Identifying Relatedness in Web Tables to Perform Cell Search

Janani Krishna
UMass Amherst
jkrishna@cs.umass.edu

Ananya Suraj
UMass Amherst
asuraj@cs.umass.edu

Bhuvana Sai Surapaneni
UMass Amherst
bsurapaneni.@cs.umass.edu

ABSTRACT

Today, the internet consists of a large amount of unstructured data, but it also contains over 100 million tables that are structured in HTML. Each such table contains relational data and a small schema of labeled columns. Different from unstructured texts, each table comes with its own schema; therefore it is much easier to interpret the entities and relations contained in a table. Informative web tables range across a large variety of topics, which can naturally serve as a significant resource to satisfy user information needs. A clean collection of relational-style tables could be useful for improving web search, schema design, and many other applications. Driven by such observations, in this project, we aim to investigate an important yet largely under-addressed problem: Given millions of tables, how to precisely retrieve table cells to answer a user's question. In this paper, we first find related tables i.e. tables that are candidates for joins or unions, which enable users to reuse available data and find new patterns effectively. We then propose an efficient technique to perform table cell search by harnessing this information about related tables. An answer to a query can be found without the need to explore all possible tables in a large corpus of data, thereby reducing the amount of computation required as well as improving the speed in which the result is obtained. We have also designed a framework to iteratively improve search performance when similar queries are asked repeatedly.

1. INTRODUCTION

Knowledge Discovery and Data Mining (KDD) is an interdisciplinary area focusing on methods for extracting useful knowledge from data. The ongoing rapid growth of online data due to the Internet and the widespread use of databases have created an immense demand for KDD. The challenge of extracting knowledge from data draws upon research in statistics, databases, pattern recognition, machine learning, data visualization, optimization, and high-performance computing, to deliver advanced business intelligence and web discovery solutions. Data discovery is designed to answer immediate, spur-of-the-moment questions. An ideal data discovery solution allows information from many sources to be accessed whenever needed.

Today, tables are pervasive over the internet and serve as important information resources. Further, enterprises store their business critical data about products, employees and other information as tables in spreadsheets or relational databases. Efficiently and accurately finding relevant information in these tables are often very important from a management and analysis perspective. If there is just one table with structured rows and columns, it is easy for a human analyst to find the required information. But when there are millions of such tables, manual checking becomes infeasible. Without knowing the semantics of the tables, it is very difficult to leverage their content, either in isolation or in combination with others.

We propose a novel approach to perform table cell search - find the related tables and harness this information to speedup and improve the accuracy of search queries. In this paper, we describe techniques to capture different types of relatedness and schema matching, including tables that are candidates for joins and tables that are candidates for union. Before, we do this, we are faced with two problems. First, some tables may have partial schema and might be extremely heterogeneous. In most cases, important aspects of the table that can help us theorize about relatedness are embedded in metadata that surrounds the tables or are embedded as textual descriptions of the tables. Secondly, there are different ways in which two tables can be related to each other and also the degree to which they are related might differ. We define and find two kinds of relatedness - Entity Complement and Schema Complement as our first step.

Further, we define and use the concept of Relational Chains in order to perform table cell search which is our ultimate goal. We have explained our methods to prune these relational chains in order to reach the required answer cell. This paper also proposes a performance boosting technique to improve table search. It is often the case that the same kind of information is queried frequently from a database. For example, an enterprise might query information about their current products and employees on a regular basis. However, it is rare that any search may be conducted for information that is more than a few years old and so, no longer relevant. We have designed a novel framework that takes advantage of this fact and improves response time by giving more importance to data that is frequently accessed. We evaluate the framework discussed in this paper by testing it with some publicly available datasets and reporting our findings. We also compare our techniques with a naive approach to cell search and show how the response time improves over time through our framework. Additionally, we conclude by discussing the advantages, drawbacks and extensions for our system along with relevant related work.

2. PROBLEM DESCRIPTION

Prior work addresses the problem of identifying related tables in a database and on the web [1][4]. However, entire tables (that are considered relevant) are returned to the user for information summarization or reusing data to find new patterns. Other work focuses on using the structure of web documents to answer questions based on creating indexes on the whole document rather than using individual tables that make up the document [10]. Some work has been done on cell searching using the relations between tables and returning the exact answer requested by the user [7]. Annotation of table cells is also another method that has been employed for cell searching [11]. These existing techniques are sometimes too expensive to perform as data grows in size and results are expected almost instantaneously. Even though web tables have schema, it is not easy to parse millions of tables at once while looking for a particular answer.

We consider a large corpus of HTML web tables T, with only partial metadata such as column headers and relational information represented by the table which can only be inferred by examining cell values and surrounding table descriptions. Given a table T1, we must find tables in T that are semantically related to T1. If two tables T1 and T2 could have belonged to a larger table T, then we can surmise that they are in fact, related to each other.

