Finding Related Tables

Anish Das Sarma[♯], Lujun Fang[†], Nitin Gupta[♯], Alon Halevy[♯],
Hongrae Lee[♯], Fei Wu[♯], Reynold Xin[‡], Cong Yu[♯]
[†] University of Michigan, [♯] Google Inc., [‡] University of California, Berkeley
ljfang@umich.edu, {anish, nigupta, halevy, hrlee, wufei,congyu}@google.com,
rxin@cs.berkeley.edu

ABSTRACT

We consider the problem of finding related tables in a large corpus of heterogenous tables. Detecting related tables provides users a powerful tool for enhancing their tables with additional data and enables effective reuse of available public data. Our first contribution is a framework that captures several types of relatedness, including tables that are candidates for joins and tables that are candidates for union. Our second contribution is a set of algorithms for detecting related tables that can be either unioned or joined. We describe a set of experiments that demonstrate that our algorithms produce highly related tables. We also show that we can often improve the results of table search by pulling up tables that are ranked much lower based on their relatedness to top-ranked tables. Finally, we describe how to scale up our algorithms and show the results of running it on a corpus of over a million tables extracted from Wikipedia.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General

General Terms

Algorithms, Design, Management, Performance

Keywords

web tables, related tables, data integration

1. INTRODUCTION

Several online services are pursuing the vision of creating repositories of high quality structured data [19, 4, 2, 5]. The data sources in the repository may either be contributed by users directly or extracted from the Web.

The main benefit of creating such repositories is to fuel data integration, by facilitating the discovery and reuse of existing data sets. For example, an economics student creating a data set with economic indicators for a particular country should be able to easily find data about the population and GDP of that country to add as columns in her table, or data about economic indicators in neighboring countries to add as new rows.

To realize this vision, we must provide users effective means to explore the data sets available, and decide which data sets fit their needs in terms of content, coverage, and quality. Importantly, the search for related content should be part of the natural workflow the user follows. For example, if the user is looking at a particular table, she should be able to simply type in a keyword describing a column she wants to add to the table.

In Google Fusion Tables we are investigating a variety of mechanisms for exploring data sets. In the simplest case, we provide keyword search over the repository of public data sets, while in another we provide search in the context of an existing table.

Regardless of the input to the search problem, we are faced with a fundamental problem of discovering *related* tables in a vast collection of heterogeneous data. This paper describes a framework for defining relatedness of tables and algorithms for finding related tables.

The problem is challenging for two main reasons. First, the schemas of the tables in the repository are partial at best and are extremely heterogeneous. In some cases the crucial aspects of the schema that are needed for reasoning about relatedness are embedded in text surrounding the tables or textual descriptions attached to them. Second, one needs to consider different ways and degrees to which data can be related. The following examples illustrate some of the challenges.

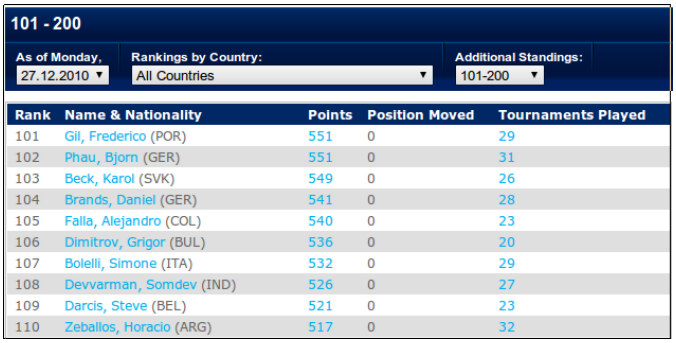
1 - 100

As of Monday, 27.12.2010 | Rankings by Country: All Countries | Additional Standings: Top 100

Rank	Name & Nationality	Points	Position Moved	Tournaments Played
1	Nadal, Rafael (ESP)	12,450	0	20
2	Federer, Roger (SUI)	9,145	0	21
3	Djokovic, Novak (SRB)	6,240	0	21
4	Murray, Andy (GBR)	5,760	0	19
5	Soderling, Robin (SWE)	5,580	0	24
6	Berdych, Tomas (CZE)	3,955	0	26
7	Ferrer, David (ESP)	3,735	0	24
8	Roddick, Andy (USA)	3,665	0	21
9	Verdasco, Fernando (ESP)	3,240	0	25
10	Youzhny, Mikhail (RUS)	2,920	0	24

Figure 1: 2010 Men Tennis Top 100 from ATP World Tour

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.



101 - 200

As of Monday, 27.12.2010 | Rankings by Country: All Countries | Additional Standings: 101-200

Rank	Name & Nationality	Points	Position Moved	Tournaments Played
101	Gil, Frederico (POR)	551	0	29
102	Phau, Bjorn (GER)	551	0	31
103	Beck, Karol (SVK)	549	0	26
104	Brands, Daniel (GER)	541	0	28
105	Falla, Alejandro (COL)	540	0	23
106	Dimitrov, Grigor (BUL)	536	0	20
107	Bolelli, Simone (ITA)	532	0	29
108	Devvarman, Somdev (IND)	526	0	27
109	Darcis, Steve (BEL)	521	0	23
110	Zeballos, Horacio (ARG)	517	0	32

Figure 2: 2010 Men Tennis 100 - 200 from ATP World Tour

2010 Men's Tennis ATP Rankings

Year: 2010 | Type: ATP Rankings

Men's Singles Rankings

RK	PLAYER	COUNTRY	MOVEMENT	POINTS
1	Rafael Nadal		0	12450
2	Roger Federer		0	9145
3	Novak Djokovic		0	6035
4	Andy Murray		1	5760
5	Robin Soderling		1	5580
6	Tomas Berdych		0	3955
7	David Ferrer		0	3735
8	Andy Roddick		0	3665
9	Fernando Verdasco		0	3240
10	Mikhail Youzhny		0	2920

Figure 3: 2010 Men Tennis Top 100 from ESPN

Consider the tables in Figure 1 and 2. The first describes the top 100 men tennis players and the second describes the next top 100 performers. These two tables are related: their schema is identical, and they provide complementary sets of entities. Their *union* would produce a meaningful table.

The table in Figure 3 is also related to the table in Figure 1. The two tables describe the same entities, but the table in Figure 1 adds more information (i.e., columns) about each entity, such as *Tournaments Played*. The *join* of the two tables would produce a meaningful table. Note that even in this simple example, the system would have to determine that the two tables are *about* the tennis players in order to compare the set of attributes in a meaningful fashion.

The table in Figure 4 is also related to the table in Figure 3, but the relationship is a bit more subtle. The two tables describe the top tennis players from 2009 and 2010, respectively. While the tables cannot be directly joined or unioned, they both can be seen as the result of a selection and projection on a larger table that contains the rankings of tennis players in different years. Note that the year that the tables' data refers to is not part of the table itself, but needs to be inferred from the context.

Inspired by these examples, this paper makes the following contributions. We describe a framework that captures different kinds of relatedness. The key idea is that tables are considered related to each other if they can be viewed as results to queries over the same (possibly hypothetical) original table. Second, we present algorithms for detecting

2009 Men's Tennis ATP Rankings

Year: 2009 | Type: ATP Rankings

Men's Singles Rankings

RK	PLAYER	COUNTRY	MOVEMENT	POINTS
1	Roger Federer		0	10550
2	Rafael Nadal		0	9205
3	Novak Djokovic		0	8310
4	Andy Murray		0	7030
5	Juan Martin del Potro		0	6785
6	Nikolay Davydenko		0	4930
7	Andy Roddick		0	4410
8	Robin Soderling		0	3410
9	Fernando Verdasco		0	3300
10	Jo-Wilfried Tsonga		0	2875

Figure 4: 2009 Men Tennis Top 100 from ESPN

tables that are *entity complement* and therefore are candidates to be unioned. The crux of the algorithms is to determine that the entities in a table T_2 are a coherent expansion of the entities in a table T_1. (E.g., T_2 describes a same or similar concept as T_1 does.) Based on the same ideas, we describe an algorithm for detecting tables that are *schema complement*, thereby candidates for a join.

