Extending Tables using a Web Table Corpus

by

Saeed Sarabchi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

The web contains a large volume of tables that provide structured information about entities and relationships. This data may be used as a source for exploratory searches and to gather information about desired entities. This thesis focuses on one particular exploratory search where given a query table and a corpus of web tables, the goal is to find a ranked list of additional columns (from the table corpus) that describe the entities of the query table. We refer to this task as "table extension."

There are challenges in performing a table extension. A main challenge is that in the absence of schema information for web tables, it is not often clear which tables and/or columns may be relevant to the query. Also, multiple related columns may represent the same concept and this can lead to duplicate columns in the extended table. In this thesis, we propose a 5-step framework to address these challenges. Our framework establishes functional dependency relationships between columns and uses those dependencies in identifying more appropriate extensions. Duplicate columns are also detected and consolidated through some form of clustering. We evaluate our framework on a publicly available gold standard containing 233 web tables, using DBpedia as ground truth. Our evaluation reveals that the number of unique relevant columns extended by our proposed solution is on average 3 times more than that of two state-of-the-art baselines. Furthermore, the precision of extending a table using our method is higher than that of both baselines, meaning that fewer irrelevant columns are retrieved.

Acknowledgements

Firstly, I would like to express my deepest and sincerest gratitude to my supervisors, Professor Davood Rafiei and Professor Joerg Sander. This work would not have been possible without their utmost support and guidance. Prof. Davood Rafiei and Prof. Joerg Sander put a great deal of time and energy into my research by bringing brilliant ideas and encouraging me to think deeper and more critical about problems, while providing insightful and constructive feedback. They were extremely patient and understanding in times that I needed help. It was an honour for me to have such great mentors, in both personal and professional manner.

I would like to thank Professor Osmar Zaiane, whom I learned a lot from during my MSc program.

I would also like to thank my friends Ehsan Kamalloo, Ali Jahani, Sepehr Kazemian, Mohammad Motallebi, Farzane Aminmansour, Kamyar khodamoradi, Shadan Golestan, Athar Mahmoudinezhad, Anahita Dousti, Sara Soltani-nezhad, Amir Pournajib, Nouha Dziri and all others for making this program more fun and interesting for me.

Lastly, I would like to thank my family, especially my mother and my father, for their unconditional love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The web contains a vast corpus of tables, consisting of relational and non-relational tables. Relational tables provides structured data about entities and relationships and may be used as a source for exploratory searches and to gather information about entities [1–4]. This is in contrast to HTML tables that are solely used for formatting text. In this thesis, we focus on relational HTML tables, which we refer to as *"web tables"*. A challenge regarding using web tables as a data source is the fact that there is no explicitly defined schema attached to a web table. Although a number of web table datasets use some form of heuristics to identify the key column and the headers of a web table, that only gives a partial view of a web table's schema.

One particular exploratory query, which is our focus in this thesis, is given by a set of entities of interest, for which we want to find more information in the form of columns that describe the given entities. We may also require the additional columns to be ranked with regard to their relatedness to the given entities. We refer to this query as "*table extension*."

1.1 Motivating Example

Suppose a user wants to gather information about a table that describes a set of countries shown in Table 1.1. We refer to this table as a *query table*. Given a corpus of tables consisting of 6 relational tables about countries, companies, cities and caves as depicted in Tables 1.2, 1.3, 1.4, 1.5, 1.6 and 1.7, a possible extended table is shown in Table 1.8. It would be helpful for the user to see a

Countries
Canada
Spain
USA
India

Table 1.1: Sample query table

ranked list of extended columns based on the “relatedness” of the
extended columns to the provided country entities, so that “more related” columns come before “less related” ones as is the case for Table 1.8.

Country	**Capital**	**Latitude**	**Longitude**
England	London	51.50° N	0.12 W°
Canada	Ottawa	45.42° N	75.69 W°
Spain	Madrid	40.41° N	3.70 W°
India	Delhi	28.61° N	77.20 W°

Table 1.2: First Table about countries, with key column ’Country’

India	New Delhi
France	Paris
USA	Washington D.C.
Canada	Otawa

Table 1.3: Second Table about countries, with the first column as the key

Country	**Capital**
India	New Delhi
Canada	Ottawa

Table 1.4: Third Table about countries, with the first column as the key

Company	**Country**	**NASDAQ Symbol**
Apple	USA	AAPL
Infosys	India	INFY
Shopify	Canada	SHOP

Table 1.5: Table about companies, with “Company” column as the key

World Rank	**City**	**Country**	**Mayor**
30	Bangalore	India	Padmavathi G
48	Ahmadabad	India	Gautam Shah
112	Toronto	Canada	John Tory
114	Chicago	USA	Rahm Emanuel
183	Montreal	Canada	Valérie Plante
197	Barcelona	Spain	Ada Colau Ballano
200	Agra	India	Sri Aditya Nath Yogi
237	Dallas	USA	Mike Rawlings

Table 1.6: Table about largest cities with their mayors, with key column ’City’

Cave	Length (m)	Depth (m)	Country	Continent
Kazumura Cave	65,500	1,101	USA	North America
Hellhole Cave	67,500	225	USA	North America
Hölloch	200,421	938	Switzerland	Europe
Siebenhengste-Hohgant-Höhle	164,500,000	1,340	Switzerland	Europe
Grotte aux Fées	3,630	249	Switzerland	Europe
Cadomin Cave	2,791	220	Canada	North America
Castleguard Cave	20,357	384	Canada	North America
Cuevas del Drach	115	12	Spain	Europe
Grotto of Casteret	500	60	Spain	Europe

Table 1.7: Table about caves, with key column 'Cave'

Country	Capital	Latitude	Longitude	Continent	Cave	City	Company	Mayor
Canada	Ottawa	45.42 N°	75.69 W°	North America	Cadomin Cave Castleguard Cave	Montreal Toronto	Shopify	John Tory Valerie Plante
Spain	Madrid	40.41 N°	3.70 W°	Europe	Cuevas del Drach Grotto of Casteret	Barcelona	-	Ada Colau Ballano
USA	Washington D.C.	38.90 N°	77.03 W°	North America	Kazumura Cave Hellhole Cave	Dallas Chicago	Apple	Rahm Emanuel Mike Rawlingse
India	New Delhi	20.59 N°	78.96 W°	Asia	-	Bangalore Ahmadabad	Infosys	Padmavathi G Gautam Shah Sri Aditya Nath Yogi

Table 1.8: Sample extended table

Consider the alternative scenario where the user wants to put together an extended table such as Table 1.8 by finding the attribute names related to the entities in the query column. For each attribute name, she may search for the values associated with each of the entities. This task can be difficult because the user may not already know all the related attributes to the entities of interest and she might miss some of the related columns. Also, the related columns can vary based on the tables in the given corpus. As an example, if the corpus has geographical data about locations, then an output table may contain columns such as capital, coordinates and area of the input countries. However, if the given corpus has financial data, then the user may be able to augment her query with related financial columns such as GDP and currency and she may not find related location data.

1.2 Problem Statement

For simplicity, we assume that each table in our given corpus as well as the query table can have multiple columns, but only one of those columns is tagged as the table's key column. Plus, we consider the key column of the query table as the ***query column*** from now on.

Definition 1. *Table Extension: Given a query column Q consisting of values* $\{q_1, q_2, \ldots, q_m\}$*, plus a corpus of tables* $T \in \mathcal{T}$*, a table extension of Q with respect to* $\mathcal{T}$ *is a ranked list of additional columns* $[c_1, c_2, \ldots, c_k]$ *satisfying the following conditions:*

- *Each* $value_{i|1 \leq i \leq m}$ *in* $c_{j|1 \leq j \leq k}$ *is either an atomic value or a set of atomic values describing* q_i*.*
- *All values in column* c_i *have the same relationship with the corresponding entities of the query column Q.*
- *Each column* c_i *contains values for at least* θ_{cover} *percent of the entities in the query column Q.*

1.3 Challenges

A major challenge in a table extension is to determine which tables and columns should be selected to be used in the final extended table. A possible solution is to pick all columns in those tables that contain values from the query column. However, this solution can result in many extended columns that are not correct since in a relational table, a column is not necessarily described by other columns mentioned in that table. As an example, we can notice that not all the columns in tables containing the query column entities describe those entities (such as "NASDAQ Symbol" in Table 1.5 and "Length" and "Depth" in Table 1.7). Therefore, a viable method should be selective, picking only those columns from the input corpus that describe the query column. This is a challenging task.

We break down the columns suitable for performing a table extension into 2 categories:

- Columns in tables containing the values of the query column Q in their key column
- Columns in tables containing the values of the query column Q in their non-key column

In the following, we will investigate the questions and challenges faced when trying to identify each of the mentioned column categories.

In a relational table, each table has a key column and all non-key columns in that table describe the key column. Consequently, a method to perform a table extension can select those tables with key columns that have a high overlap with the query column and then pick all of their non-key columns since those columns describe the key of their tables, hence the query column. These selected columns contribute to the first column category. In our example, Tables 1.2 and 1.3 have this characteristic, i.e. the country columns of both tables appear to be their key columns and they highly overlap with the query column. Hence, all non-key columns of the tables can be picked as the columns of the desired extended table.

One challenge with this method is that if there are multiple tables with overlapping key columns, there is a chance that a few selected columns describe the same property or originate from the same concept. In our example, the second column in tables 1.2 and 1.3 are both characterizing the capital of countries. However, presenting duplicate columns in the output result is not desirable for users. Hence, a question is how to detect and merge duplicate columns into a single column in the output.