We identify two forms of relatedness - Entity Complement and Schema Complement, the definitions of which will be covered in the next section. However, on a high level, we can say that two tables T1 and T2 are related to each other if on applying two queries Q1 and Q2 over a larger table T_0, we get T1 and T2 as a result. Though the attributes may be named differently, the semantic meaning of each column (by analyzing the values) makes this possible. The only constraint is that the queries Q1 and Q2 must have similar structure i.e. they must both be projections on T_0 or selections on T_0.

Once we have identified related tables in our corpus, we next tackle the problem of finding correct answers in the corpus of tables to user queries. The information on related tables not only helps us reduce computation and speed things up, but it also has other advantages as well. The targeted questions in this task are those that contain at least one entity, named topic entity and an attribute of that topic entity. The topic entity and the attribute that is required may not belong to the same table (this is another advantage of having found related tables beforehand).

Each cell in a row of a table represents a relation between the various attributes of that table as well as any table that can be joined with it. This relation can be depicted as a chain where the cell value is a node and the edge is labelled with the corresponding column name. We call this a relational chain. Figure 1 shows such a chain for the topic entity ”USA” and attribute ”president”.

We form relational chains for the topic entity by considering related tables and the context of each value in these tables

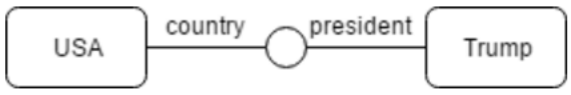

Figure 1: Relational chain with topic entity ”USA” and attribute ”president”

i.e. USA could also have the value United States.

Chains of entity-attribute pairs are formed starting with the entity that was queried along with all related attributes in the entity sets that were grouped earlier. The resulting chains are then narrowed down to only those pairs that are closely related to the question pattern by using semantic analysis and contextual entity mapping techniques. This is a two-stage pruning process which then returns the top-K relational chains to the user.

For example, user query is, ”Which domain does Microsoft fall under?” Here topic entity is ”Microsoft” and attribute is ”domain”. Chains are formed with all attributes related to Microsoft, say HQ location, CEO, employees and domain. The chains are pruned and irrelevant results like CEO are removed and the results are ranked using a ranking algorithm (this would return ”domain” as a top ranked outcome). The cell value corresponding to ”Microsoft” row and ”domain” column, say ”software” would be returned as the answer.

It is also beneficial to iteratively improve the search performance of the system by use of weight functions that are updated every time an answer is returned to the user. This mechanism has been explained in a later section of this paper.

Roadmap: The following sections describe the datasets we have used, our methodology and constraints as well as experiments that we have conducted so far along with their evaluation. We conclude by discussing related work, advantages and some extensions of our system.

3. METHODOLOGY

In this section, we discuss the methodology employed by us. Figure 2 depicts the various stages in a concise manner.

3.1 Dataset and Preprocessing

We have used a corpus of 10,000 web tables from Web Data Commons which were extracted from the Common Crawl[12]. This dataset contains a wide variety of tables along with metadata about title, url, type of orientation etc. However, there is a lot of missing information and noisy data. In order to clean these tables, as a preprocessing step we have used the tool developed by Venetis et al [1] to identify schema header rows and the subject columns. Subject columns are the most informative columns of the table. It is very likely that users queries are based on these columns. Therefore, in order to make the search process effective, we are storing subject column identifiers in an inverted index. This allows us to identify an initial set of tables that contain the topic entity that a user might query.

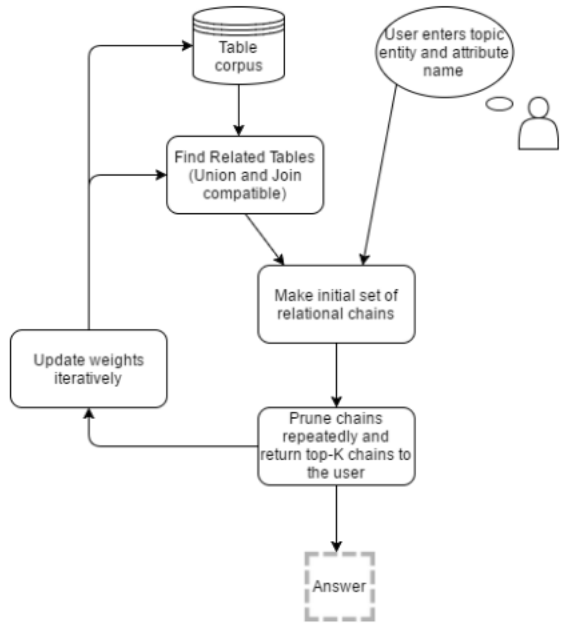
Table corpus
User enters topic entity and attribute name
Find Related Tables (Union and Join compatible)
Make initial set of relational chains
Update weights iteratively
Prune chains repeatedly and return top-K chains to the user
Answer

Figure 2: Relational chain with topic entity 'USA' and attribute 'president'

3.2 Finding Related Tables

The web corpus has a myriad of heterogeneous tables that contains some metadata information about each table. The general notion of the relation between two tables is constructed using the semantics of the columns or attributes of the two tables, with the web tables the metadata is not complete and requires inferring from the cell values. As mentioned in the previous section, we first identify subject columns and schema rows by using the tool specified by [1]. The common idea for finding the relatedness between two tables T1 and T2 is that their content could have been merged together to form a single virtual table.