Next, we describe experiments showing the effectiveness of our algorithms and providing an evaluation of the different components of the algorithms. We also show that discovering related tables can also improve table search. In particular, we show that tables that are related to top-ranked tables but that do not appear in the top results are often judged to be on par with top ranked results. Hence, discovering related tables can provide a semantics-based method to improve table search. Finally, we discuss how to scale up the computation of related tables to large table corpora and demonstrate the result on the corpus of over 1 million tables extracted from Wikipedia.

Section 2 proposes a framework for defining relatedness of tables. Section 3 and 4 then describe the algorithms to measure entity complement and schema complement respectively. We describe our experiments in Section 5, and how to scale up the computation of related tables in Section 6. We review the related work in Section 7 and conclude in Section 8.

2. PROBLEM DEFINITION

We assume a large corpus of heterogenous tables $\mathcal{T}$, such as the collection of HTML tables found on the Web [11, 24]. The quality of the tables in $\mathcal{T}$ varies a lot, and we usually have only partial meta-data about each table. For instance, we may only have a guess at the column headers, and the relations represented by the table need to be inferred from cell values and the surrounding text.

Given the corpus $\mathcal{T}$ and a table T_1, our goal is to return a ranked list of tables in $\mathcal{T}$ that are *related* to T_1. As we saw in the examples, tables can be related to each other in a variety of ways. However, the common theme underlying all the notions of relatedness is that tables T_1 and T_2 are related if they include content that conceivably could have been in a *single* table T. This observation is the basis for the framework we propose for measuring relatedness of tables:

- A pair of tables T_1 and T_2 is said to be related if we can identify a virtual table T such that T_1 and T_2 are the results of applying two queries, Q_1 and Q_2, respectively, over T. The schema of T_1 (resp. T_2) may involve renaming of the attributes of $Q_1(T)$ (resp. $Q_2(T)$).

- The table T should be *coherent*. For example, we could decide that a table storing the prices of tea in China is related to the table with the winners of the Boston Marathon, because in principle we can imagine a table T that stores both. However, T would not be coherent by any reasonable design principle. In contrast, a table storing the ranks of the top tennis players in the world in the past 10 years is coherent, and therefore the table with the top ranked players in 2011 is related to the table with the top players in 2010.

- The queries Q_1 and Q_2 should have *similar structure*. For example, they can both be projections on T or both be selections on T, or same sequence of selections and projections on T (although the selection or projection conditions can be different). As we see below, different structures of Q_1 and Q_2 correspond to specific types of related tables.

The above framework captures the vast majority of related tables we see in practice. In our paper, we consider two most common types of related tables: *Entity Complement* and *Schema Complement*, resulting from applying different *selection* or *projection* conditions in similarly structured queries, respectively, over the same underlying virtual table. Our definitions of entity and schema complements are asymmetric, to address the differences between the queried table and the result tables. In a sense, combining related tables can be viewed as reverse-engineering vertical/horizontal fragmentation in distributed DBMS. However, since web tables are noisy in nature, the requirements here are more flexible: for example, overlap between related tables or renaming of attributes should be allowed.

DEFINITION 1. **Entity Complement (EC)**. *Table $T_2 \in \mathcal{T}$ is entity complement to $T_1 \in \mathcal{T}$ if there exists a coherent virtual table T, such that $Q_1(T) = T_1$ and $Q_2(T) = T_2$, where:*

1. *Q_i takes the form $Q_i(T) = \sigma_{P_i(X)}(T)$, where X contains a set of attributes in T and P_i is a selection predicate over X.*

2. *The combination of Q_1 and Q_2 cover all the tuples in T, and Q_2 covers some tuples not covered by Q_1.*

3. *Optionally, each Q_i renames or projects a set of attributes A (same for different Q_i) with the restriction that $\exists A' \subseteq A, A' \rightarrow X$ in T.*

In other words, T_1 and T_2 are obtained by applying different selection predicates P_1 and P_2 on the same set of attributes X in T, and apply projections that include the key attributes with respect to X. The tables in Figures 1 and 2 are entity complement to each other, since we can have a virtual table T containing top-200 men tennis players in 2010 and apply selection conditions over the "Rank" attribute. Note the attribute set A to be projected does not need to contain all the attributes in X as long as $\exists A' \subseteq A, A' \rightarrow X$ in T. For example, tables in Figure 1 and 2 are entity complements to each other even if the *Rank* attribute is not projected, since the "Rank" attribute can be inferred from "Player" attribute in T given that each player has a fixed ranking in 2010.

The relatedness of two tables depends on how close the selection conditions P_1 and P_2 are to each other. The closeness of the selection conditions can be approximated by the degree of coherence of entities in T_1 and T_2 (to be discussed in detail in Section 3.1). For example, the table about South American countries is more related to that of North American countries than the table with Asian countries.

Note that entity complement tables T_1 and T_2 can be union-ed in a "lossless" fashion over the common attributes (possibly hidden but inferrable). More formally, we have that $\Pi_X(T_1') \cup \Pi_X(T_2') = \sigma_{P_1(X) \vee P_2(X)} \Pi_X(T)$, where T_i' is augmented T_i with derivable attributes.

DEFINITION 2. **Schema Complement (SC)**. *Table $T_2 \in \mathcal{T}$ is schema complement to $T_1 \in \mathcal{T}$ if there exists a coherent virtual T, such that $Q_1(T) = T_1$ and $Q_2(T) = T_2$ where:*

1. *Q_i takes the form $Q_i(T) = \Pi_{A_i}(T)$, where A_i is the set of attributes (with optionally renaming) to be projected.*

2. *$A_2 \setminus A_1 \neq \emptyset$, $A_1 \cup A_2$ covers all T's attributes, and $A_1 \cap A_2$ covers key attributes of A_1 and A_2 (i.e., $\exists X \subseteq A_1 \cap A_2, X \rightarrow A_i$).*

3. *Optionally, each Q_i applies a fixed selection predicate P over the set of key attributes X.*

In other words, T_2 contains the same set of entities (due to identical selection conditions) as T_1 does, for a different and yet semantically related set of attributes. The table in Figures 1 is schema complement to the table in Figure 3. Note that schema complements allow us to perform lossless joins: $T_1 \bowtie T_2 = \Pi_{A_1 \cup A_2} \sigma_{P(X)}(T)$.

We will focus our discussion of relatedness on entity complement (Section 3) and schema complement (Section 4). However, our framework is flexible enough to incorporate other types of relatedness. For example, the relationship between the tables in Figures 3 and 4 involves queries Q_1 and Q_2 that differ in their selection condition on attribute `year` that is not inferable from the projected attributes. So, in contrast to the entity complement condition above, here T_1 and T_2 didn't retain all attributes to infer X, resulting in a "lossy union". For example, same players in two tables can have different points, which are not explained by the attributes in the two tables. This distinction shows that looking at consistency of values across the two tables is a critical component in detection of entity complement, as we shall study in more detail later.

Note that different relatedness types are not mutually exclusive. Furthermore, the context of the search for related tables can often dictate what kind of relationship the user is looking for. For example, the user may be explicitly searching to add rows to her table, in which case entity complement tables should be proposed. One of the benefits of our framework is to recognize the different kinds of relatedness and point them out when multiple ones apply.

In summary the problem addressed by this paper is as follows: Given a corpus of tables $\mathcal{T}$, a query table T, a constant k, a relatedness type R, select k tables, $T_1, T_2, \ldots, T_k \in \mathcal{T}$, with the highest relatedness scores of type R to T.

3. ENTITY COMPLEMENT TABLES

This section considers the problem of finding a ranked list of entity-complement tables to an input table T_1. As a pre-processing step, we implemented a rule-based and machine-learned classifier from [30] for the detection of *header rows* (*schema rows*) and *subject columns* (a column contains the entities the table is about) for all tables in the corpus. For example, in the table in Figure 3, the pre-processing step detects "Player" column as the subject column and the the attribute names as the header row. When we cannot detect a subject column we return no related tables, since both entity and schema complement definitions require existences of key attributes. Following our definitions, our algorithm is guided by the following criteria:

Entity Consistency: We would like a related table T_2 to have the same type of entities as T_1, as required by the coherence of the vitual table T and closeness of Q_1 and Q_2 in Definition 1. E.g., entities in Figures 1 and Figure 2 are both active men tennis players in 2010. We use E_i or $E(T_i)$ to represent the set of entities described in T_i. In particular, they are the cell contents of T_i's subject column.