A challenge with merging multiple columns is that there can be multiple values for the same entity, each coming from a different table and those values may not be consistent with each other and cannot be reduced to a single value. Referring to our example, if we group the second columns in tables 1.2, 1.3 and 1.4, there would be two candidate values “Otawa” and “Ottawa” for the capital attribute associated with “Canada”. A question here is which value should be selected for the cell associated with the “Capital” attribute for “Canada” in the extended table.

A shortcoming of finding related columns by selecting tables with overlapping key columns is that we may miss suitable columns form those tables that contain the query entities under a column other than the key column. As an example, if we take a look at Table 1.7, we can notice that the key column of this table does not overlap with the query column and the column that contains values of the query column is not the tagged key column of the table. However, the “Continent” column also describes the query column. A challenge is to identify such related tables and columns in those tables that describe the query column. These columns contribute to the second column category.

Furthermore, in tables containing the entities in the query column in a non-key column, there may exist columns that could give meaningful information about the query values when aggregated

for each of those query entities. For instance, the columns "Company" in Table 1.5, "City" in Table 1.6 and "Cave" in Table 1.7, when aggregated for each of the entities in "Country" column, can give meaningful information about countries, in the sense that they specify which cities, caves and companies are located in which countries. Here the challenge is how we can identify such columns, which we will also address in this thesis.

1.4 Overview of our Approach

To perform a table extension, we first extract those tables from the corpus that are potential candidates for having suitable source columns to be used in an extended column. From those tables, we select suitable columns for performing a table extension. Then, we merge those columns that represent the same concept but coming from different tables into separate clusters. After this step, we consolidate each column-cluster into a single extended column by selecting a single value or a set of values for each column-cluster and query value. In the last step, we rank the extended columns based on their relatedness to the query column.

1.5 Research Contributions

We present a table extension framework, as depicted in Figure 1.1, where given a query consisting of a set of entities, outputs a ranked list of related columns w.r.t. the query. We address each of the steps in the framework, however our major contributions are in steps *Column Selection* and *Column Ranking*.

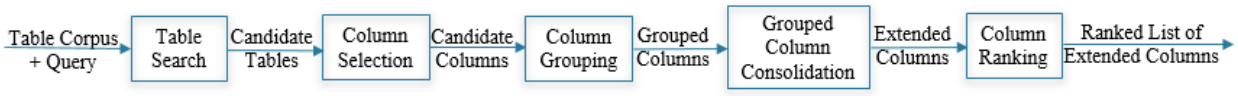


Figure 1.1: Proposed Framework

Contributions in Column Selection step: We identify related columns to a query column in tables that their *non-key* column overlap with the query column. To the best of our knowledge, this has not been studied in the previous works. For doing so, we utilize the notion of *Functional Dependency (FD)* as well as *co-occurencies between column values* to capture the relatedness between two columns.

We also study 3 methods for detecting *Functional Dependencies*: (1) *Error Threshold*, which is based on the number of violations in a table for an FD to be satisfied, (2) *Violation Detectability*, which tries to detect positive evidence in a table for an FD to hold over a schema, and (3) *Violation Detectability with Column Grouping*, which tries to find positive evidence from other tables for an FD to hold over a schema. We analyse the effects of each method on table extension results as well.

Contributions in Column Ranking step: We introduce methods for ranking an extended column based on the *Header Relatedness* and *Row Relatedness* between the query column and the extended column. We measure how effective each method is on ranking extended columns.

Moreover, we use a novel automatic evaluation method based on DBpedia [5] as the ground truth in order to cross-check the values in an extended table.

1.6 Thesis Organization

In the remainder of this thesis, we will first discuss related work. Following the related work, we will describe our proposed table extension framework. In the fourth chapter, we will present the results of our evaluations and we will conclude the thesis in the last chapter.

Chapter 2

Related Work

In this chapter, we will review the related work on *table extension* and other related work, namely *table extraction*, *table search*, *schema matching* and *table stitching*.

2.1 Table Extension

In table extension, the goal is to add relevant column(s) to an input column or table by leveraging a corpus of web tables. There are a number of works that address this task; however, the input and output in those works are not always the same. In the following, we will present some of the related works on this topic

InfoGather In this work [4], a function named *Augmentation By Attribute name (ABA)* is proposed with the following aim: given a query column Q, an attribute name a as well as a corpus of tables $T \in \mathcal{T}$, output the value corresponding to each query row $q \in Q$ on attribute a using the corpus data. For doing so, it first splits each table in the corpus into multiple 2-column-tables called *Entity Attribute Binary* tables *(EAB)*, in which the first column is the key column or the entity column of the table[1] ($T.K$) and the attribute column, denoted as $T.A$ is any column other than $T.K$. Then, those EABs that have an attribute column $T.A$ with header name 'a' are selected and scored based on their "direct" and "indirect" similarity to the query column. A direct similarity is calculated by overlap similarity between each EAB and the query, whereas an "indirect" similarity to the query is calculated via a method called *Holistic Matching*, which propagates the direct similarity scores to EABs that do not have enough overlap similarity with the query, but are similar

[1] In InfoGather, it is assumed that the key of each table is a single column.

to the direct matches. For selecting the output value corresponding to each query row, all matched values from the high-scored EABs are grouped together and a representative value of the group is considered as the output value.

ABA is similar to our work in the way that it tries to augment a query column, however, its difference is that this augmentation is only done for one column for which an attribute name is given. The strength of this work is the *Holistic Matching* which considers indirect matches with the query. However, this method has a limitation in scoring those EABs coming from the same or a similar webtable. The reason is that the similarity scores in *Holistic matching* are computed for each EAB pair based on a classification method, which assigns a relatedness score between each pair of EABs using a number of features such as EAB's content, source URL and context (i.e. a fixed number of words before and after the EAB's source webtable in its HTML page). Here, the issue is that the similarity score between EABs coming from the same source table (or even similar source tables) is very high even if those two EABs represent completely different concepts, which makes the classification results (i.e. the similarity scores) biased towards the EABs from the same (or similar) original tables. This results in poor "indirect" EAB matches to the query column which results in poor output values corresponding to query rows. The reason for this bias stems from the labelling and training process of the classification method in *Holistic Matching*. In order to get the labelled pairs of EABs to train the classifier, InfoGather searches for those EABs which slightly differ in content, and those EABs are mostly coming from the same tables or from those tables which differ only in a few rows or columns, with very similar (or even the same) source URL and context. Hence the classifier is trained to assign a high similarity score to the mentioned EABs.

Another function introduced in InfoGather is called *Attribute Discovery (AD)* that, given a query column, outputs a set of relevant column names in groups, each group consisting of an attribute-name with their synonyms. To do this, *AD* first selects those EABs with high value overlap between their key column ($T.K$) and the query column. Then it finds the synonyms for each of the EAB's attribute column ($T.A$) based on the previously calculated pairwise similarities in *Holistic Matching Framework*.

This function is different from the problem studied in this thesis since InfoGather outputs only a set of attribute-name groups, without any value attached to them, and is similar in that each of those attribute headers can be used for augmenting the query with related columns. The bias

problem in *Holistic Matching*, as discussed above, applies to the *AD* function as well, and this results in placing a number of attribute-names that are coming from the same web table and are not necessarily synonyms, in the same group.

Table extension for a query column can be implemented via combining the two aforementioned functions by first, performing *Attribute Discovery (AD)* on the input query column which results in a set of output header groups to fill the extended columns with some values. Attribute discovery may be followed by *Augmentation By Attribute name (ABA)* for each attribute header in each output header group. A problem with this method is that for each attribute header group, which represents a single concept, there exists multiple values corresponding to a query value, coming from applying *ABA* for each element in the attribute header group. A solution is to consolidate the output values for each query row in each output header group. We have implemented this solution and used it as one of our baselines for our evaluations.

In [6], the authors extend InfoGather to support numeric and time-dependent attributes. Their approach is to extract numeric units and timestamps from the contents and context of the source tables if possible, and propagate it to similar tables that lack such information, based on their previously calculated pairwise similarities.

Mannheim Search Join In this work [3], an operation is presented named *Unconstrained Queries*. The goal of this operation is the following: given a query table Q, its key column $Q.k$, and a corpus of webtables, output as many columns as possible that describe $Q.k$ using the corpus data, which is similar to our problem statement. To perform this operation, it first searches for web tables with overlapping key columns and selects those with an overlap higher than a threshold. Then it selects all of the non-key columns and puts them into separate groups, so that each group represents a single concept describing $Q.k$. Then each group is consolidated into one column and those columns are the output of this operation.

Another operation proposed in Mannheim Search Join is *Constraint Query*, where given a query table and an attribute name, it tries to fill the attribute column. This is similar to *Augmentation By Attribute Name* in InfoGather. For doing so, it selects tables where the key-column of the table overlaps with the the query key column and the size of the overlap is more than a given threshold. From those selected tables, it picks attributes with the same header name as the input header name. Then it consolidates the results for each query row from the selected attribute values.

One of the limitations of Mannheim Search Join is that it does not utilize all related columns in the corpus tables for table extension: this method discards tables containing relevant columns to the query whose key column does not represent the query column. Plus, there is no ordering for the output columns in terms of relatedness to the query table.