Countries and their Major Cities		
Country	Major City	Currency
USA	NewYork City	USD
Germany	Berlin	Euro

Table 1: Countries and their Major Cities

In order to analyze the problem of relatedness we have defined certain framework, that includes: For a given pair of tables T1 and T2 , it is said to be related if there exist a virtual table T such that T1 and T2 are the outcomes on applying two queries Q1 and Q2 over the virtual table T. The given table T should be coherent, in terms of context and semantics. The two queries Q1 and Q2 should hold similar structure , i.e both should have the same order of the selection and projection.

For example, consider Table 1 which depicts some Countries and their Major Cities. Also consider Table 2 which also has

Country Data		
Nation	Major City	Currency
India	Mumbai	Indian Rupee
Australia	Melbourne	Australian Dollar

Table 2: Country Data

some information about countries. By observing the schema of the two tables (though the attributes are named differently - country and nation for instance), we can see that they are compatible to perform a union operation. We can now say that Table 1 and Table 2 are Schema Complement to each other. Also consider Table 3 which depicts informa-

Countries and Capitals	
Country	Capital
United States	Washington DC
Australia	Canberra

Table 3: Countries and Capitals

tion about countries and their respective capitals. Though this table has different schema when compared to Table 1 and Table 2, we can see that both Table 1 and Table 2 can be joined with Table 3 as they all have a common joinable attribute i.e. country (or nation). We can say that Table 3 is Entity Complement with Table 2 and Table 3.

The framework described above can be further defined using certain formalisms that are as follows:

1. **Entity complement:** In a nutshell the two tables T1 and T2 are entity complement of each other if there exists a virtual table T, provided that Q1(T) = T1 and Q2(T) = T2 where,

   - The queries take the form $Qi(T) = \sigma Pi(X)(T)$, where the X contains the list of attributes that belong T and the P is the selection predicate applied over the attributes Xi on table T.
   - The queries Q1 and Q2 in combination covers all the tuples of T and also that tuples of $Q1 \cap$ tuples of $Q2 \neq \phi$.
   - In addition each query Q_i renames a set of attributes A (same for different Q_i) with the restriction that $\exists A' \subseteq A, A' \rightarrow X$ in T.

   In short T1 and T2 are obtained by applying different selection predicates P1 and P2 on the same set of attributes A in T , and apply projections that include the key attributes with respect to A. Furthermore the closeness of the selection conditions can be regarded by the degree of coherence of tables T1 and T2. Thus we can deduce that the entity complement tables T1 and T2 are unionable over the attributes, 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.

   Properties of EC:
   EC obeys the commutative property, i.e If T1 is EC to T2 then T2 is EC to T1. But, it does not follow the transitivity property, i.e if T1 is EC to T2 and T2 is

EC to T3, then T1 need not be EC to T3 as T1 and T3 may not have a common joinable column.

2. **Schema Complement:** Given two tables T1 and T2, they are said to be schema complement of each other if there exists a virtual table T provided Q1(T) = T1 and Q2(T) = T2 where,

   - The query is defined as $Q_i(T) = \pi_{A_i}(T)$ where the set of attributes is to be projected.
   - A2 \A1 = ϕ, $A_1 \cup A_2$ covers all T's attributes, and $A_1 \cap A_2$ covers key attributes of A_1 and A_2 (i.e., $\exists X \subseteq A_1 \cap A_2, X \rightarrow A_i$).
   - In addition each query Q_i applies a fixed selection predicate P over the set of key attributes X.

   In short the table T2 contains the same set of attributes as T1 but these attributes may have different names and still be semantically related. In comparison to entity complement, schema complement can be said to form the union-able tables.

   Properties of SC:
   SC obeys the commutative property, i.e If T1 is SC to T2 then T2 is SC to T1. It also follow the transitivity property, i.e if T1 is EC to T2 and T2 is EC to T3, then T1 need not be EC to T3 as T1 and T3 may not have a common joinable column.

3.2.1 Computing Entity Complement

We have identified a two-stage process to determine if two tables are Entity Complement of each other. The stages are described as follows:

- Given the description of the tables we have used the cosine similarity [9], a common vector based similarity measure which transforms the input string into vector space and then performs inner product of those vectors to determine the similarity between them. We used this technique to compare the similarity between the context of the two tables. Two table descriptions that have a high cosine similarity score imply that both the tables have a similar context. We compute cosine similarity as follows:

$$\text{cosine similarity} = \frac{A.B}{|A||B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