Entity Expansion: T_2 should substantially add new entities to those in T_1 (e.g., the players in Figure 2 are different from those in in Figure 1), as required by Point 2 in Definition 1.

Schema Consistency: The two tables T_1 and T_2 should have similar (if not the same) schemas, thereby describing similar properties of the entities, as required by Point 3 in Definition 1.

We describe our entity consistency and expansion measures in Section 3.1. Section 3.2 describes schema consistency. Section 3.3 describes how the individual scores of these components are combined into one measure.

3.1 Entity Consistency and Expansion

The main challenge in constructing a single measure for the relatedness of two tables' (T_1 and T_2's) entity sets is the inherent trade-off between entity consistency and entity expansion: Adding more entities expands the initial set but may compromise its consistency.

EXAMPLE 1. *Consider the following sets of entities:*

- $E(T_1) = \{India, Korea, Malaysia\}$
- $E(T_2) = \{Japan\}$
- $E(T_3) = \{Canada, United\ States\}$
- $E(T_4) = \{Japan, China\}$
- $E(T_5) = \{Malaysia, Japan, China, Thailand, Canada\}$.

Each of the tables T_2, T_3, T_4, T_5 describe additional entities to $E(T_1)$; therefore, to some degree all of them are entity complements. Clearly, $E(T_1)$ includes a set of Asian countries. Therefore, it is reasonable to assert that $E(T_2)$ is more related to $E(T_1)$ than $E(T_3)$. Similarly, we could deduce that $E(T_4)$ is even better than $E(T_2)$ since it provides a larger (and related) set of entities. The comparison between $E(T_5)$, $E(T_2)$ and $E(T_4)$ is less clear. T_5 has the largest set of Asian countries, but also contains a non-Asian country.

In the remainder of this section, we start by discussing the sources of signals we use to compute a consistency score. Then, we present concrete entity consistency score definitions. We consider two high level approaches: (1) For each additional entity in T_2, compute its relatedness to each entity in T_1, and then aggregate the pairwise entity relatedness; (2) Take the set of additional entities in T_2 as a whole, and directly compute its consistency with respect to T_1. We also discuss how to capture the amount of expansion and combine it with the entity consistency score.

Sources of signal

The problem of determining relatedness of entity sets would be simplified if there was a very detailed classification of all the entities in the world. In the example of deciding the relatedness of two countries, we would need not only classification of countries by continent, but also by parts of continents (e.g., south-east Asian countries tend to be grouped quite often). We would also need groupings based on other aspects, such as coffee-producing countries or mountainous countries. Since no such detailed classification exists, we infer entity groups based on several signals we can mine from the Web. In particular, we consider three sources of signal:

WebIsA database: WebIsA (used by [30], implemented using the techniques mentioned in [26]) is a database of entities and their classes, e.g., (Paris, City), constructed from mentions of entities in text. This dataset has a high coverage/recall since it includes even fine-granularity clusters mentioned in text documents, such as south-east asian countries. On the other hand, WebIsA contains a lot of noise because the extraction techniques are far from perfect. Given a name of an entity, the WebIsA database will return a set of classes that it considers the entity to be a member of. The names of these classes are the labels we use in our algorithm, and we call them WebIsA labels. Using WebIsA we obtained around $1.5M$ labeled subject columns and ∼155M instances [30].

Freebase types: Freebase [3] is a curated database of entities types and properties. Freebase typically has high precision though significantly lower coverage compared to WebIsA. Given an entity, a search in Freebase returns a set of Freebase types that the entity may be a member of. We call these types Freebase labels. Using Freebase, we obtained around $600K$ labeled subject columns and ∼16M instances. Throughout this paper, we also assume uniqueness of Freebase ids, which are used for entity resolution; we say that two rows in two tables correspond to the same entity if and only if the cells in their subject column map to the same Freebase entity identifier. In other words, we treat Freebase identifiers as the golden standard for entity resolution; obviously, entity resolution is an orthogonal problem and more sophisticated techniques may be plugged in directly.

Table co-occurrence: We construct labels by counting co-occurrences of entities in tables. Specifically, each Web table T can be regarded as a "label" and an entity has the label T if it appear in T. Note that computing table co-occurrences is much more expensive than the other two label sources, since the number of tables containing an entity are usually larger than the number of WebIsA or Freebase labels an entity has. We refer to these labels as WebTable labels.

Relatedness between a pair of entities

We now discuss how to decide the relatedness of a pair of entities with the signals we discussed above. We assign

weighted labels $L(e_i) = \{l_i^1 : w_i^1, l_i^2 : w_i^2, ...\}$ to each entity e_i, and we represent the vector of labels by $L(\vec{e_i})$. The labels here are a combination of WebIsA labels, Freebase labels and WebTable labels. We then compute the relatedness between e_i and e_j as the dot product of the two vectors:

$$re(e_i, e_j) = L(\vec{e_i}) \cdot L(\vec{e_j}). \quad (1)$$

The dot product captures the simple intuition that entities are more similar if they: (1) share more labels. (2) share labels with large weights. A naive baseline approach to assigning weights on labels would be to simply consider *uniform weights*; with uniform weights, relatedness becomes equivalent to computing the number of common labels:

$$re_u(e_i, e_j) = |\{l | l \in L(e_i) \cap l \in L(e_j)\}| \quad (2)$$

We improve on the above baseline by considering domain sizes of labels to assign weights. Specifically, we consider weights normalized by domain size, capturing the intuition that assigning a label with a very large domain gives less information than on a more specific domain. For example, the label "car" has a smaller domain than "thing", hence is more useful. With weights being the inverse of domain sizes of labels, we obtain weighted relatedness re_w:

$$re_w(e_i, e_j) = \sum_{l \in L(e_i) \cap L(e_j)} \frac{1}{|D(l)|}, \quad (3)$$

where $D(l)$ is the domain of label l. (In the case of a labels from table co-occurrence described above, the domain size is given by the number of entities in the table.) In Section 5, we show the benefits of constructing labels from all three sources and our weighting scheme.

Relatedness between entity sets

Next we present two approaches to computing relatedness between entity sets: (1) a baseline approach that aggregates relatedness of pairs of entities, and (2) an improved measure computing relatedness between sets of entities directly.

To compute the relatedness for two sets of entities E_1 and E_2, a baseline approach simply averages relatedness of all pairs of entities between them:

$$S_{EP}^{AvgPair}(E_1, E_2) = \frac{1}{|E_1||E_2|} \sum_{e_1 \in E_1, e2 \in E_2} re(e_1, e_2). \quad (4)$$

Equation 4 does not capture the amount of expansion obtained from E_2. To capture the amount of expansion of E_2, a different normalization coefficient $\frac{1}{|E_1|}$ other than $\frac{1}{|E_1||E_2|}$ can be used (i.e., we multiply $S_{EP}^{AvgPair}$ by size of E_2):

$$S_{EP}^{SumPair}(E_1, E_2) = \frac{1}{|E_1|} \sum_{e_1 \in E_1, e2 \in E_2} re(e_1, e_2). \quad (5)$$

The drawback of both above equations, however, is that they fail to capture the relatedness of more than pairs of entities. For example, *China* can be related to *Japan* because they are both *Asian countries*, *U.S.* can be related to *Japan* because they are both *developed countries*. However, it is less clear how the set of {*China, U.S.*} is related to *Japan*.