Octopus In this work [1], a function named *MultiJoin* is introduced similar to *Augmentation By Attribute Name* [4] and *Constraint Query* [3], with the goal of extending a query column Q with one or more additional columns which describe an input attribute name A. This is similar to Augmentation By Attribute Name in Infogather [4] and Constraint quey in Manheim [3] with the difference that the corpus of web tables is not given as an input. MultiJoin extracts tables using a web-search engine, so that for each value v in the query column, it searches for "$v + A$" over the web. All tables $t \in \mathcal{T}$ from the high-ranked result web pages are extracted and are scored based on their relatedness to the input attribute name A. The authors define this relatedness based on the co-occurrence counts of v and the text of A in web documents. For each related table t that is retrieved, MultiJoin makes a group g of tables based on *column-matching* between t and tables similar to t, and scores each group based on the weighted table-score of t and the overlap between the values in the tables in g and the input table's key values. Finally, *MultiJoin* picks the highest scoring group and uses the tables in that group for extension.

One of the limitations in MultiJoin is the way it extracts tables using a search engine, since those tables may not be necessarily related to the search query, hence the extension maybe done using unrelated data. Another limitation is the grouping algorithm that is used in this approach. The pairwise distance/similarity function used for grouping tables is based on the sum of per-column distance/similarity scores for the best possible column-to-column matching. This scoring is biased towards tables with a small number of attributes when the pairwise distances are summed, or towards tables with a large number of attributes when the similarity scores are added. In either situation, certain tables that semantically belong to a group may not be included in the cluster because of their sizes.

WikiTables In this work [7], the authors introduce a function called *Relevant Join* which takes a query table and outputs a ranked list of columns relevant to the query table from the set of relational tables in Wikipedia pages. The goal of this function is similar to the problem studied in this thesis in that it extends a query table by a set of related columns, except that it is specific for

Wikipedia tables. To do a table extension, *Relevant Join* first selects a set of candidate columns from the set of tables that have at least one overlapping column with the query table. The candidates columns include all columns other than the overlapping column. Then each column is classified as either "relevant" or "non-relevant" to the query table. This classification is similar to the *Holistic Matching Framework* used in InfoGather [4] with both table-related features (e.g. column values) as well as features specific to Wikipedia (e.g. the *in-links* and *out-links* in table source pages). Finally, the relevant columns are ranked by a feature-based ranking model with the same features used in their classification model.

One limitation of Relevant Join is that, since the output columns are directly coming from the table columns without performing any aggregation or grouping of similar columns, there can be a large number of duplicates in the output columns. Another limitation is that the labelling process in training the classifier is done manually without clear instructions, and this is a challenge for reproducibility. Also, the classifier may not be extended to tables other than Wikipedia tables, since the Wikipedia-specific features are reported as the major contributors to the classification method.

Top-K Entity Augmentation In this work [2], an entity augmentation operation is also proposed. Given a query column, an attribute name, as well as a corpus of tables, the proposed method augments the attribute column with values associated with each query key using the corpus data. This is similar to operations such as *Augmentation By Attribute Name*, *Constraint Query* and *MultiJoin* . A difference with these methods is in the value selection step, where *Top-K Entity Augmentation* outputs a ranked list of k possible augmentations for the attribute column, compared to the other operations where the augmentation is a single column. The augmentation columns in this work are constructed in a way that the number of source tables from which the values of an augmentation column come from is minimized. This is to ensure that the *consistency* between the values of each column is the highest. Meanwhile, the number of source tables constructing the augmentation columns are also maximised to ensure the *diversity* of the output.

2.2 Other Related Work

In this section, we provide an overview of related works which are not directly addressing the problem of table extension, but are applicable in some of the steps for table extension. These

works are grouped into *Table Extraction*, *Table Search*, *Schema Matching* and *Table Stitching*. In the rest of this chapter, we review each group and its relationship to the problem studied in this thesis.

2.2.1 Table Extraction

Table extraction can be considered as the initial phase for any operation on web tables, since it provides the input table corpus for any of these operations.

Early work on extracting HTML tables starts with differentiating between genuine tables, also known as relational data, from non-genuine tables used for web-page formatting, based on rule-based methods [8, 9]. Many of these rule-based methods are domain-specific, and they may not yield good results on domains where those rules are not satisfied.

Wang and Hu [10] trained a classifier to tag genuine tables. They used *layout features* such as "average number of columns and rows" and "average value length", *content type features* which deals with different data types included in a table (e.g. image, form, hyperlink, alphabetical, digit, etc), as well as a feature which deals with the content of a table and the words used in a table which the authors refer to as *word group feature*.

In 2008, Cafarella et al. [11] extracted 154 million genuine tables from a 14.1 billion HTML table corpus from Google's web crawl. Their approach was based on a classification method similar to [10], but with a richer set of layout and content features.

Lehmberg et al. [12] gathered a web table corpus containing 233 million tables from the 2015 version of the CommonCrawl[2], using a table extraction method similar to that of [11]. The table corpus constructed in this work contains a richer set of metadata for each table including table orientation, table caption, header row and key column, as well as context information such as the text before and after the table the title of the HTML page. This metadata sometimes provide useful information about the semantics of tables.

2.2.2 Table Search

The goal of table search is to find and rank tables with regard to a specific query. Table search can be considered as one of the steps in a table extension, where candidate tables are retrieved first and

[2]https://commoncrawl.org/about/

are then used to extend columns.

Early work on table search [1, 13] focused on the following problem: given a query consisting of a set of keywords and a corpus of tables, output a ranked list of tables w.r.t. the query. Cafarella et al. [13] proposed several ranking algorithms and the most effective one (referred to as *schemaRank*) ranks each table based on text-derived features such as the number of query hits (i.e. the query keywords that are found) on table header and body, as well as the coherence of attributes in the table schema. The schema coherency was computed based on the average "Pointwise Mutual Information" of all possible pairs of attributes in a schema. In [1], a ranking function is proposed which ranks the tables related to a query based on the *correlation* between the query phrase and table values, as computed based on their co-occurrence in web documents. This correlation, named "Symmetric Conditional Probability", determines how likely a query phrase and table value appear together in a web document.

Das Sarma et al. [14] proposed algorithms that take a table as input and outputs a ranked list of related tables in the sense that they are either joinable and unionable. For finding joinable tables, the authors score each table based on the overlap between the key column of the table and that of the query table, as well as a schema-relatedness measure between the two tables. They define this relatedness measure as the probability of co-occurrence between the attributes of a matching table with that of the input table. This co-occurrence probability is computed based on the *Attribute Correlation Statistics Database (AcsDB)* introduced in [13], which contains the frequencies that each pair of attributes co-occur in the same schema. In order to find unionable tables, the authors score each corpus table based on the consistency of its key column with the key column in the query table. The consistency between two sets is computed based on an Is-A database [15], which is basically a set of tuples in the form of (subject, type) where each tuple identifies the type of a subject gathered from the web. The other factor for finding unionable tables is the schema consistency between a corpus table and the query table, calculated based on their pairwise attribute matching scores, with the aid of column annotations in the Is-A database.

There are other works [15, 16] that aim at improving table search by annotating table components. In [15], the authors assign label(s) to each table column based on an Is-A database. Furthermore, they assign appropriate label(s) to each pair of columns in a table based on a "relation" database, which is a set of triples in the form of (subject, relation, object). In the triples,

"relation" denotes the relationship between the subject and the object. Both the Is-A database and the relation database are gathered from YAGO knowledge base. In this work, columns are annotated with *types*, and column pairs in a table are annotated with *relations*. The authors also annotate table cells with related *entities* from YAGO.

2.2.3 Schema Matching

The problem of schema matching can be defined as given two schemas of any type, output all possible correspondences between them. In this setting, a schema is a "formal structure that that represents an engineered artifact" [17], such as table structure, ontology definition, etc. Schema matching is directly related to the *column grouping* step in our proposed table extension framework, in that both associate columns with the same concepts to each other.

There is a large body of work on schema matching [17–19]. Rahm and Bernstein [18] proposed a taxonomy on schema matching approaches. They categorized the existing approaches into 4 broad categories, namely schema-only based matchers, contents-based matchers, hybrid matchers and composite matchers. Also, the matchers can be based on machine learning techniques [4], non-learning methods [3], or a combination of both [20].

A vast body of work has been done for matching webtables to knowledge bases [21]. This matching can be used for table annotation [16], knowledge base construction [22] and knowledge-based extension [23, 24]. Zhang et al. [23] proposed an instance-based schema mapping solution to statistically find an effective mapping between a web table and a knowledge base via the matched data examples. Sekhavat et al. [24] proposed a probabilistic method that augments an existing knowledge base with facts from web tables by leveraging a Web text corpus and natural language patterns associated with relations in the knowledge base. Dong et al. [22] augmented Freebase using web tables and other web sources. Ritze et al. [25] proposed an iterative instance and schema matching method to annotate rows with entities, columns with properties and tables with types in DBpedia.

2.2.4 Table Stitching

The goal of table stitching is to combine tables into a single meaningful union table. Table stitching is different from our work in that it tries to extend tables vertically, while we aim to extend tables

horizontally.

Lehmberg and Bizer [26] proposed a table stitching method consisting an instance-based matching phase to generate the attribute correspondences between tables, followed by a holistic correspondence refinement. Ling et al. [27] utilize attribute names in table headers to align columns for creating union tables. This method performs a table annotation technique developed earlier [15] to find proper attribute names for table columns. Gupta and Sarawagi [28] proposed a method that, given a few example rows, assembles a unified vertically extended table using lists on the web and through mapping the list records to the query schema using a statistical model. Pimplikar and Sarawagi [29] proposed a method that, given a set of query keywords and a corpus of web tables, compiles a set of tables matching each keyword. Those tables are later merged into one table.. For merging the tables, the method identified the mapping between the columns of corpus tables and each query keyword based on a graphical model that jointly maps all tables by incorporating diverse sources of clues such as matches in different parts of the table, corpus-wide co-occurrence statistics, and content overlap across table columns.