- Further we use freebase entity ids from a reverb, linked to freebase, KnowItAll [14] to deduce EC such that two rows in two tables correspond to the same entity if and only if the cells in their column set map to the same Freebase entity identifier. In other words, we treat Freebase identifiers as the golden standard for entity resolution. In order to find the relatedness between tables based on the freebase entity ids, we have each entity e_i (value of subject column from table Ti) with labels: $L(l_i) = \{l_i^1, l_i^2, ...\}$ to each entity e_i, and we represent the set of labels by L(e_i). These labels are contributed by the freebase entity ids that were computed for each entity e_i. The relatedness between two entities e_i and e_j is calculated using the following formula:

$$reu(e_i, e_j) = L(e_i) \cap L(e_j) \tag{2}$$

where e_i and e_j are values from subject columns of tables T_i and T_j respectively. In order to calculate relatedness between both tables, we must consider the domains which they express. A domain that is more general in nature must be given lesser importance than a more specific domain. For example, a table that gives information about Sports is more general when compared to a table about Tennis. We calculate the Domain Score of two tables T_i and T_j by using the following formula:

$$DS = \frac{1}{|S_i| + |S_j|} \tag{3}$$

where S_i and S_j denote the number of subject columns in T_i and T_j respectively. As mentioned earlier, a subject column contains the most important data of a table and so it can be said that it also represents the domain to which a table belongs.

We then calculate table relatedness score by applying the following formula:

$$\alpha = \frac{1}{|E_i|.|E_j|.DS} \tag{4}$$

$$Score(T_i, T_j) = \sum_{E_i \in T_i, E_j \in T_j} \alpha \sum_{e_i \in E_i e_j \in E_j} r(e_i, e_j) \tag{5}$$

Where E_i and E_j are the set of entities of T_i and T_j respectively.

The final Entity Complement Scores of T_i and T_j can be calculated by assigning 30% weightage to cosine similarity score and 70% weightage to Score(T_i,T_j). Hence, the equation is:

$$\text{Final EC} = 0.3 * \text{cosinesimilarity} + 0.7 * \text{Score}(Ti, Tj) \tag{6}$$

These factor values were finalized from the continuous experiments performed over the set of tables from our corpus. Finally, if the Final EC score is above a certain threshold (in our case, 0.4), then we say that the two tables T_i and T_j are Entity Complements of each other.

3.2.2 Computing Schema Complement

We have identified a two-stage process to determine if two tables T_i and T_j are Schema Complement of each other. Before we begin the two stages, all rows in table T_i are compared to all rows in T_j to check the row similarity between the two tables. If the two tables T_i and T_j have tuples such that tuples(T_i) $\cap$ tuples(T_j) = tuples(T_i) then the two tables are not considered to be schema complements of each other. In order to perform this comparison in an efficient manner, we use Jaccard Similarity [8] where the coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. It can be expressed as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A| \cap |B|} \tag{7}$$

If the Jaccard Similarity scores for two tables is 1, then this means that the tables T_i and T_j are duplicates of each other and no new information can be gained by performing a union operation.

If the Jaccard Similarity is less than 1, then we must proceed with the following steps:

- We perform the semantic analysis between the attribute names using a natural language processing tool, WordNet.[15]. This tool groups the synonyms of the words into a set known as synset. For two attributes A_i and A_j, if
$$|\text{synset}(A_i) \cap \text{synset}(A_j)| > 0 \qquad (8)$$
Then, we can say that A_i and A_j have similar semantic meaning. For finding the semantic similarity between two tables, we first find the synset of all the attributes within each table. Two tables T_i and T_j are schema complement iff there exist a one to one mapping from $A_i \in T_i$ and $A_j \in T_j$ that satisfies equation (8).
- If condition defined in the above step is satisfied then, we use the same technique as Entity Complement that defines the cosine similarity of the context of the tables. The formula is expressed in equation (1). The final Schema Complement Scores of T_i and T_j can be calculated by assigning 50% weightage to cosine similarity score and 50% weightage to Domain score defined in equation (3). Hence, the equation is:
$$FinalSC = 0.5 * \text{cosine similarity} score + 0.5 * DS \qquad (9)$$

These factor values were finalized from the continuous experiments performed over the set of tables from our corpus. Finally, if the Final SC score is above a certain threshold (in our case, 0.2), then we say that the two tables T_i and T_j are Entity Complements of each other. We store the captured relatedness information and scores for later use during table cell search.

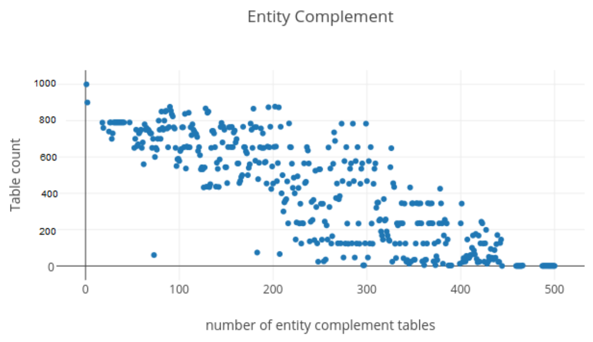

Figure 3: Entity Complement

3.2.3 Scaling

Finding related tables (both EC and SC) from a large corpus of tables is computationally expensive. As we have used a corpus with around 10,000 web tables, a mechanism to reduce the number of comparisons is necessary. For Schema Complement tables, we can use the properties of SC as defined in section 3.2 i.e. Commutativity and Transitivity to improve scaling in this phase. When a table T1 is found to be Schema Complement of another table T2, then we may also make an entry that stores T1 as SC of T2 (commutativity) and hence T1 and T2 will not be compared with each other again. Further, when table T1 is Schema Complement of T2 and T2 is SC of T3, then it follows that T1 is SC of T3 (transitivity) and so we need not compare T1 and T3. This marginally reduces the amount of computation required. However, there is room for improvement and this has been further detailed in section 6 since the main aim of this paper is to leverage relatedness for cell search rather than providing an efficient scaling technique to find relatedness. Using the constraints described above, we found