Therefore, we propose computing relatedness of sets of entities directly, but using a similar label-vector idea as for pairs of entities. Suppose we can represent an entity set E_i as $L(E_i) = \{l_i^1 : w_i^1, l_i^2 : w_i^2, ...\}$, then relatedness of two entity sets E_1 and E_2 can be similarly computed as the similarity of label vectors:

$$S_{EP}^{Set}(E_1, E_2) = L(\vec{E_1}) \cdot L(\vec{E_2}). \quad (6)$$

Labels and weights of an entity set are derived from labels and weights of composing entities. A straightforward way to decide the weight of label l for an entity set E_i is to use the average of l's weights over the entities constituting E_i:

$$w(E_i, l) = \frac{1}{|E_i|} \sum_{e_i \in E_i} w(e_i, l). \quad (7)$$

We note that under this weighting method, the result of $S_{EP}^{Set}(E_1, E_2) = L(\vec{E_1}) \cdot L(\vec{E_2})$ is identical to Equation (4). Also note that when $w(e_i, l)$ is always equal to 1 (i.e., uniform label weights), it is equivalent to the majority-vote column label weighting method discussed in [30]. To improve over these two baselines, the weighting method needs to have a non-linear increase of weights, thereby capturing higher-order correlations between k-sets of entities (instead of pairs of entities):

$$w(E_i, l) = \frac{(\sum_{e_i \in E_i} w(e_i, l))^{n_i}}{|E_i|^{m_i}}, (n_i > 1, n_i \geq m_i). \quad (8)$$

The difference between n_2 and m_2 controls how S_{EP}^{Set} captures the amount of entity expansion from E_2 to E_1. When $n_2 = m_2$, the quantity of entity expansion is ignored. E.g., $n_1 = 1, m_1 = 1, n_2 = 1, m_2 = 1$ makes S_{EP}^{Set} equivalent to $S_{EP}^{AvgPair}$. When $n_2 > m_2$, the amount of entity expansion is captured. E.g., $n_1 = 1, m_1 = 1, n_2 = 1, m_2 = 0$ makes S_{EP}^{Set} equivalent to $S_{EP}^{SumPair}$. $n_i > 1 (i = 1, 2)$ captures the non-linear increase of weights for E_1 and E_2.

In our experiments in Section 5, we show that our approach above significantly improves over the baselines.

3.2 Schema Similarity

We now discuss how to compute the schema similarity between the query table T_1 and a candidate related table T_2, denoted $S_{SS}(T_1, T_2)$. Our system implements state-of-the-art schema mapping techniques to obtain a schema similarity score; i.e., schema mappings are computed by a combination of similarity in attribute names (we used Java's SecondString similarity package [1, 12]), data types, and values (we used a variant of Jaccard similarity). Since there's a large body of work on schema mapping [15, 28], we only describe the salient features of our setting.

Use of Labels: Since our tables are extracted from the Web, we often don't have a schema, either because the table was published without schema, or detection of the schema is hard [30]. Therefore, we rely strongly on the column-labels in addition to attribute names. As mentioned earlier, column-labels are generated using WebIsA database as well as Freebase: We aggregate cell-value labels to obtain a set of labels for the column. We then employ a generalized Jaccard similarity measure over the set of labels and the attribute names, where the generalization considers pairwise string similarity (instead of string equality in traditional Jaccard). Intuitively, we consider the set of labels and attribute name on each table to form nodes in a bipartite graph, and pairwise edges representing string similarity; we then compute the max-weight matching to obtain the name similarity.

Schema Mapping Score: We aggregate pairwise attribute matching scores to compute an overall schema mapping score

as follows. Given schemas $S_1(A_1^1, \ldots, A_{n_1}^1)$ and $S_2(A_1^2, \ldots, A_{n_2}^2)$ over tables T_1 and T_2, we first obtain pairwise matching scores between every pair of attributes A_i^1 and A_j^2. Subsequently, we compute an overall one-to-one schema mapping through a *bipartite max-weight matching* between the two sets of attributes: We construct a weighted bipartite graph $G(V_1, V_2, E)$ where the set of vertices V_i correspond to the set of attributes in S_i, and the weighted edge between A_i^1 and A_j^2 gives its attribute matching score. The max-weight bipartite matching then gives the overall schema mapping. Let the weight of this matching be W and let the number of edges in the matching be N. The schema mapping score is then computed as $S_{SS}(T_1, T_2) = \frac{W}{n_1+n_2-N}$; intuitively, we find the overall strength of the mapping by aggregating the strength of the chosen attribute matches and dividing it by the total number of distinct attributes.

Consistency of Values

Recall from Section 2 that value consistency could be a key to distinguish whether two tables are entity complements or the relation between them is more complex. We discuss how to determine whether two tables have *consistent* (i.e., non-conflicting) values. For instance, if two tables have the entity "United States", we expect both tables to have "Washington, DC" in the "capital" attribute. However, this may not always be true due to noisy web data or data recorded at different times (e.g., for attribute President). Furthermore, for numeric attributes like "population", the values may not be exactly the same or may be given using different units. We assume that unit transformations are handled by another module and do not consider them here.

We consider value consistency when two tables share some entities. For tables that contain shared entities, we evaluate their consistency by averaging the similarity in their values over all corresponding value pairs. The similarity of textual fields is obtained by string similarity. For numeric values v_1, v_2, we use a simple numeric similarity: $(1 - \frac{|v_1 - v_2|}{\max\{v_1, v_2\}})$. Rather than computing an independent value consistency score for two tables, we incorporate value consistency as another signal into the schema similarity score: attributes in two tables can be matched only if their value consistent score is larger than a threshold (e.g., 0.8).

3.3 Summary

The entity complement score of T_2 to T_1, $EC(T_1, T_2)$, is computed by combining the entity consistency and expansion score $S_{EP}(T_1, T_2) = S_{EP}(E_1, E_2 \setminus E_1)$ and schema consistency score $S_{SS}(T_1, T_2)$:

$$EC(T_1, T_2) = S_{EP}(T_1, T_2) * S_{SS}(T_1, T_2). \quad (9)$$

While more complex combinations of these two factors are possible, the above equation captures the intuition that both scores need to be non-zero if T_2 is entity complement to T_1.

4. SCHEMA COMPLEMENT

In this section, we describe how we identify schema complement tables. Given a table T_1, we are interested in finding tables T_2 that provide the best set of additional columns. Intuitively, we want to add as many properties as possible to the entities in T_1 while preserving the "consistency" of its schema. We consider the following factors:

Coverage of entity set: T_2 should consist of most of the entities in T_1, if not all of them. This is required by Point 3 in Definition 2. We compute the coverage of T_2's entity set with respect to T_1 as follows. We first construct the unique set of entity identifiers E_1 and E_2 using Freebase, and then compute the fraction of T_1's entities covered by T_2: $S_{ECover}(T_1, T_2) = \frac{|E_1 \cap E_2|}{|E_1|}$.

Benefits of additional attributes: T_2 should contain additional attributes that are not described by T_1's schema. This is required by Point 2 in Definition 2. To quantify the benefits of adding the set of additional attributes, we need to combine the *consistency* and *quantity* of T_2's additional attributes.

The additional attributes in T_2 are determined by performing schema matching (described in Section 3.2) between T_1 and T_2's schema. An attribute of T_2 is an additional attribute if it is not mapped to any attribute in T_1 with a score above a threshold. (Obviously, T_2's subject column attribute is never an additional attribute, since a pre-requisite to schema complement is that T_2's subject column maps to T_1's subject column.) We use $S(T_i)$ to denote the set of attributes in T_i and $S(T_2) \setminus S(T_1)$ to represent the additional attributes in T_2's schema.

A baseline measure for the benefit of T_2 simply counts the number of additional attributes:

$$S_{SB}^{count}(S(T_1), S(T_2)) = |S(T_2) \setminus S(T_1)|. \quad (10)$$

However, the S_{SB}^{count} measure does not capture the relative importance of additional attributes. We can obtain a more meaningful measure by leveraging the AcsDB [10], a data structure that summarizes the frequencies of all possible schemas in the Web table corpus. Given a set of attributes S, the AcsDB provides the frequency, $freq(S)$, of S in the table corpus.

Given the AcsDB, we can measure the contribution of the schema of T_2's w.r.t. T_1 using the *schema auto-complete score* [10]. Intuitively, the measure below determines the likelihood of seeing the new attributes in T_2's schema given T_1's attributes; the higher the likelihood of seeing these new attributes, the higher is the score.

$$\begin{aligned} S_{SB}^{set}(T_1, T_2) =& P(S(T_2) \setminus S(T_1) | S(T_1)) \\ =& \frac{freq(S(T_2) \cup S(T_1))}{freq(S(T_1))}. \end{aligned} \quad (11)$$

This basic measure has two drawbacks: (1) the score is monotonically decreasing, so adding more attributes to T_2 only hurts the benefit measure; (2) the $freq(S)$ in AcsDB is not as meaningful for large schemas like $S(T_2) \cup S(T_1)$ because they appear very few times in the web table corpus.