Chapter 3

Proposed Framework

In this chapter, we present our table extension framework. It consists of 5 steps, namely *Table Search*, *Column Selection*, *Column Grouping*, *Grouped Column Consolidation* and *Column Ranking* depicted in Figure 1.1. In the following, we will describe the details of each steps.

3.1 Table Search

In this step, our goal is finding tables which can be used for a table extension, i.e. those tables that describe the entities in the query column. Hence such tables must contain entities from the query column and at least one column that describes the query column. Therefore, our problem in this step can be stated as follows:

Definition 2. *Table Search: Given a set of tables $T \in \mathcal{T}$, plus a query column Q, find candidate tables containing at least a column that describes Q.*

To be qualified as a candidate, the table should have a column which characterizes the query column Q. To find this column in a table, we want the value overlap between a column of that table $T.C$ and Q to be high, so that we make sure those two columns refer to the same entities. If this measure is low, it can be an indication that the two columns may represent different concepts. To compute the value overlap between $T.C$ and Q, we use the following measure:

$$ValueOverlap(T.C, Q) = \frac{|T.C \cap Q|}{min(|T.C|, |Q|)}. \tag{3.1}$$

Where $|T.C \cap Q| = |\{c|c \in T.C \ \& \ c \in Q\}|$. As an example, if we consider Table 1.1 as Q and "Country" column in Table 1.2 as $T.C$, then the value overlap between $T.C$ and Q is

$ValueOverlap(T.C, Q) = \frac{3}{4} = 0.75$. We select those tables that have at least one column with *ValueOverlap* higher than a threshold and in each of those selected tables, we denote the column which has the highest $ValueOverlap$ with the query column as $\hat{Q}$, and because of this overlap, we assume that $\hat{Q}$ represents Q. In case of ties in $ValueOverlap$, $\hat{Q}$ is randomly chosen from the columns having the highest $ValueOverlap$ with the query column. We use an inverted index on the values that have appeared in table columns to speed up the computation for value overlap between query columns and corpus table columns.

3.2 Column Selection

In this step, we aim to select those columns from tables that describe the query column Q. The output from the *Table Selection* step is the input for this step. As discussed in Section 1.3, a challenge in a table extension is to pick suitable columns from the corpus tables containing Q, since not all the columns in those tables describe the query column. Our problem here can be defined as follows:

Definition 3. *Column Selection: Given a table T containing a column $\hat{Q}$ having high $ValueOverlap$ with the query column Q, find columns $T.C \in T$ that describe Q and output column pairs ($\hat{Q}$,$T.C$).*

The reason we output the column pair rather than the columns alone is the fact that we want to preserve the mappings between the query values and the column values, so that we can assign associated values to each query value in the extended columns.

In the following, we discuss the methods we use to select the two column categories discussed in Section 1.3, namely the columns describing Q in tables with $\hat{Q}$ as their key column, and the columns describing Q in tables with $\hat{Q}$ as a non-key column.

We use a straightforward way to pick the first column-category. We select all non-key columns in tables with a key-column that overlaps with the query column Q, since in a relational table, non-key columns describe the key-column of the table. Furthermore, these columns extend the query column with atomic values, not set-values.

In order to pick the second column-category, we first select those tables from the *Column Selection* output where $\hat{Q}$ is a non-key column. From those selected tables, we pick columns $T.C$ such that each value $q \in \hat{Q}$ is in a relationship with one and only one value $c \in T.C$. In this case,

we can argue that $\hat{Q}$ identifies $T.C$, hence $T.C$ describes $\hat{Q}$, consequently $T.C$ can be considered as a property of $\hat{Q}$, and since $\hat{Q}$ is assumed to represent the same concept as $T.Q$, then $T.C$ can be regarded as a property of $T.Q$. As an example, referring back to the motivating example in Section 1.1, in Table 1.7, "Country" can be considered as $\hat{Q}$ and "Continent" can be regarded as $T.C$, and we can notice that each value in "Country" is in a relationship with one and only one value in "Continent". Hence, similar to the first column-category, these columns extend the query column with atomic values as well.

To capture the mapping relationship between $T.C$ and $\hat{Q}$, we leverage the concept of *Functional Dependency*, since if column $T.C$ is functionally dependent on column $\hat{Q}$ (i.e. $\hat{Q} \rightarrow T.C$), then the mentioned mapping constraint will be satisfied by $T.C$. Functional dependency can be defined as below [30]:

Definition 4. *Given a relational schema R and attributes X and Y in R, a **functional dependency (FD)** constraint of the form $X \rightarrow Y$ holds in R if for every instance of R, for every pair of tuples t and s in those instances: if t and s agree on X, then t and s agree on Y.*

Based on Definition 4., a functional dependency is defined over a schema and it should be satisfied in every instance of the given schema. However, since we have no prior knowledge about the functional dependency constraints imposed over our given table corpus, we try to detect them based on the table corpus data. In other words, although defining a functional dependency is a top-down process, we try to detect it from a bottom-up perspective. As a naive solution, we consider a functional dependency as *true* if it holds on at least one instance. This solution would introduce "accidental" functional dependencies as well since an FD may hold in one instance but not in other instances. To overcome this issue, we look for evidence in other tables to make sure if an FD is true or not which we will discuss later in the chapter.

We can find related columns such as "Continent" in a table where the schema attached to the table is not normalized, and to be specific, is not in Boyce-Codd normal form (BCNF). The reason is that in such tables, a column $T.c$ is describing another column $\hat{Q}$ which is not the key column of the table. Hence the FD: $\hat{Q} \rightarrow T.C$ is present in the schema where the left-hand-side of the FD is a non-key column. This violates that the schema is in BCNF, since each left-hand-side in each FD in a BCNF schema is a super-key [30]. If we were to decompose this

non-BCNF table into BCNF tables, $\hat{Q}$ would become the key column of one of the decomposed tables. Therefore, similar to the first column-category, we would be able to pick all columns in that BCNF table (except its key column $\hat{Q}$) as the properties of $\hat{Q}$. In our previous example, Table 1.7 is a non-BCNF table and if we were to decompose this table into BCNF tables, we would have 2 tables T_1:("Cave","Length","Depth","Country") and T_2:("Country","'Continent"), and since "Continent" is a non-key column in T_2 and the key column of this table is "Country", hence, "Continent" can be regarded as a property of "Country".

The definition of functional dependency in Definition 4. can be referred to as *exact functional dependency*. However, functional dependency between $\hat{Q}$ and $T.C$ may be violated due to name ambiguity or null values introduced by low quality data in web tables [31, 32]. Hence, we use a relaxed version of functional dependency called *approximate functional dependency* to handle those violations caused by poor quality data. The idea behind this type of functional dependency is to accept a small portion of violations less than a threshold ϵ. In this case, we can claim that the functional dependency is "approximately" satisfied and we denote this type of functional dependency as approximate functional dependency. The formal definition of this type of FD is as follows [32]:

Definition 5. *Approximate Functional Dependency: Given a relational schema R, attributes X and Y in R and error threshold ϵ, an* ***approximate functional dependency*** *constraint of the form $X \xrightarrow{\approx} Y$ holds in R if, for every instance $r \in R$, there exists $r' \subseteq r$ with $\epsilon < \frac{|r'|}{|r|}$, so that for every pair of tuples t and s in r', if t and s agree on X, then t and s agree on Y.*

In Definition 5., $|r|$ and $|r'|$ are the number of rows in r and r' respectively. Also, similar to *exact functional dependency*, as a naive baseline, we consider an *approximate functional dependency* as *true* if it holds on at least one instance.

We use Definition 5. in our baseline method for detecting related columns in the second column-category. In this method, for each candidate table T from the *Table Selection* step where $\hat{Q}$ is a non-key column, we check each column C in T and determine if an *approximate functional dependency (AFD)* $\hat{Q} \xrightarrow{\approx} T.C$ holds, given an error threshold ϵ, and if yes, then we output column-pair ($\hat{Q}$,$T.C$). We call this baseline ***Error Threshold.***

Discussion on selecting the error threshold: If we lower the error threshold, fewer FDs are satisfied but with fewer number of violations. Hence the number of extended columns decreases

and from those extended columns, a larger portion is relevant to the query column. Whereas if we raise the threshold, more FDs are satisfied but with higher number of violations, hence a smaller portion of extended columns are relevant to the query column. We varied the error threshold in our experiments from 0 to 0.3, and we identified a sweet spot to balance the number of relevant columns with the portion of relevant columns in the final extension when the error threshold is set to 0.05.

Error Threshold has a shortcoming similar to that of *exact FD* detection, i.e. introduction of accidental FDs. As an example, If we apply this method to Table 1.5 assuming that the query column is about countries, then "NASDAQ Symbol" will be detected as a related column to countries since the FD $Country \rightarrow NASDAQ\ Symbol$ is satisfied over this table. However, "NASDAQ Symbol" is a property of "Company", not "Country". The reason of this "error" is that $Country \rightarrow NASDAQ\ Symbol$ is accidentally satisfied and is not true. However, if we apply *Error Threshold* to Table 1.7, "Continent" will be detected as a related column since the FD $Country \rightarrow Continent$ is satisfied in this table, which holds over the schema associated with this table as well. Here the challenge is to differentiate between the *schema constraints FDs* and *accidental FDs*.