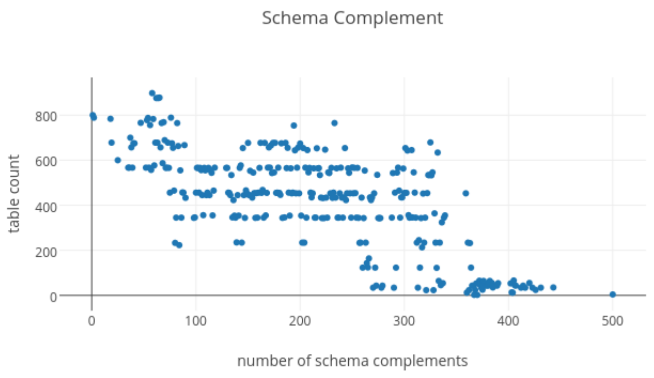

Figure 4: Schema Complement

the set of Entity Complement and Schema Complement tables for the corpus of 10,000 tables. On a normal machine with 8GB RAM 1.6 GHz Intel Core i5, this operation took over 15 hours to complete. With the help of a cluster, this could have been much faster. Figure 3 and Figure 4 depict the Entity Complement and Schema Complement counts for our dataset.

3.3 Table Cell Search

In this section we have described our methods to perform cell search as follows:

First, the user enters a topic entity and an attribute (pertaining to that topic entity) in which the user is interested. For example, consider the user query "What is the capital of USA?" Here, the topic entity is USA and the attribute required is capital. We now use the inverted index that was created during preprocessing of data. This inverted index contains subject column identifiers (represents the values that belong to the subject column of all tables) and the list of tables associated with each of these values.

We find all subject identifiers that have the same Freebase entity ids as the topic entity, in this case, USA as given by the user. This allows us to get a list of top-K tables that contain this topic entity. By top-K tables, we mean those tables that have higher scores. The scoring of tables will be explained in the next section. By only considering the Top-K tables, we may stop searching the inverted index once we have K different tables. This reduces computation by a large margin.

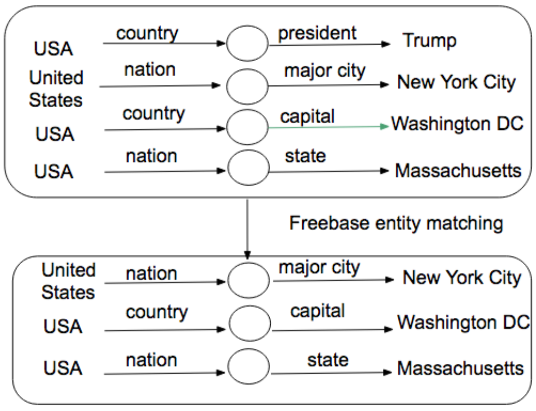

Figure 5: Pruning Relational Chains using Freebase

Next, we use the information on related tables to find the tables that are related to the list of tables found in the above step. Here, new tables will be added to the list as it is not necessary that a topic entity is present in a particular table but the answer could be reached by performing a join on two tables. (Note: we do not actually perform this join operation to reach the answer cell).

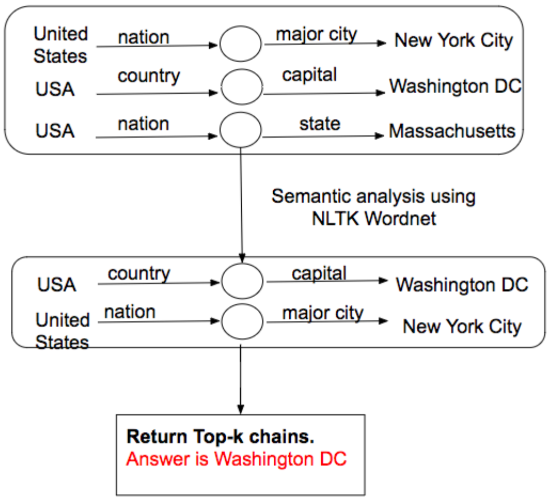

Figure 6: Pruning Relational Chains using WordNet

For the next step, we are using relational chains. As explained in Section 2, a relational chain can be defined as the relationship between a cell in one column and a cell in a different column (but in the same row). This relation can be depicted as a chain where the cell value is a node and the edge is labelled with the corresponding column name. Figure 1 depicts a relational chain.