The following measure partially overcomes these drawbacks by considering the maximal benefit that a subset of attributes of T_2 can provide:

$$\begin{aligned} S_{SB}^{setmax}(T_1, T_2) =& \max_{S \subseteq (S(T_2) \setminus S(T_1)) \wedge S \neq \emptyset} P(S | S(T_1)) \\ =& \max_{a \in S(T_2) \setminus S(T_1)} P(a | S(T_1)) \\ =& \max_{a \in S(T_2) \setminus S(T_1)} \frac{freq(\{a\} \cup S(T_1))}{freq(S(T_1))}. \end{aligned} \quad (12)$$

Although $\{a\} \cup S(T_1)$ is more likely to appear in AcsDB than $S(T_2) \cup S(T_1)$ as its size is smaller, the number of appearances are still too small to derive statistical significant

results. A more effective measure is to derive the benefit measure by considering co-occurrence of *pairs* of attributes, rather than the entire schemas. Specifically, we begin by determining the consistency of a new attribute a_2 to an existing attribute a_1, denotedy by $cs(a_1, a_2)$, using AcsDB schema frequency statistics as follows:

$$cs(a_1, a_2) = P(a_2|a_1) = freq(\{a_1, a_2\})/freq(\{a_1\}). \quad (13)$$

The consistency of an additional attribute a_2 ($a_2 \notin S(T_1)$) to the original schema $S(T_1)$ is then computed as:

$$cs(S(T_1), a_2) = \frac{1}{|S(T_1)|} \sum_{a_1 \in S(T_1)} cs(a_1, a_2). \quad (14)$$

We can then compute the benefit of $S(T_2)$ to $S(T_1)$, denoted as $S_{SB}(T_1, T_2)$, by aggregating the consistencies of each $a_2 \in S(T_2) \setminus S(T_1))$ to $S(T_1)$. We consider three kinds of aggregation: sum (giving importance to the amount of extension), average (normalizing the total extension by the number of new attributes), and max (considering the most consistent attribute as representative):

$$S^{sum}_{SB}(T_1, T_2) = \sum_{a \in S(T_2) \setminus S(T_1)} cs(S(T_1), a), \quad (15)$$

$$S^{avg}_{SB}(T_1, T_2) = \frac{1}{|S(T_2) \setminus S(T_1)|} \sum_{a \in S(T_2) \setminus S(T_1)} cs(S(T_1), a), \quad (16)$$

$$S^{max}_{SB}(T_1, T_2) = \max_{a \in S(T_2) \setminus S(T_1)} cs(S(T_1), a). \quad (17)$$

Putting it all together

We combine the entity coverage score S_{ECover} and the attribute benefit measure S_{SB} to obtain the overall schema complement score. Of course, more complex combinations can be explored.

$$SC(T_1, T_2) = S_{ECover}(T_1, T_2) \times S_{SB}(T_1, T_2). \quad (18)$$

5. EXPERIMENTAL RESULTS

We present an initial set of experiments demonstrating the effectiveness of our techniques for finding related tables (Section 5.1). We then describe an experiment that suggests that finding related tables has an added potential of improving search results for tables (Section 5.2).

5.1 Evaluating related tables

Label source	Top-1	Top-3	Top-5
WebIsA	1.9	1.8	1.6
Freebase	2.0	2.1	1.9
WebTable	1.5	1.5	1.6
WebIsA + FB	1.9	2.2	1.9
All combined	1.9	2.0	1.9

Table 1: Comparing different sources of labels for EC. We use $S^{AvgPair}_{EP}$ to compute the top-k results.

We evaluate the effectiveness of different scoring functions for entity complement (EC) and schema complement (SC), based on user judgements. Given a query table T_1, a relatedness R (R is either EC or SC), and a candidate related table T_2, we ask each user to provide a score from 0 to 5 indicating how closely T_2 is related to T_1 with respect to R (0 being not related and 5 being most related).

S_{EP}	Top-1	Top-3	Top-5
AvgPair	2.0	2.1	1.9
SumPair	1.6	1.8	2.0

Table 2: The impact of entity expansion for EC. Average ratings of top-k results with and without encoding the amount of expansion (SumPair vs. AvgPair) using Freebase labels.

S_{EP}	Label source	Top-1	Top-3	Top-5
AvgPair	WebIsA	1.9	1.8	1.6
Set	WebIsA	1.8	1.8	1.7
AvgPair	Freebase	2.0	2.1	1.9
Set	Freebase	2.1	2.1	2.0
AvgPair	WebTable	1.5	1.5	1.6
Set	WebTable	2.8	2.3	2.1

Table 3: The impact of entity-set based relatedness measures for EC. Average ratings of top-k results for entity pair relatedness aggregation measures vs. entity-set based related measures. For entity-set based measures, the parameters (defined in Section 3) are set to $m_1 = n_1 = m_2 = n_2 = 2.0$.

Experimental setup: We evaluated the relatedness algorithms on 18 queries. For each relatedness R and query table T_1, we obtain user ratings as follows: (1) for each R we consider multiple scoring functions $R_1, R_2, ..., R_n$ (from Sections 3 and 4); (2) for each possible scoring function R_i, we generate top-5 related tables for the query table T_1 based on R_i; (3) we randomly sort the set of all top-5 related tables from all scoring functions and remove the duplicates; (4) we show the combined set of related tables to the user, and they provide ratings for each table. Our results are based on aggregating the feedback of 8 users.

Metric: We measure the effectiveness of each scoring function as follows. For each query table, we generate top-k (k = 1 to 5) related tables based on it. We then average ratings of top-k (k = 1, 3, 5) related tables for any query across all user judgements obtained as described above. A higher average means a better scoring function.

Entity Complement Results: Tables 1, 2, and 3 summarize the ratings for entity complement scores obtained using all approaches described in Section 3. We use the same algorithm for schema similarity (Section 3.2) across all experiments, hence we are actually comparing the effectiveness of the different algorithms for entity relatedness score S_{EP}

Algorithm	Top-1	Top-3	Top-5
sum	3.5	3.4	3.4
max	3.1	3.1	3.2
avg	2.7	2.9	3.0
setmax	1.8	2.1	2.0
count	3.1	3.1	3.0

Table 4: Average rating for top results, for different S_{SB} definitions in SC.

and the effectiveness of different sources of labels. We make the following observations.

Best Approach: Our method for computing relatedness based on comparing the entire sets S_{EP}^{Set} offers best results when used with labels computed from WebTable, and significantly better than baselines that consider relatedness of all pairs in the two sets S_{EP}^{*Pair}. Using WebTable signals, S_{EP}^{Set} achieves around 87% rating improvement to S_{EP}^{*Pair}, its entity-pairs counterpart and around 40% improvement to the next best algorithm (entity pair relatedness aggregation with Freebase labels).

Table 1 – Label source comparison for entity pair based algorithms: When using entity pair based algorithms (e.g., $S_{EP}^{AvgPair}$), Freebase labels are most effective if we consider only one source of labels. Interestingly, adding WebIsA and WebTable labels does not have an apparent impact.

Table 2 – Impact of expansion quantity: Entity consistency is more important than entity-set expansion. The measure $S_{EP}^{AvgPair}$, which rewards consistency over entity-set expansion is significantly better (up to 25% improvement) compared with $S_{EP}^{SumPair}$, which rewards expansion of the entity set over consistency.

Table 3 – Impact of set-based relatedness measure: As described above, the best result combines set-based relatedness with labels from WebTable. When we consider labels from WebIsA or Freebase, the set-based measure performs only slightly better than comparing entities pairwise. This can be explained by the fact that WebIsA and Freebase are good at capturing general concepts, which are less sensitive to entity set based relatedness. The WebTable corpus captures much richer set of concepts, but also contains more noisy signals. The set-based relatedness helps distill the useful concepts (by encouraging labels that are common to most entities in the set) while disregarding the noise (by discouraging labels that occur only a few times in the set) from WebTable corpus.

Schema Complement Results: Table 4 compares effectiveness of different schema complement scoring functions, varying the scoring functions for measuring the benefits of additional attributes S_{SB}. S_{SB}^{sum} achieves the best results, while S_{SB}^{max} and S_{SB}^{count} also perform reasonably well. Note that S_{SB}^{max} focuses on consistency of expansion, S_{SB}^{count} focuses on the amount of expansion, while S_{SB}^{sum} focuses on both. The baseline S_{SB}^{set} obtains a score close to 0 (not shown in the table), since co-occurrence statistics for large schema are not meaningful. S_{SB}^{setmax} is slightly better, but still significantly worse than the best approaches. Finally, S_{SB}^{avg} is suboptimal since the amount of expansion is completely ignored. In summary, *sum* aggregation is the most effective, and it indicates that when considering schema complement, users indeed focus on the number of additional attributes, in addition to the consistency of additional attributes.