As a naive way to address this challenge, we reject those FDs that have no "positive evidence" in their instances that shows the FD is true, i.e. holds over its associated schema. We define this "positive evidence" as having duplicate values in the left-hand-side (lhs) of the FD. We argue that these duplicate values increase the chance of violation in an FD, since if the associated values of the duplicates in the righ-hand-side of that FD differ from each other, then the FD is violated and if those associated values are the same, then we can claim with higher confidence that the FD is true. In case of having no positive evidence, i.e. no duplicates in the right-hand-side of the FD, then we take a conservative path and consider the FD as accidental, hence we reject the FD. We compute this positive evidence as below:

$$FD\_PositiveEvidence(X \rightarrow Y) = \frac{|Duplicate(X)|}{|X|}. \tag{3.2}$$

Where $Duplicate(X)$ is the multi-set of duplicate values in column X. Formally, $Duplicate(X) = \{\{v_i | v_i \in X \,\&\, \exists\, j\ s.t.\ v_j \in X \,\&\, v_i = v_j \,\&\, i > j\}\}$. We reject those FDs with FD_PositiveEvidence less than a threshold. We call this method ***Violation Detectability***. As an example, in Table 1.7,

$FD\_PositiveEvidence(Country \rightarrow Continent) = \frac{5}{9}$ and in Table 1.5,
$FD\_PositiveEvidence(Country \rightarrow NASDAQ\ Symbol) = \frac{0}{3} = 0$. Hence, given a threshold of higher than zero, we reject $Country \rightarrow NASDAQ\ Symbol$ and accept $Country \rightarrow Continent$.

The effects of varying the FD_PositiveEvidence threshold is opposite to that of the error threshold in *Approximate Functional Dependencies*, since if we lower the threshold, more FDs are accepted but with a higher chance of being accidental, hence a smaller portion of extended columns are relevant to the query column. If we raise the threshold, few FDs are accepted but with a lower chance of being accidental, hence a larger portion of extended columns are relevant to the query column. In the experiments, we varied the FD_PositiveEvidence from 0 to 0.3 and found that 0.05 is a sweet spot to achieve a balance between the portion of relevant columns with the number of relevant columns in the final extension.

Violation Detectability is a naive way of rejecting accidental FDs. One issue with this method is that the number of violations is regardless of the values in the FD's lhs. We can improve this method by associating the number of violations to each value in the FD's lhs values, so that it can differentiate between the case when more lhs values are violated versus few of them are violated. Another problem with the mentioned method is that if there is no duplicate value in the lhs of an FD, it will be rejected, since the value of $FD\_PositiveEvidence$ for that FD is zero even if it is true. For example, if we look at Table 3.1, the constraint $Country \rightarrow Continent$ holds, however, there is no duplicate value in the "Country" column. Hence, by using the *Violation Detectability* method, this table will be rejected although "Continent" describes "Country". One way to address this problem is to look for evidence in other tables. In other words, we search for duplicate values in the columns of other tables with the same left-hand-side and right-hand-side columns in the FD. Let's assume the *FD_PositiveEvidence* for FD $X \rightarrow Y$ is zero. If we can find other columns in other tables containing X and Y and merge them together, we may increase the violation chance. Hence we can be more confident in rejecting or accepting an FD. We call this method ***Column Grouping***.

As an example, if we merge the columns "Country" and "Continent" in Table 3.1 with the corresponding columns with the same name in Table 1.7, then the $FD\_PositiveEvidence(Country \rightarrow Continent)$ for the merged columns is $\frac{7}{12} = 0.58$, plus there is no violation of the functional dependency $Country \rightarrow Continent$. Hence, *Column Grouping* will pick the column-pair ("Coun-

Company	Country	Continent
Apple	USA	North America
Infosys	India	Asia
Shopify	Canada	North America

Table 3.1: A table about companies, with "Company" column as the key and no duplicates in the country column

try","Continent") from Table 3.1 to be used in the extended table. Also, by picking this column-pair, the coverage in the extended column "Continents" will increase, since the value pair ("India","Asia") did not exist before using the *Column Grouping* method, and now we can use this new value pair to extend the continent value for "India" with "Asia".

Following our efforts to pick related columns from the second column-category, i.e. columns in tables where $\hat{Q}$ is a non-key column, we can find additional columns that do not necessarily describe query column Q, but are "related" to Q and give more information about it, such as "Cave" in Table 1.7 and "City" in Table 1.6 and "Company" in Table 1.5. In order to pick such columns, by taking a closer look at column-pairs ("City","Country"), ("Cave","Country") and ("Company","Country") in the table corpus, we speculate that their pairwise value occurrences are more than pure coincidence. Hence, there could be a correlation between those value pairs over the corpus tables. As an example, there are multiple tables that contain value pairs ("Montreal","Canada") or ("Chicago","USA") in the column-pair ("Country","City"). As a result, we can speculate that "Country" and "City" can be related to each other since there is a strong correlations between the pairwise values in the column-pair. Here, a question is how we can calculate such correlations among value-pairs.

One way to calculate correlations between two values is to leverage *Point-wise Mutual Information (PMI)*, defined at row-level in tables. PMI is defined as below [33]:

$$PMI(v_1, v_2) = \log \frac{p(v_1, v_2)}{p(v_1)p(v_2)}. \tag{3.3}$$

Where $p(v)$ is the probability that a value v occurs in a table in the corpus, which can be estimated as follows:

$$p(v) = \frac{|T(v)|}{N}. \tag{3.4}$$

Where $|T(v)|$ is the number of tables that contain value v, and N is the number of tables in the table corpus. The probability of values v_1 and v_2 co-occurring in one row $p(v_1, v_2)$ can be estimated as follows:

$$p(v_1, v_2) = \frac{|T(v_1, v_2)|}{N}. \tag{3.5}$$

Where $|T(v_1, v_2)|$ is the number of tables where v_1 and v_2 occur in one row. PMI computes the logarithmic ratio between the probability that v_1 and v_2 co-occur in a row and the probability that they co-occure by chance. This measure is usually normalized to the range of $[-1, +1]$ which is called *Normalized PMI (NPMI)* based on the following formula [33]:

$$NPMI(v_1, v_2) = \frac{PMI(v_1, v_2)}{-\log p(v_1, v_2)}. \tag{3.6}$$

In order to accept or prune a column-pair, we compute the average NPMI scores over all the pairwise-values of the column-pair and if this score is lower than a threshold, we prune the column-pair.

It is worth mentioning that such columns extend the query column with set-values, because each query value may be in a relationship with one or more value. As an example, if we consider Table 1.6, the query value "Canada" is in a relationship with the set {"Toronto","Montreal"}, as well as the query value "USA" which is in a relationship with the set {"Chicago","Dallas"}.

3.3 Column Grouping

As discussed in the challenges section in Chapter 1, there is a chance that a few selected columns represent the same concept. For example, the "Capital" column in tables 1.2 and 1.3 both characterize the same concept which is the country capitals. However, it is not desirable for the user to see duplicate columns in the result table. Hence, we need to group selected column pairs from the previous step so that the columns with the same concept fall under the same cluster. To define our task:

Definition 6. *Column Grouping: Given a set of column-pairs $c \in C$, cluster all column-pairs in C into column-pair groups $g \in G$ so that each cluster g contains a set of column-pairs with the same concept.*

We use a method similar to *Agglomerate Hierarchical Clustering* [34] in order to perform column grouping. Note that hierarchical clustering is one possible choice, and alternative clustering methods can also be used, as long as the columns with the same concept are grouped together. Our reason to use hierarchical clustering in this step is that it does not require the number of clusters to be known in advance.

In our approach, we compute the clusters in this way that first, each column-pair is a cluster. Then the distances between each pair of clusters is computed and those clusters with distance lower than a *merge threshold* are merged with each other. This cycle repeats for each hierarchy level until there are no cluster with distance lower than the merge threshold or all clusters are merged into one. Also the clustering hierarchy levels can be cut in any desired level. We define the distance between two column-pairs based on the header string distance of their $\hat{Q}$ as well as the tuple overlap distance between their $\hat{Q}$ as follows:

$$\text{dist}(\text{cp}_1, cp_2) = \begin{cases} 1, & \text{if } Type(cp_1) \neq Type(cp_2) \\ string_{dist}(cp_1.header, cp_2.header), & \text{if } Type(cp_1) = Type(cp_2) = \text{`numeric'} \\ string_{dist}(cp_1.header, cp_2.header), & \text{if } Type(cp_1) = Type(cp_2) = \text{`date'} \\ H(string_{dist}(cp_1.header, cp_2.header), overlap_{dist}(cp_1, cp_2)), & \text{otherwise} \end{cases} \tag{3.7}$$

Where H is the *Harmonic Mean* between the string distance of the non-query-mapped column headers and the tuple-overlap distance of column-pairs. By using Harmonic Mean, we tend to mitigate the negative impact that a high header dissimilarity or a low tuple overlap introduces in the distance of the column-pairs. The $string_{dist}$ function is defined as character-wise edit distance. The $overlap_{dist}$ is equal to $(1 - overlap_{sim})$, where $overlap_{sim}$ defined as follows:

$$overlap_{sim}(cp_1, cp_2) = \frac{|cp_1 \cap cp_2|}{min(|cp_1|, |cp_2|)}. \tag{3.8}$$

Where $|cp_1 \cap cp_2|$ is the number of tuples that cp_1 and cp_2 have in common, and $|cp_1|$ and $|cp_2|$ are the sizes of column pairs in cp_1 and cp_2 respectively.

In order to compute the distance score between merged clusters, we follow the single link clustering, which uses the shortest distance between any column of one cluster to any column of the other cluster.