For the given example, we form relational chains as shown in the first stage of Figure 5. Note that attribute names country and nation though different, will still be considered due to similar semantic meaning. Further, USA and United States are both eligible as they will have the same Freebase Entity ids as the topic entity itself.

These relational chains may be duplicates or have values that are completely irrelevant with respect to the attribute requested by the user. Also, the relational chains formed can have the topic entity and the attribute (represented by a single chain) belong originally to different tables. The chains are still formed as we are leveraging relatedness information. The next stage is to prune these relational chains in order to remove the ones that are not relevant and reach the required answer.

First, we find Freebase Entity ids of the attribute entered by the user as well as the attribute names in the relational chains using KnowItAll [14]. We prune all chains that do not have any common Freebase Entity ids with the user entered attribute. The pruned list of chains can be seen in the bottom of Figure 5. It is clear that the attribute 'president' was pruned as it would not have any common Freebase Entity ids with the attribute 'capital'.

In the next stage of pruning, we apply semantic analysis using NLTK's WordNet [15]. This tool groups the synonyms of the words into a set known as synset. We find all synonyms of the user entered attribute and compare them to the synonyms of all remaining attributes from the list of relational chains. All chains that have attributes with no semantic relationship to the user entered attribute are then pruned. This can be seen in Figure 6.

From the remaining list of relational chains, we return only top-K chains that have high column scores (explained in section 3.4). This vastly reduces number of comparisons required and we also reach the answer cell quickly. The value of K can be altered depending upon the user's information needs. For our example, we have assumed K to be 1.

This technique allows us to perform an accurate and efficient search over a large corpus of tables by performing fewer comparisons at each stage. Our technique has been evaluated and analyzed in a Section 4.

3.4 Performance Boosting

In order to iteratively improve the search performance, we introduce the concept of weights in table cell search. In addition to the table relatedness scores from section 3.2, we also assign weights to every table as well as the column of each table. Weights are initialized as follows:

- Table weight: Number of subject columns
- Column weight: Uniform weight of 0.5 over all columns

When the result to a query is returned by the system, we analyze the top K returned answer cells and update the weights of corresponding columns and tables of the answer chain using the following equations:

$$w_T = \frac{1}{\alpha^2}.w_T + w_T \tag{10}$$

$$w_T = w_T - \frac{1}{\alpha^2}.w_T \tag{11}$$

where (alpha) is a coefficient given by, $\alpha = 0.05 * w_T$, where w_T is the current weight of the table or column as the case may be. Tables and columns that contain the returned answer will be rewarded as shown in equation (10) whereas the other tables are penalized as shown in equation (11).

By applying the above method, we ensure that over a large number of iterations, search performance improves as there will be fewer number of comparisons required and the answer cell will be reached faster (as we consider Top K tables and columns based on their weights during our pruning process). Such a technique would definitely be beneficial in a closed domain such as enterprise data where some tables will obviously be more important than others.

4. EVALUATION

In order to evaluate our framework, we have chosen to use a KB-Based QA Dataset i.e. WebQuestions [17]. This is a popular dataset for benchmarking QA engines, especially ones that work on structured knowledge bases. First, we compare the different techniques used by our framework. We consider 3 cases:

i) When the system has found all related tables but without a performance boosting mechanism.
ii) When the same set of queries is run repeatedly over 5000 iterations having already found related tables.
iii) When the system has all tables without the information of their relatedness.

We tested the above scenarios by running a set of 50 queries from the WebQuestions Dataset on our corpus of tables. In case (ii), the same set of queries are run repeatedly over 5000 iterations so that we can observe and analyze a performance boost, if any.

One of the important factors of any table search technique is the Average Response Time. This can be defined as the average time taken for the system to find the required answer cell from the corpus of tables and return it to the user. We analyze this in Figure 7. It is evident that the average

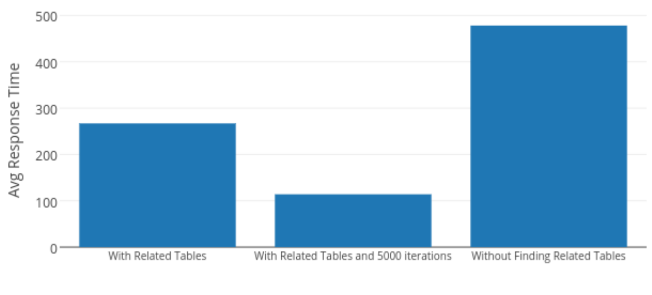

Figure 7: Technique vs Average response time

response time has improved over a large margin after 5000 iterations (case ii) in comparison to the case without performance boosting (case i). Also, we notice that finding related tables vastly improves the speed of answer cell retrieval by comparing the Average Response Time values for cases i and iii. This shows that the combination of both techniques (finding related tables and performance boosting) suggested by this paper, provide an improvement in response time.

Another important factor is the accuracy of the returned result. We define accuracy as the number of correct answer values returned by the system over the number of queries given. The number of queries used for testing in this situation is 50.