5.2 Augmenting table search

The most natural use-case of related table discovery is to present users with related tables when they are exploring a particular table. This section demonstrates an unexpected use of discovering related tables: improving table search [10]. Specifically, we show that tables that are related to tables that are highly ranked w.r.t. a query are often judged to be equally relevant to the query, even if they appear much further down in the ranking.

Query	# Related	# Better Related
1 country gdp	26	7
2 country population	21	4
3 dog species	8	6
4 fish species	6	4
5 movie director	10	5
6 national parks	6	4
7 nobel prize winners	1	1
8 school ranking	6	6

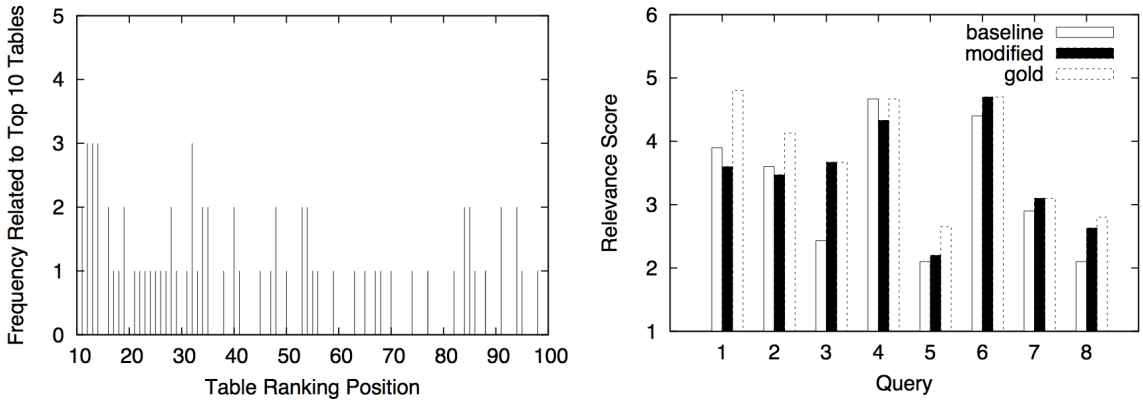
Table 5: Queries and Statistics

To illustrate this point, we experimented with the following very simple re-ranking of search results from [10]. After the first result T_1, we add all tables among the top-100 tables $T_1^1, T_1^2, \ldots$ related to it (in order of relatedness), then add the second result T_2 followed by all tables (not already listed above) related to it $T_2^1, \ldots$, and so on. In this fashion, we created a re-ranked list of tables consisting of the original top-10 tables and all its related tables from the original top-100 tables. We considered eight keyword queries (listed in the first column of Table 5), and asked four users to rate all the resulting tables (by randomly sorting the results) giving them a score between 0 and 5.

The second column of Table 5 shows the number of related tables added by the approach above. The third column shows the number of related tables that were not in the original top-10 but had an average user score that was in the top-10. For example, a value of 4 in the third column of Table 5 indicates that 4 related tables (not in the original top-10 search result list) obtained an average user rating among the top-10 when all tables are ranked based on the average user rating. We notice that in all the queries, related tables constitute a significant (if not majority) portion of the ideal top-10 results, except the nobel prize winners query, which only added one related table. This indicates that related tables can be used as an important feature in tuning keyword search results.

Figure 5(a) shows that the added tables are distributed widely within the top 100 rankings for these queries, indicating that tables throughout the top 100 are "pulled" forward by our modified algorithm. Finally, Figure 5(b) shows that in most cases, our modified ranking algorithm gives a higher average relevance score compared to the original search result (marked baseline), and very close to the "gold standard" that takes the top-10 simply based on user ratings. When we average relevance across all eight queries, the modified algorithm achieves a relevance score of 3.45, beating the baseline of 3.26, with the gold standard being 3.81.

Our results can be explained by the observation that related tables are being pulled up based on semantic signals inferred from the "hidden link structure" across all tables, where links represent "relatedness" edges; these semantic signals are sometimes orthogonal to the relatively syntactic ones used for table ranking, such as in [10] and can be used to complement other techniques for recovering semantics of tables [24, 30]. Designing the optimal method to blend in related tables into search results requires additional study and will also be greatly influenced by user interface hints.



(a) Distribution of related tables

(b) Comparison of three algorithms

Figure 5: Using related table data to improve table search ranking.

6. SCALING UP

Computing the exact relatedness score for every pair of tables can be very expensive on large table corpora. In this section we discuss how to scale up the computation of table relatedness by considering filters that reduce the number of computations we need to perform and that enable us to perform each comparison more efficiently. We describe the general approach in Section 6.1. In Section 6.2 we describe a model for choosing among different candidate filtering criteria. In Section 6.3 we describe a set of candidate filtering criteria and use the model to compute their expected running time. We run the chosen filtering technique on a corpus of over 1 million tables extracted from Wikipedia.

6.1 General Approach

We start by presenting a general filtering-based approach we employ for scaling related tables computation. For each type of relatedness measure (i.e., entity complement, EC, and schema complement, SC) R, we devise filtering criterion $F_1, F_2, \ldots$: Each F_i is equivalent to a *hash function* and the filtering condition imposes $F_i(T_1) = F_i(T_2)$ for a pair of tables T_1 and T_2. As we shall see, each $F_i(T)$ could generate multiple values, and we would like $F_i(T_1) \cap F_i(T_2) \neq \emptyset$; hence we use $F_i(T_1, T_2)$ as a shorthand for this condition. Intuitively, we would like a high relatedness score for the pair T_1, T_2 to satisfy $F_i(T_1, T_2)$ (and a low score to falsify the filter). We shall discuss the desiderata for good filtering criteria shortly.

Filtering conditions are used in two ways in our algorithm:

Fewer Comparisons: Fewer comparisons are performed by using the filtering conditions as *hash functions* to bucketize the set of all tables, and only perform relatedness computations for pairs of tables that appear together in some bucket. (Note that this is similar to *blocking* or *canopy formation* in de-duplication[21].) Since each filtering criterion may be based on multiple hash values for each table, every table may go in more than one bucket. Subsequently, we perform the set of pairwise comparisons in all buckets in parallel, as with recent work on parallelizing similarity joins [6, 31]. The pairwise comparisons for each bucket are performed on different machines, with the set of all pairs on large buckets further subdivided into multiple machines. We used map-reduce in our implementation of parallel computation of relatedness measures.

Faster Comparisons: To make the computation of relatedness score on each pair of tables faster, we apply a sequence of filters. Only when a specific filtering condition is satisfied, we apply the next filter. Finally, only when all filters in the sequence are satisfied, we perform the computation of the entire relatedness score. This process is beneficial when the filters have a low selectivity and are efficient to compute (and in particular, much more efficient than the relatedness score computation). Such a technique of applying a sequence of filters has been considered in other contexts in the past [7, 13, 23, 29].

While the set of filters that can be applied for the two improvements above may vary in general, all our filtering conditions can be framed as hashing functions (based on the non-empty intersection condition: $F_i(T_1) \cap F_i(T_2) \neq \emptyset$), and hence, we explore the same set of filters for both the optimizations.

6.2 Filtering Criteria Selection

Next we describe how to select the best filtering criterion for a relatedness measure R. Let n denote the total number of tables in the corpus and t_R denote average time to compute the comparison R for a pair of tables. The goodness of a filtering criterion F depends on the three factors:

- **Computation time:** We would like filtering criteria to be very efficiently computable. For the purpose of fewer comparisons, we need to be able to map each table to a bucket efficiently, and for faster comparisons, the filtering predicate should take much less time to apply than computation of the entire relatedness score.

  Let t_F denote the average time to compute the filtering condition F on a table.

- **Selectivity:** We would like the filtering criterion to have low selectivity, i.e., few pairs should satisfy each filtering criterion. Ideally, satisfying the filtering condition should be correlated with high relatedness score.