Also, we tag each column with its type (i.e. either text, numeric or date) by using a rule-based engine and pattern matching; For doing so we tag each value of a column with its identified type, then we tag the entire column based on the majority vote for the value types. Plus, in any

hierarchy level of our clustering method, if the columns of the clusters being compared are from different types, then they are not merged with each other. In this case, the difference between those two clusters would be 1. If the columns are of type 'numeric' or 'date', we ignore the overlap similarity in this case and only return the string distance between their non-query-mapped column header as their column distance.

3.4 Grouped Column Consolidation

Referring back to Section 1.3, a challenge with column grouping is that there can be multiple values associated with a query entity for an extended column that comes from different table sources and the values can not be reduced to a single value. Recall the motivating example: if we group the second columns in tables 1.2, 1.3 and 1.3, there would be two candidate values "Otawa" and "Ottawa" for the capital attribute associated with "Canada". The goal of *Grouped Column Consolidation* is to consolidate an extended column by selecting a set of canonical values for each query entity q from the values associated with q in the given column-pairs. We formally define this problem as below:

Definition 7. *Grouped Column Consolidation: Given a column-pair group g consisting of a set of column-pairs $(\hat{Q}, C) \in g$, where $\hat{Q}$ has a high $ValueOverlap$ with the query column Q and C describes $\hat{Q}$, output extended column-pair (Q, V) so that for each value-pair $(q, v) \in (Q, V)$:*

- $v = Canonical\_Values(multi\_set)$.
- $multi\_set = \{\{c | (\hat{q}, c) \in (\hat{Q}, C)\ \&\ q = \hat{q}\}\}$.

In Definition 7., The reason of using a *multi-set* is that duplicate values are allowed in multi-sets. This enables us to calculate the frequencies of the elements of a multi-set. In the mentioned definition, the function $Canonical\_Values$ selects the canonical values from a multi-set of values $c \in C$ corresponding to a query value $\hat{q}$. In this function, we use a method similar to fuzzy grouping [35] which clusters the values in the given multi-set based on their edit distance, so that those values with edit distance lower than a threshold are clustered together and for each cluster, it outputs the value with the highest frequency.

3.5 Column Ranking

After selecting related columns to the query and collapsing those columns with the same concepts into a single grouped column, it would be helpful for the user to see a ranked list of extended columns based on the "relatedness" of the extended columns to the query column, so that "more related" columns come before "less related" ones. The challenge here is how to quantify this relatedness and output the ranked list of extended columns. In the *Column Ranking* step, we address the mentioned challenge.

Definition 8. *Column Ranking: Given a set of consolidated column pairs $c \in C$, plus a query column Q, output a ranked list of consolidated columns based on their relatedness to Q.*

Here, we claim that "relatedness" of a column c to a query column q is dependent on two factors: (1) The relatedness between the header of the query column Q and the header of column c which is to be added as an extended column, and (2) the relatedness between the contents of two columns.

We can formalize the first factor by utilizing the concept of *relatedness of a new attribute to an existing attribute in a schema* used in [14] which we call *Header Relatedness*. The Header Relatedness between the query column q and column c can be defined as below [14]:

$$HeaderRel(q, c) = P(c.header|q.header) = \frac{|(c.header, q.header)|}{|q.header|}. \tag{3.9}$$

In equation 3.9, $|(c.header, q.header)|$ is the number of co-occurrences where both attributes are present in one table over the table corpus and $|q.header|$ is the number of tables containing $q.header$ as one of their attributes. In case the header is a set-value C, we average over the $HeaderCons$ of each header value w.r.t. $q.header$ as follows:

$$HeaderRel(q, C) = \frac{\sum_{c \in C} P(c.header|q.header)}{|C|}. \tag{3.10}$$

These co-occurrences can be calculated using Attribute Correlation Statistics Database (ACSDb) [13] which basically records the count of each unique schema over the corpus of web tables. From this information, the co-occurrences between each pair of attributes can be computed, which is in fact the correlation between those attributes in the corpus.

In order to formalize the second factor, we can use the definition of *Header Relatedness* for for a single row as well. For this, we define a measure called *Row Relatedness*:

$$RowRel(q.v_i, c.v_i) = P(c.v_i|q.v_i) = \frac{|(q.v_i, c.v_i)|}{|q.v_i|}. \tag{3.11}$$

Similar to $HeaderCons$, in case of a set-valued $c.v_i$, we average over the $RowCons$ of each value in the set w.r.t. $q.v_i$. The difference between *Header Relatedness* and *Row Relatedness* is in the calculation of co-occurences. In *Header Relatedness*, we use the corpus statistics which deals with table headers (*Header Statistics*). However, in *Row Relatedness* we use another type of statistics which deals with table content (*Content Statistics*). We use Algorithm 1 to build these statistics.

Algorithm 1: CreateStatistics

Input: $TableCorpus$
Output: $HeaderStatistics, ContentStatistics$

```
1  HeaderStatistics ← {}
2  ContentStatistics ← {}
3  for table ∈ TableCorpus do
4      for column ∈ table do
5          header ← column[table.headerRowIndex]
6          HeaderStatistics[header].AssignOrAppend(table.id)
7      for i ← 1 to table.columnCount do
8          for j ← 1 to table.rowCount do
9              cellValue ← table[j][i]
10             ContentStatistics[cellValue].AssignOrAppend(table.row_id)
11 return HeaderStatistics, ContentStatistics
```

In Algorithm 1, we iterate through each column of the corpus tables to create the *Header Statistics* and *Content Statistics*. From lines 3 to 6, we create a mapping between column headers and table_ids and call it *HeaderStatistics*. In the remaining, we create a mapping between each value of columns and their row_ids and call it ContentStatistics

Since Header Relatedness and Row Relatedness have similar formula but use different statistics, then each time we want to compute the relatedness score we can use Algorithm 2 in which we input the header or contents statistics as required.

Algorithm 2: GetRelatednessScore

Input: $Val_1, Val_2, Statistics$
Output: $RelatednessScore$

$set_1 \leftarrow Statistics[Val_1]$
$set_2 \leftarrow Statistics[Val_2]$
$intersect \leftarrow set_1.intersection(set_2)$
$RelatednessScore \leftarrow \frac{len(Extract\_Table\_ID(intersect))}{len(Extract\_Table\_ID(set_1))}$
return $RelatednessScore$

Note that *HeaderRelatedness* and *RowRelatedness* differ from the PMI measure in Section 3.2 in that PMI computes the logarithmic ratio between the probability that query value and its associated extended-column value(s) co-occur in a row and the probability that they co-occure by chance, however, *HeaderRelatedness* and RowRelatedness measure the probability of occurrence of extended-column value(s), given that the query value is present, which is what we would like to measure in *Column Ranking* step.

We can aggregate *Row Relatedness* for each row of query column q and column c to capture the relatedness between the two columns. We call this measure *Content Relatedness* which is defined as follows:

$$ContentRel(q, c) = \frac{\sum_{i=1}^{N} RowRel(q.row_i, c.row_i)}{N}. \tag{3.12}$$

Where N is the number of rows.

Finally, we define the relatedness between the query column q and an additional column c as the combination of Header Relatedness and *Row Relatedness* of the two columns. Any aggregation method can be used to combine the two measures. We use average in our solution, as follows:

$$Relatedness(q, c) = \frac{HeaderRel(q, c) + ContentRel(q, c)}{2}. \tag{3.13}$$

Chapter 4

Experimental Evaluation

In this chapter, we discuss the experimentation and evaluations conducted on our proposed framework for table extension. The goals of the study are:

- To compare our proposed method with two state-of-the-art baselines in terms of the number of unique relevant columns returned, the quality of merging columns with the same concept, and the precision of the extension.
- To measure how different FD-detection methods affect the extension results.
- To measure the effectiveness of the column ranking method.

In the rest of the chapter, we describe the dataset used for the experiments. Then we present our results and compare them with the results from baseline methods.

4.1 Dataset

We use WDC Entity-Level Gold Standard[1] for our experimental evaluation. This dataset contains 233 web tables selected from a random sample of *Web Data Commons Web Tables Corpus* [12], a web table corpus extracted from a web crawl in 2015. For schema matching purposes, each table in this dataset is mapped to a class in the DBpedia ontology,[2] and each table column in the dataset is mapped to a property in DBpedia, if possible. This dataset has also been used in a number of previous works for schema matching [25, 26, 36]. An overview of the table mapping is presented in Table 4.1. A detailed statistics of this dataset mapping is presented in Appendix A.

