anupam. 2010. Using linked data to interpret tables. In *Proceedings of the the First International Workshop on Consuming Linked Data*, Vol. 665. CEUR-WS.org. http://ceur-ws.org/Vol-665/MulwadEtAl_COLD2010.pdf

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

[33] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* 11, 7 (March 2018), 813–825. https://doi.org/10.14778/3192965.3192973

[34] Masayo Ota, Heiko Müller, Juliana Freire, and Divesh Srivastava. 2020. Data-Driven Domain Discovery for Structured Datasets. *Proc. VLDB Endow.* 13, 7 (March 2020), 953–967. https://doi.org/10.14778/3384345.3384346

[35] Christos H Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. 2000. Latent semantic indexing: A probabilistic analysis. *J. Comput. System Sci.* 61, 2 (2000), 217–235. https://doi.org/10.1006/jcss.2000.1711

[36] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. 2020. YAGO 4: A Reason-able Knowledge Base. In *ESWC*. Springer International Publishing, 583–596. https://doi.org/10.1007/978-3-030-49461-2_34

[37] Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. Citeseer, 29–48.

[38] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML Tables to DBpedia. In *WIMS 2015*. Association for Computing Machinery, Article 10, 6 pages. https://doi.org/10.1145/2797115.2797118

[39] Karen Sparck Jones. 1988. *A Statistical Interpretation of Term Specificity and Its Application in Retrieval*. Taylor Graham Publishing, GBR, 132–142.

[40] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-Trained Language Models. In *SIGMOD 2022*. Association for Computing Machinery, 1493–1503. https://doi.org/10.1145/3514221.3517906

[41] Zareen Syed, Tim Finin, Varish Mulwad, and Anupam Joshi. 2010. Exploiting a Web of Semantic Data for Interpreting Tables. In *Proceedings of the Second Web Science Conference*. ACM. https://ebiquity.umbc.edu/paper/html/id/474

[42] Kunihiro Takeoka, Masafumi Oyamada, Shinji Nakadai, and Takeshi Okadome. 2019. Meimei: An Efficient Probabilistic Approach for Semantically Annotating Tables. In *AAAI 2019*. AAAI Press, 281–288. https://doi.org/10.1609/aaai.v33i01.3301281

[43] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering Semantics of Tables on the Web. *Proc. VLDB Endow.* 4, 9 (June 2011), 528–538. https://doi.org/10.14778/2002938.2002939

[44] Muhammad Yahya, John G. Breslin, and Muhammad Intizar Ali. 2021. Semantic Web and Knowledge Graphs for Industry 4.0. *Applied Sciences* 11, 11 (2021). https://doi.org/10.3390/app11115110

[45] Nasser Zalmout, Chenwei Zhang, Xian Li, Yan Liang, and Xin Luna Dong. 2021. All You Need to Know to Build a Product Knowledge Graph. In *KDD 2021*. Association for Computing Machinery, 4090–4091. https://doi.org/10.1145/3447548.3470825

[46] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.* 13, 12 (2020), 1835–1848. https://doi.org/10.14778/3407790.3407793

[47] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *SIGMOD 2020*. Association for Computing Machinery, 1951–1966. https://doi.org/10.1145/3318464.3389726

[48] Ziqi Zhang. 2017. Effective and efficient Semantic Table Interpretation using TableMiner[+]. *Semantic Web* 8, 6 (2017), 921–957. https://doi.org/10.3233/SW-160242

[49] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proc. VLDB Endow.* 9, 12 (Aug 2016), 1185–1196. https://doi.org/10.14778/2994509.2994534