  Selectivity, denoted by m_p, is the total number of candidate related pairs that pass the filter criterion. We denote by m_{up} the number of *distinct* pairs that satisfy the filtering criterion. Note that m_{up} is always less

or equal to n^2, but m_p can be larger than n^2 because every filter can generate multiple values and therefore a table can appear in multiple buckets.

- **Loss rate:** We would like very few table pairs with high relatedness score to falsify filtering criteria. That is, we would like $F_i(T_1, T_2) = \textit{false} \Rightarrow \textit{relatedness}(T_1, T_2) < \tau$, for some threshold τ. The loss rate of a filtering criteria is defined as the number of table pairs that don't satisfy the above, and we would like to design filtering criteria with low loss rates.

The unit computation times and selectivity of filter criterion decides the total running time for related table discovery. When using bucketization-based optimization, the total running time for related table discovery can be estimated as:

$$t_{bucket} = nt_F + m_p t_R \approx m_p t_R.$$

If a de-duplication step is performed before pairwise relatedness comparison, the estimate is:

$$t_{bucket-dedup} = nt_F + t_{dedup} + m_{up} t_R \approx t_{dedup} + m_{up} t_R,$$

where t_{dedup} is the time to perform de-duplication for all candidate pairs in different buckets, though the exact running time for de-duplication (run as a separate round in map-reduce) can be very data dependent. And if we directly apply the filter to all pairs of candidate tables, the total estimated running time is:

$$t_{allpair} = n^2 t_F + m_{up} t_R.$$

Therefore, for a table corpus and a relatedness R, the best filtering criterion can be decided as follows:

1. Decide a set of candidate filtering criteria for R (denoted as $F_1, F_2, ..., F_{fn}$).
2. Compute the loss rate for each candidate filtering criterion F_i and ignore any filtering criterion with loss rate $> \tau$.
3. For each remaining candidate criterion F_i, estimate the total running time for related table discover as

   $$t_{est}(F_i) = \min(t_{bucket}, t_{bucket-dedup}, t_{allpair})$$

   by computing $m_p t_R$, $m_{up} t_R$ and $n^2 t_F$ and an estimation of t_{dedup} if needed. Pick the filter criterion F_i (and its corresponding optimization algorithm) with the lowest estimated running time.

6.3 Evaluation on Wikipedia tables

We use the corpus of tables extracted from Wikipedia (over 1 million tables) to demonstrate how to select filtering criteria for entity complement EC and schema complement SC and compute the related tables using the selected filtering conditions. The entity complement algorithm we use here is $S_{EP}^{AvgPair}$ score with WebIsA and Freebase labels, and the schema complement algorithm uses the S_{SB}^{max} score. We then summarize and present some basic statistics of related Wikipedia tables.

We describe 9 candidate filtering criteria. The first set of criteria apply to both entity and schema complement: **Subject-Name (SN)**–two tables should have the same subject column name; **Subject-Name-Label (SNL)**–two tables must share the subject column name or at least one subject column label; **Subject-Name-PrunedLabel (SNLPr)**–two tables must share the subject column name or at least one subject column label not on a predefined prune list. Some labels are very general therefore meaningless (e.g., *thing, factor*). We manually identified 20 such labels and put them in a prune list.

	EC loss rate	SC loss rate
SN	63.6%	55.7%
SNL	0.0%	0.0%
SNLPr	0.0%	0.0%
NPair	66.3%	N/A
NLPair	25.0%	N/A
NLPrPair	25.5%	N/A
Entity1	N/A	0.0%
Entity2	N/A	3.2%
Entity3	N/A	5.1%

Table 6: Estimated loss rates of different filtering criteria.

The second set of criteria apply only to entity complement. **Name-Pair (NPair)**–two tables should share both of: (1) a subject column name, and (2) at least one non-subject column name; **Name-Label-Pair (NLPair)**–two tables should share both of: (1) a subject column name or label, and (2) at least one non-subject column name or label; **Name-PrunedLabel-Pair (NLPrPair)**–two tables should share both of: (1) a subject column name or label not on the prune list, and (2) at least one non-subject column name or label not on the prune list.

The third set of filters apply only to schema complement. Entity-n requires that the two tables share at least n entities. Our experiments consider $n = 1, 2, 3$.

For each filtering criterion, we estimate its loss rate for entity complement and schema complement (if applicable) based on the user ratings obtained in Section 5.1. Any pair of tables with average rating larger than 0 are considered related. The results are show in Table 6. We noticed that *SN* has high estimated loss rate for both entity and schema complement; *NPair*, *NLPair*, *NLPrPair* has high estimated loss rate for entity complement. It is interesting to observe that pruning the most general and meaningless labels almost have no effect on the loss rate (e.g., SNL and $SNLPr$ have the same loss rate of 0).

Next we estimate the total related table discovery time for both entity and schema complement under different filtering conditions. (For illustration purposes we show the time estimation even for the high loss rate filtering criteria.) Following the estimation method discussed in Section 6.2, we first compute the following basic numbers: total number of tables $n = 1015496$, unit computation time for entity complement $t_{EC} = 1.6ms$, unit computation time for schema complement $t_{SC} = 0.4ms$, unit computation time for all filtering conditions are less than $0.01ms$: $\{t_{SN} = 0.0005$ms, $t_{SNL} = 0.0017$ms, $t_{SNLPr} = 0.0019$ms, $t_{NPair} = 0.0009$ms, $t_{NLPair} = 0.0065$ms, $t_{NLPrPair} = 0.0072$ms, $t_{Entity1} = 0.0014$ms, $t_{Entity2} = 0.0014$ms, $t_{Entity3} = 0.0008$ms$\}$, and the number of total pair comparisons m_p and unique pair comparisons m_{up} for different filtering criteria are shown in Figure 6.

We aggregate these numbers to compute $m_{up}t_{EC}$, $m_{up}t_{SC}$, $m_p t_{EC}$, $m_p t_{SC}$, $n^2 t_F$ (Table 7). In almost all cases, all pair filtering comparison ($n^2 t_F$) is rather cheap compared to relatedness computing. Therefore we can safely

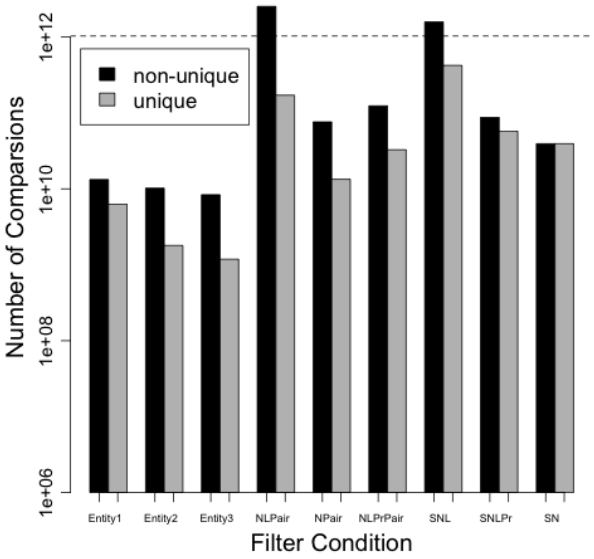


Figure 6: Table pair comparison counts (m_{up} and m_p) for different filtering conditions. The dashed horizontal line are total number (n) of table pairs in the corpus.

	$m_{up}t_{EC}$	m_pt_{EC}	$m_{up}t_{SC}$	m_pt_{SC}	n^2t_F
SN	17(X)	17(X)	4(X)	4(X)	0.1
SNL	186	697	48	178	0.5
SNLPr	25	39	6	10	0.4
NPair	6(X)	33(X)	/	/	0.3
NLPair	76(X)	1116(X)	/	/	1.8
NLPrPair	14(X)	55(X)	/	/	2.1
Entity1	/	/	0.7	1.5	0.4
Entity2	/	/	0.2	1.2	0.3
Entity3	/	/	0.2	0.9	0.3

Table 7: Estimated total running time (thousand hours) breakdown. Annotated with (X) if it comes with high loss rate.

use $t_{allpair} = n^2t_F + m_{up}t_R$ as the estimated running time and no need to worry about de-duplication for bucketization. For entity complement, $SNLPr$ has the lowest $m_{up}t_R$ among all those low loss rate filter criteria. The estimated total running time is 4000 (n^2t_F) + 25000 ($m_{up}t_{EC}$) = 29000 hours. Schema complement computations are much cheaper. Using $Entity1$ as the filtering criterion, the estimated running time is 400 (n^2t_F) + 700 ($m_{up}t_{SC}$) = 1100 hours; it is even cheaper when using $Entity3$, although it comes with the price of a bit higher loss rate.