[1] http://webdatacommons.org/webtables/goldstandard.html
[2] http://mappings.dbpedia.org/server/ontology/classes/

Superclass	Classes	#Tables
Place	Country, City, Mountain, Airport, etc.	102
Agent	Company, Political Party, Scientist, Baseball Player, etc.	71
Work	Film, Video Game, Book, Newspaper, etc.	46
Species	Animal, Bird, Plant, etc.	14
	Total	**233**

Table 4.1: Dataset Overview

#	Club	Country	Points
1	Barcelona	ESP	2037
2	Real Madrid	ESP	2008
3	Bayern München	GER	1973
4	Paris Saint Germain	FRA	1914
5	Atlético Madrid	ESP	1880
6	Juventus	ITA	1863
7	Manchester City	ENG	1862
8	Arsenal	ENG	1853
9	FC Porto	POR	1850
10	Manchester United	ENG	1804

Figure 4.1: Sample web table [37]

For each table in our dataset, table content as well as the key-column and the header row of the table are specified by properties named *relation*, *Key Column Index* and *Header Row Index*, respectively. Values from these three properties are mainly used for a table in a table extension. The dataset also stores additional metadata for each table such as *Source Page URL* and *Text Before and after Table*, which are not used in our solution, but are used in other methods such as InfoGather [4]. A sample table in our dataset, stored in a Json file, is shown in Figure 4.2, which describes the table in Figure 4.1.

```
{
  "relation":[
    ["#","1","2","3","4","5","6","7","8","9","10"],
    ["Club","Barcelona","Real Madrid","Bayern München","Paris Saint Germain","Atlético Madrid","Juventus",
    "Manchester City","Arsenal","FC Porto","Manchester United"],
    ["Country","ESP","ESP","GER","FRE","ESP","ITA","ENG","ENG","POR","ENG"],
    ["Points","2037","2008","1973","1914","1880","1863","1862","1853","1850","1804"]
   ]
  "pageTitle": "FootballDatabase - Club Rankings and Statistics",
  "title": "",
  "url": "http://footballdatabase.com/index.php?page\u003dplayer\u0026Id\u003d660",
  "hasHeader": true,
  "headerPosition": "FIRST_ROW",
  "tableType": "RELATION",
  "tableNum": 0,
  "s3Link":
  "common-crawl/crawl-data/CC-MAIN-2015-32/segments/1438042981460.12/warc/CC-MAIN-20150728002301-00000-ip-10-
  236-191-2.ec2.internal.warc.gz",
  "recordEndOffset": 99246001,
  "recordOffset": 99230046,
  "tableOrientation": "HORIZONTAL",
  "TableContextTimeStampBeforeTable": "{10283\u003dOn Wednesday, December 6, 2006 Islanders General Manager
  Garth Snow attended the Fifth Annual John Theissen Holiday Fundraiser.}",
  "TableContextTimeStampAfterTable": "{37811\u003dIn 2005, Slovakian champion FC Artmedia upset 39-time
  Scottish league champion Celtic 5-0 in their European Champions League second-round qualifying match.}",
  "lastModified": "Sat, 19 Jun 2010 15:14:57 GMT",
  "textBeforeTable": "Chelsea Ronnie MacDonald Bayern München Peter P. Juventus Mitsurinho Real Madrid Jan
  S0L0 Barcelona Globovision Football",
  "textAfterTable": "Full World Ranking Match Centre Argentina Primera 2015 26 July 2015 Vélez Sarsfield 0
  - 2 Olimpo Brazil Serie A 2015 26 July 2015 Vasco da Gama 1 - 4 Palmeiras Mexico Liga",
  "hasKeyColumn": true,
  "keyColumnIndex": 1,
  "headerRowIndex": 0
}
```


Figure 4.2: Dataset Sample Json File associated to Figure 4.1

4.2 Evaluating Table Extension Compared to Baselines

Based on the related work discussed in Section 2.1, *Mannheim Search Join* is the only method which is capable of performing a table extension and is not targeted for a specific domain. Hence we use this method as one of the baselines for our evaluation. In order to perform a table extension, we can also apply *Attribute Discovery* [4] to methods that try to fill an attribute column. From those methods, we pick *InfoGather* as another baseline. We evaluate and compare the following table extension methods:

1. Proposed Solution: We have implemented the proposed solution explained in Chapter 3, with *Error Threshold* as the FD-detection method.

2. InfoGather [4]: InfoGather does not directly support table extension, however, we can perform a table extension by combining *Attribute Discovery* and *Augmentation By Attribute name*. For this reason, we have implemented and combined the two operations in the following way: we first perform *Attribute Discovery* for an input query column which outputs a set of attribute-name groups, with each group representing a single concept. For example, sup-

pose the query column is consisted of a number of camera models. A sample attribute-name group may consist of a set such as $\{brand, manufacturer, maker, vendor\}$ where all of its elements refer to a single concept which is the manufacturer of the camera model. After this step, we execute *Augmentation By Attribute name* for each attribute name in each group. We consolidate the output values for each query row in each output header group as discussed in Section 2.1. We use the same consolidation method in this baseline as the one we used in the *Grouped Column Consolidation* step in our proposed framework so that the comparisons between the methods are fair.

3. Mannheim SearchJoin [3]: We have also implemented Mannheim SearchJoin. However, since this work has similar steps for *Column Grouping* and *Grouped Column Consolidation*, for consistency we have used the same methods that we used in our solution so that the methods can be compared fairly.

We evaluate the quality of our table extension, compared to the baselines discussed above, under two settings:

1. **Pre-selected queries**: We choose 7 queries from the topics used in our dataset that cover a wide range of distribution of tables about those topics. The name of these topics, ordered by the number of tables about them are as follows: Country, Video Game, Film, Company, Cities, Language, Currency. In order to pick the query values for each topic, we select a random column from each topic in our dataset, and from that column, we pick 10 random entities.

2. **Random Queries**: We randomly select 5% of columns in our table corpus, which amounts to 57 queries, and use each selected column as a query column for table extension evaluation.

We use DBpedia [5] as the ground truth in order to cross-check the values in an extended table. Since each DBpedia triple is in the format of $\langle subject, predicate, object\rangle$, we perform the cross-check in the way that for each value v associated with a query value q in an extended column c, we match triples $\langle q, predicate, v\rangle$ and $\langle v, predicate, q\rangle$ with DBpedia triples and gather all matched $predicates$. Each value v is tagged with the most frequent matched $predicate$. In the case that v is a set-value, it is tagged with the most frequent matched $predicate$ in the set, and in case of ties,

v is tagged with all most frequent matched predicates. Finally, each extended column c is tagged with the most frequent tag associated to the values in c, and in case of ties, c is tagged with all most frequent tags associated to values in c. We consider each value v in an extended column c as 'correct' if the DBpedia tag associated with v is the same as the DBpedia tag of column c.

For example, assume the query column is consisted of a number of countries, and a table extension is performed as shown in Figure 4.3. In order to cross-check the first extended column over DBpedia, we issue $\langle Canada, predicate, Ottawa\rangle$ and $\langle Ottawa, predicate, Canada\rangle$ for its first row, $\langle Spain, predicate, Madrid\rangle$ and $\langle Madrid, predicate, Spain\rangle$ for the second row, ..., and $\langle Egypt, predicate, Africa\rangle$ and $\langle Africa, predicate, Egypt\rangle$ for the last row. The value tag for the first, second and fourth column value is $Capital$. For the third value no predicate can be found in DBpedia, which may suggest that the value is incorrect. The fifth value is tagged as $Continent$. Since $Capital$ is the majority among the value tags, the column is tagged as $Capital$. We consider $Ottawa$, $Madrid$ and $London$ as 'correct' since their value tags are the same as the column tag, while $Australia$ and $Egypt$ are considered 'incorrect' because their tags differ from the column tag.

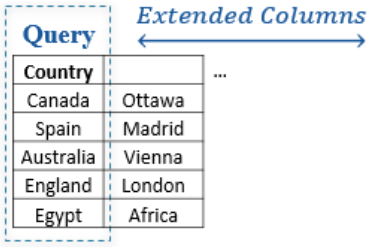


Figure 4.3: Sample extension

We have executed the algorithms on the gold standard dataset described earlier and compared the results based on the following criteria:

- **Number of Unique Relevant Columns:** We report the number of unique relevant columns extended by each method. We expect that our proposed method returns more unique relevant columns than the baseline methods since our method is utilizing more useful columns from the table corpus than the other baselines. We consider an extended column as 'relevant' if the precision of the contents of that column is higher than a threshold. By raising this threshold,

the number of relevant columns returned by each method becomes smaller, and by lowering the threshold this number grows. We set this threshold to 20% in our experiments. We compute this measure as below:

$$Column\ Content\ Precision = \frac{\#correct\ values}{\#values}. \tag{4.1}$$

- **Grouping Factor:** For each method, we measure the quality of combining extended columns with the same concept and outputting unique columns as follows:

$$Grouping\ Factor = \frac{\#unique\ relevant\ columns}{\#relevant\ columns}. \tag{4.2}$$

  The expectation is to have the highest possible grouping factor in an extension, hence the least possible number of duplicates in extended columns. To identify duplicate columns, we check whether the columns have the same DBpedia tag or not. However, due to DBpedia's triplet structure, it is sometimes difficult to determine whether two columns with the same tag represent the same concept. For example, in an extended table with countries as query values, DBpedia fails to detect that columns 'lakes' and 'mountains' are not duplicates, since they are both tagged as 'location'. To avoid such cases, we perform an additional step where we check the overlap between columns with the same tag. This is measured as the number of rows where the two columns have the same values for a corresponding query value. If this overlap is higher than a threshold, we mark those columns as duplicates. We check the overlap of only those values associated with the same query value. For set-valued columns, two column values match if the intersection between the two sets is not empty, since this indicates that the two sets are about the same concept.