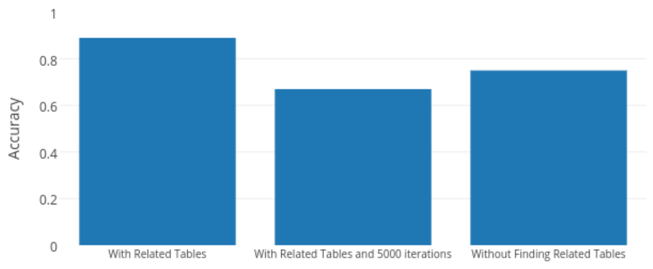

Figure 8: Technique vs Accuracy

We can see from Figure 8 that finding related tables without performance boosting (case i) has a good impact on accuracy. This might be due to the fact that, we find answer cells even if the user entered topic entity and the required attribute are in different tables. As in case iii, we do not consider table relatedness so the accuracy is not as good as that achieved by case i.

Case ii has the lowest observed accuracy. This is probably because we are reducing the scores of tables and columns that do not return the answer during a particular iteration. However, it is possible that at a later time, a query might ask for data that is contained in a heavily penalized table or column. This may not return the right result as the table which contains the answer might have a very low score and hence will not be considered in the Top-K list. This is a trade-off that must be considered by the user. If time is of the essence, then the performance boosting technique is the way to go, otherwise if the importance must be placed on accuracy, performance boosting techniques described in this paper may not always be a good idea.

Further we analyze our performance boosting technique by varying the number of queries that are passed to it over 5000 iterations. These queries are passed repeatedly and hence table and column scores will depend on the frequency of these queries.

Here we have 3 variations to the number of queries passed:
i. A set of 3000 queries
ii. A set of 500 queries
iii. A set of 50 queries

From Figure 9, we notice that we get the best average response time after 5000 iterations when we have a set of 50 queries. This is because, the tables and columns that contain the results to these queries would have been rewarded over time and so, the following iterations will show an improved response time as fewer comparisons are made.

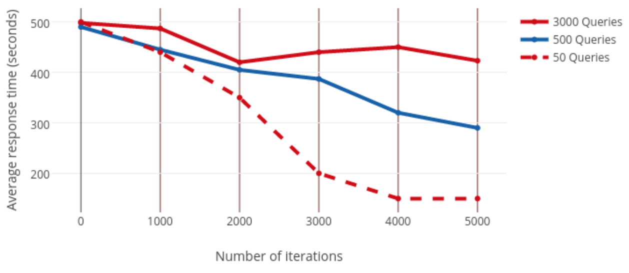

Figure 9: Performance Analysis

When we passed 500 queries, the performance gain is not as obvious as in the previous case. This is because there are a larger number of queries and hence a larger number of tables that are considered. Perhaps, over another 1000 iterations, a better response time could be expected.

On passing 3000 queries, we do not really see any effect on the Average response time as each of these queries are unique and may access unique tables each time. So, the performance boosting technique described here will not have much of an effect over just 5000 iterations.

The above analysis clearly shows that the techniques described in our framework provide a vast improvement in table cell search with respect to Average Response Time and Accuracy. We also see that a system that gets the same set of repeated queries over many iterations shows a marked improvement in response time.

5. RELATED WORK

Leveraging information about table relatedness in order to perform cell search is not a vastly explored area. While schema matching has always been an active research area in data integration, new challenges are faced today by the increasing size, number and complexity of data sources and their distribution over the network. Data sets are not always correctly typed or labeled and that hinders the matching process.

Prior work [3] on restructuring web tables into structured enterprise data tables focuses on linking the data to map cell values with instances and column headers with known data types. This significantly improves the quality of matching results and decision making. For unmatched attributes, tags are collected from Freebase and learning is performed using machine learning techniques like boosting or stacking. Cosine similarity and pearson product-moment correlation coefficient are used for type analysis. While this paper seems to make important contributions, it did not scale well on large datasets. Also, the method relies solely on linked data (method of publishing structured data so that it can be interlinked to give better query results), but does not consider synonimity between columns or attributes. For example: Attributes that have the same semantic meaning could be named differently like language and dialect. The following paper discusses a solution to alleviate this problem.

Cafarella et al. [4] proposed an attribute synonym finding tool, based on ACSDb [5], which identifies the synonymity between a pair of schema attributes. It is built upon the idea that two synonymous attributes will never appear together in same schema, but they will appear in the similar context i.e their co-attributes come from the same domain. The ranking algorithm takes a set of context attributes, C, a input and ranks all possible pairs according to the below formula. The pairs with syn score greater than a particular threshold are considered as synonymous.

$$syn(a, b) = \frac{p(a)p(b)}{\epsilon + \sum_{z \in A}(p(z|a, C) - p(z|b, C))^2} \tag{12}$$

In both the above mentioned papers, schema matching is performed by applying machine learning techniques and finding synonimity between attributes.