We ran our system to compute related table pairs on the entire Wikipedia corpus of tables consisting of around 1 million tables. We ran our system using 10000 machines, and it took about 3 hours to finish the computation. This is consistent with our estimation. Figure 7 shows some basic statistics on this Wikipedia dataset. We can see that the number of related tables roughly follows a power-low distribution. We also notice that there is a large variance in the number of related tables, with some tables having many related tables (for this experiment we applied a very small relatedness threshold to obtain a maximum number of related tables).

7. RELATED WORK

We are not aware of any prior work that addresses the problem of identifying related tables on the Web. However, the algorithms we describe touch on a few bodies of related work which we discuss below.

Extracting Web tables

There have been several pieces of work that extract tables from the Web: [18] extract tables from arbitrary Web pages relying on positional information of visualized DOM element nodes in a browser. Cafarella et al. [11] developed the WebTables system for web-scale table extraction, implementing a mix of hand-written detectors and statistical classifiers that identified 154 million high-quality relational-style tables from a raw collection of 14.1 billion tables on the Web. Elmeleegy et al. [16] split lists on Web pages into multi-column tables in a domain-independent and unsupervised manner. The Octopus System [9] includes a context operator that tries to identify additional information about the table on a Web page. For example, the operator would identify the year of the page listing the program committee members of VLDB, even if the year is not explicit on the page. Using this recovered information, it is easier to union tables found on different Web pages. This technique is orthogonal to the algorithms we described and can be incorporated as another method for detecting related tables.

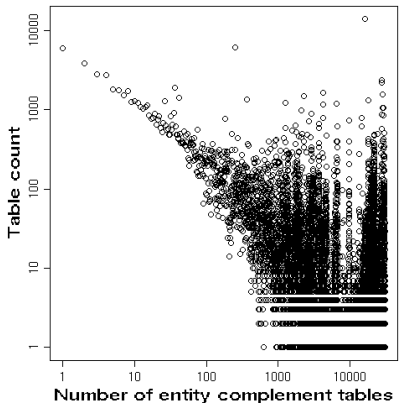
Our table corpus is obtained from an enhanced WebTables system that expands upon [11] by incorporating the all state-of-art table extraction described above.

List expansion

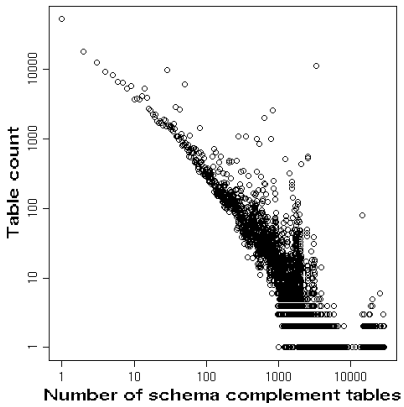
The problem of entity complement is related to the task of *list expansion*, which is to generate lists of named entities starting from a small set of seed entities. Several approaches to this general problem have been proposed, including supervised entity extraction targeted at a limited set of classes [8, 25], and systems such as KnowItAll [17] and [14], generating lists of queries for each target predicate and applying a set of extraction rules over the returned documents to get named entities. Other approaches include applying isA Hearst patterns to generate instances and classes, automatic set expansion using a similarity matrix between words [27], systems such as SEAL that identify lists of items on webpages [32, 33]. All these works focus on adding more *n*amed entities to a small set of seed entities. Entity complement computes the value of the additional set of rows that any table adds to the input table, which is obtained by a combination of the relatedness of the *set* of additional entities, and their attribute values.

Keyword search on tables

There have been numerous recent papers describing ranking algorithms for keyword search queries over table corpora, including treating tables as pseudo-documents that include table's surrounding text and page titles [10], leveraging a database class labels and relationships extracted from the Web [30], and using the YAGO ontology to annotate tables with column and relationship labels [24]. In other work, [20] considered how to answer fact queries with lists on the Web and there is a large body of work for ranking tuples within a single database in response to keyword queries [22]. All the above works take keyword queries as input, and the goal is to find or construct tables most relevant to the keywords.



(a) Entity complement table count distribution.



(b) Schema complement table count distribution.

Figure 7: Related table count distribution in Wikipedia corpus.

In contrast, our input is a table in itself, and our goal is to find other tables that may be *combined* with the input table such as by a join or union.

8. CONCLUSIONS AND FUTURE WORK

We introduced the problem of finding related tables from a large heterogenous corpus that are related to an input table. We presented a framework that captures a multitude of relatedness types, and described algorithms for ranking tables based on *entity complement* and *schema complement*. We described user studies that evaluated the quality of our related tables detection algorithms, and showed how related tables discovery may enhance table search results. Finally, we showed how to scale the computation of related tables. In future work, we plan to devise algorithms for other relatedness types such as *temporal snapshots*, and explore relatedness of tables in the context of a given query on the table corpus.

9. REFERENCES

[1] http://secondstring.sourceforge.net/.
[2] http://www.factual.com/.
[3] http://www.freebase.com/.
[4] http://www.socrata.com/.
[5] http://www.tableausoftware.com/public.
[6] F. Afrati, A. D. Sarma, D. Menestrina, A. Parameswaran, and J. D. Ullman. Fuzzy joins using mapreduce. In *ICDE*, 2012.
[7] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *SIGMOD*, 2004.
[8] R. Bunescu and R. J. Mooney. Collective information extraction with relational markov networks. In *ACL*, 2004.
[9] M. Cafarella, A. Halevy, and N. Khoussainova. Data Integration for the Relational Web. *PVLDB*, 2(1):1090–1101, 2009.
[10] M. Cafarella, A. Halevy, D. Wang, E. Wu, and Y. Zhang. WebTables: Exploring the Power of Tables on the Web. *PVLDB*, 1(1):538–549, 2008.
[11] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Uncovering the Relational Web. In *WebDB*, 2008.
[12] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003.
[13] A. Condon, A. Deshpande, L. Hellerstein, and N. Wu. Flow algorithms for two pipelined filter ordering problems. In *PODS*, 2006.
[14] D. Davidov. Fully unsupervised discovery of concept-specific relationships by web mining. In *ACL*, 2007.
[15] Z. (Eds.) Bellahsene, A. Bonifati, and E. Rahm. *Schema Matching and Mapping*. Springer, 2011.
[16] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting Relational Tables from Lists on the Web. *PVLDB*, 2:1078–1089, 2009.
[17] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. nsupervised named-entity extraction from the Web: An experimental study. *AIJ*, 2005.
[18] W. Gatterbauer and P. Bohunsky. Table extraction using spatial reasoning on the CSS2 visual box model. In *AAAI*, 2006.
[19] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *SIGMOD*, 2010.
[20] R. Gupta and S. Sarawagi. Answering Table Augmentation Queries from Unstructured Lists on the Web. *PVLDB*, 2(1):289–300, 2009.
[21] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, 1995.
[22] P. Ipeirotis and A. Marian, editors. *DBRank*, 2010.
[23] M. Kodialam. The throughput of sequential testing. In *In Integer Programming and Combinatorial Optimization*, 2001.
[24] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. In *VLDB*, pages 1338–1347, 2010.
[25] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *CONLL*, 2003.
[26] M. Paşca and B. Van Durme. Weakly-Supervised Acquisition of Open-Domain Classes and Class Attributes from Web Documents and Query Logs. In *ACL*, 2008.
[27] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, and V. Vyas. Web-scale distributional similarity and entity set expansion. In *EMNLP*, 2009.
[28] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4), 2001.
[29] U. Srivastava, K. Munagal, J. Widom, and R. Motwani. Query optimization over web services. In *VLDB*, 2006.
[30] P. Venetis, A. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. In *PVLDB*, 2011.
[31] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD*, 2010.
[32] R. Wang and W. Cohen. Language-Independent Set Expansion of Named Entities Using the Web. In *ICDM*, 2007.
[33] R. Wang and W. Cohen. Iterative Set Expansion of Named Entities Using the Web. In *ICDM*, pages 1091–1096, 2008.