In the work done by A. Das Sarma et al. [2], the results of table cell search are improved by identifying the tables that are ranked much lower for the query, but they are more related to the top ranked results. They assign weighted labels to each entity. These labels are a combination of WebIsA, Freebase and WebTable labels. The relatedness between two entities is computed as a dot product of the labeled vectors. However, entire tables (that are considered relevant) are returned to the user for information summarization or reusing data to find new patterns. Also, the authors have manually evaluated their approach to table cell search without having experimented on real-world data (like QA datasets) as done in this paper.

The SCPRank algorithm proposed in [6] uses symmetric conditional probability (SCP), to measure the correlation between a cell in the extracted database and a query term.

$$scp(q, c) = \frac{p(q, c)^2}{p(q)p(c)} \tag{13}$$

SCPRank computes a series of per-column scores, each of which is simply the sum of per-cell SCP scores in the column. A table's overall score is the maximum of all of its per-column scores. Finally, the algorithm sorts the tables in order of their scores and returns a ranked list of relevant tables.

Given heterogenous data, we have developed a semi-structured schema in which we find related tables using a combination of techniques mentioned above. In contrast to previous work, we extract cells in the corpus of web tables that answer a particular query. In order to do this, we need to create links between data cells.

The concept of relational chain in introduced in [7]. This relational chain connects two cells in a table and represents the semantic relation between them. They are generated using search engine snippet matching. Deep neural networks are used to identify the best matched relational chains for the given query and then corresponding answer cells are extracted. However, this technique seems to be too expensive when faced with large amounts of data. Even though web tables have schema, it is not easy to parse millions of tables at once while looking for a particular answer. The authors do not provide any framework to improve search performance over time. Instead, the approach used here always requires the same amount of computation for every set of queries (frequently asked questions are not considered).

In past, there has been quite some work related to question answering framework over structured data using information retrieval techniques and freebase data [13]. To the best of our knowledge no other work has been done using the exact combination of techniques described in this paper.

6. FUTURE WORK

The computation of table relatedness score for every pair of tables in the corpus is very expensive especially when there are over a million tables in the corpus. One suggestion to scale up the number of tables is to consider having certain filters that reduce the number of computations required and improve the efficiency of comparison of two tables.

A filtering approach for finding EC and SC can involve using hash functions. This is advantageous to reduce the number of comparisons needed as well as to increase the speed of these comparisons. By bucketizing the set of all tables, we can perform comparison only for those tables that appear together in the same bucket. A similar technique has been used for canopy formation in de-duplication in [16]. Since there may be multiple hash values for each table, every table may go in multiple buckets. Parallel computing technologies like MapReduce can be used to perform comparisons in all buckets simultaneously. If multiple filtering conditions are used, we need to compare two tables only if the previous filtering condition is satisfied. This would reduce the computation required by a large margin and hence fasten the process of each comparison.

In terms of pruning relational chains while performing table cell search, additional complex techniques such as employing Deep Neural Networks can boost accuracy of the search result.

7. CONCLUSION

In this paper, we proposed an end-to-end framework to precisely locate table cells in millions of web tables for question answering. Our table cell search framework first found related tables based on certain criteria defined by us. We then described an efficient technique to form and prune relational chains in order to reach the answer cell. A novel approach to improve cell search efficiency in closed-domain scenarios such as enterprise tables was defined and analyzed. We evaluated our framework with state-of-the-art KB-based QA systems. Through extensive experiments, we showed that our framework could outperform other systems by a large margin on real-world questions.

8. REFERENCES

1. 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.
2. A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee,F. Wu, R. Xin, and C. Yu. Finding Related Tables. In SIGMOD, pages 817-828, ACM, 2012.
3. Ahmad Assaf, Eldad Louw, Aline Senart, Corentin Follenfant, Raphael Troncy and David Trastour. Improving Schema Matching with Linked Data. WOD May 2012
4. M.J. Cafarella, A. Halevy, D.Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the Power of Tables on the Web. VLDB, 1(1): 538-549, 2008.
5. http://web.eecs.umich.edu/ michjc/data/acsdb.html
6. M.J. Cafarella, Alon Halevy, Nodira Khoussainova. Data Integration for the Relational Web. VLDB '09, August 24-28, 2009.
7. Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan Table Cell Search for Question Answering. ACM 978-1-4503-4143-1/16/04.
8. https://en.wikipedia.org/wiki/Jaccard_index
9. https://en.wikipedia.org/wiki/Cosine_similarity
10. D. Pinto, M. Branstein, R. Coleman, W. B. Croft,M. King, W. Li, and X. Wei. Quasm: A System for Question Answering using Semi-Structured Data. In Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries, pages 46-55, ACM, 2002.
11. G.Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching web tables using entities, types and relationships. VLDB, 3(1-2):1338-1347, 2010.
12. http://webdatacommons.org/webtables/
13. Information Extraction over Structured Data: Question Answering with Freebase Xuchen Yao and Benjamin Van Durme, Johns Hopkins University Baltimore, MD, USA
14. http://knowitall.cs.washington.edu/linked_extractions/
15. http://www.nltk.org/howto/wordnet.html
16. M.A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In SIGMOD, 115.
17. https://github.com/brmson/dataset-factoid-webquestions.