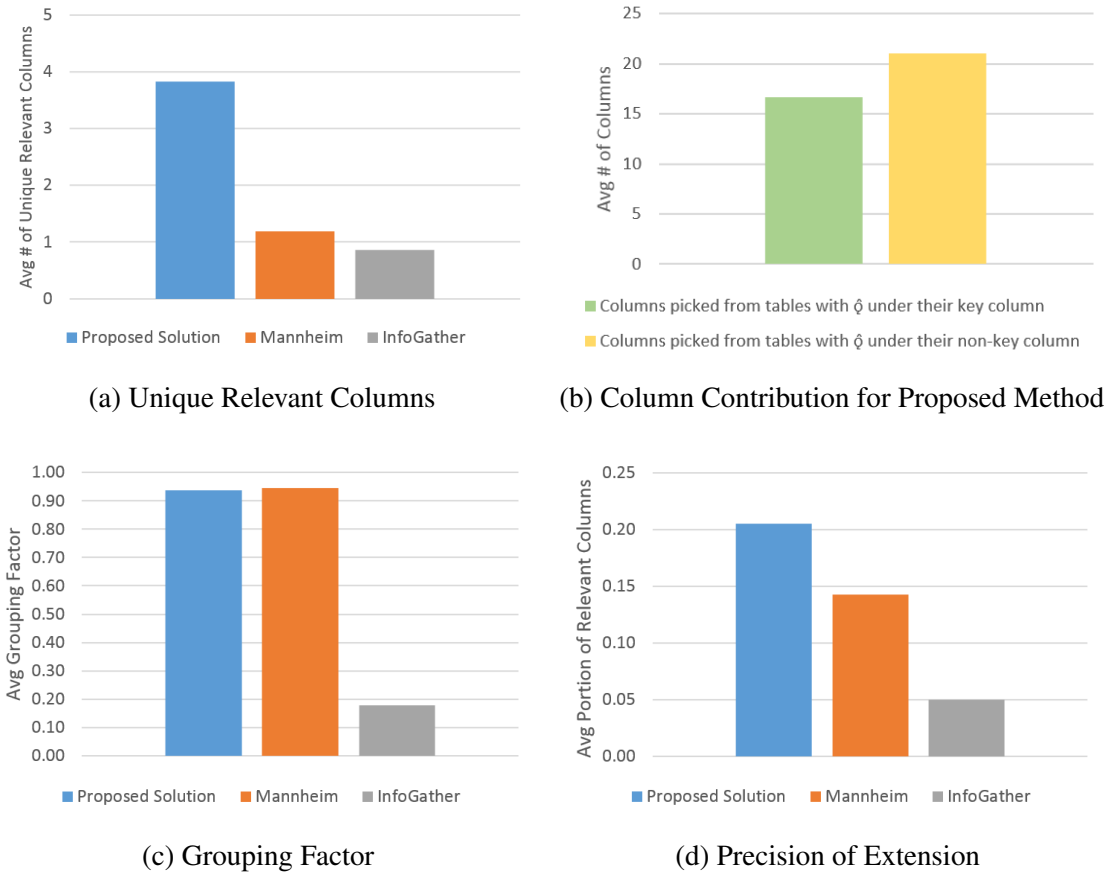
- **Precision of Extension:** We also measure the precision of an extension as follows:

$$Precision\ of\ Extension = \frac{\#unique\ relevant\ columns}{\#all\ extended\ columns}. \tag{4.3}$$

  Based on the equation above, higher precision of an extension results in larger portion of extended columns that are relevant to the query column, which means that fewer irrelevant columns would be returned.

Figure 4.4 reports the evaluation results averaged over 57 randomly selected queries. As shown in Figure 4.4a, the number of unique relevant columns extended by our proposed solution is on

average 3 times higher than that of Mannheim and InfoGather. This is mainly due to the fact that both Mannheim and InfoGather only use in their extensions those tables in which the query column matches the key column, referred to $\hat{Q}$ in Chapter 3.2. However, the proposed solution pulls columns from tables where $\hat{Q}$ is a key column, as well as from tables where $\hat{Q}$ is a non-key column. As depicted in Figure 4.4b for our table extension algorithm, a random query on average retrieves more columns from tables where the query matches a non-key column of a table.



(a) Unique Relevant Columns

(b) Column Contribution for Proposed Method

(c) Grouping Factor

(d) Precision of Extension

Figure 4.4: Evaluations on Random Queries

InfoGather consistently retrieves the smallest number of relevant columns. This is because the extended columns are populated based on a given attribute name (within the Augmentation By Attribute Name step), and attribute names may not be present in all columns or if present, may not be informative. Hence, the extended columns cannot be populated to the full extent.

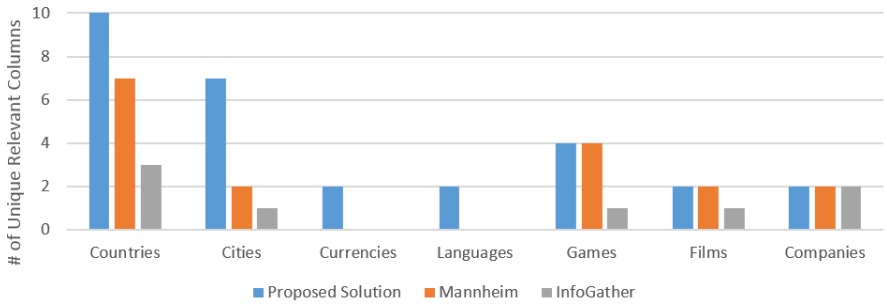
Our proposed method retrieves more relevant columns than our competitors, and a large portion of those relevant columns are unique and not redundant. This is shown in Fig 4.3c where

the grouping factor for our method is in the same scale as Mannheim. InfoGather has the lowest grouping factor because an extended column returned from this method does not necessarily represent a single attribute and may share some values with other columns, hence the column can be detected as a duplicate column. The reason of this limitation is in the *Attribute Discovery* step where a number of attribute-names coming from the same web table which are not necessarily synonyms, are placed in the same attribute-name group. This comes from a bias in the *Holistic Matching* method towards EABs coming from a same source table as discussed in Section 2.1.

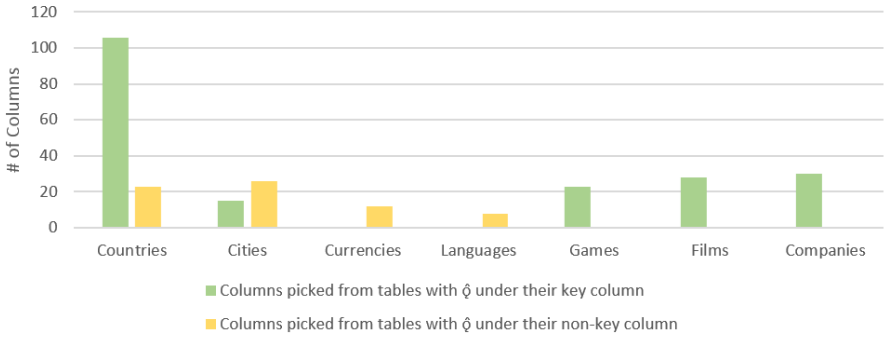
Figure 4.4d shows that the precision of extending a table using our method is higher than that of both Mannheim and InfoGather, meaning that a larger portion of the retrieved columns are relevant.

Figure 4.5 shows the evaluations of pre-selected queries. We observed similar results for these queries as well. It is noteworthy that when the query is about currencies or languages, there are no tables in the dataset with a key column matching our queries. Hence, Mannheim and InfoGather cannot extend those queries. However, our proposed solution leverages the columns in those tables with $\hat{Q}$ among their non-key columns. For example, in the case of the language query, there exists a table with a key column about countries, containing a column about the languages spoken in those countries. In this example, there is a correlation between the value-pairs $(country, language)$ in our corpus tables since their pairwise value occurrences are higher than expected by pure coincidence. Hence the country column is picked up by the proposed method as an extension of the language query. In another case, with a list of currency names as query, there exists a table about countries, which lists both the currency name and currency code for each country. Clearly, the query does not match the key column of the table, but there is a functional dependency between 'Currency Name' and 'Currency Code' in the form of $Currency\ Name \rightarrow Currency\ Code$, which suggests that the table is not in BCNF. This table can be decomposed into smaller BCNF tables, such that one of the BCNF tables contains 'Currency Name' as its key column and 'Currency Code' as its non-key column. Our proposed method picks up 'Currency Code' as a property or extension of 'Currency Name'.

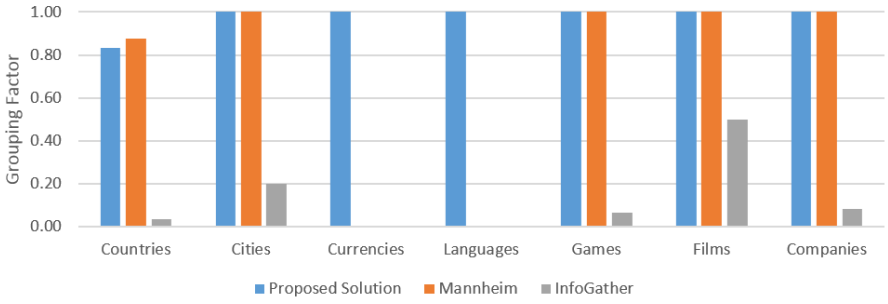
On query columns games, films and companies, both the proposed method and Mannheim have the same results since both methods pick the same source columns from tables where the query matches the key column of the table. The other steps in both algorithms, namely column grouping



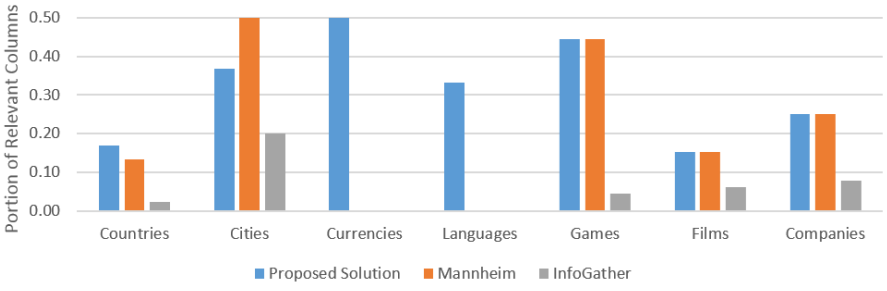
(a) Unique Relevant Column



(b) Column Contribution for Proposed Method



(c) Grouping Factor



(d) Precision of Extension

Figure 4.5: Evaluations on Pre-selected Queries

and value selection, also function in the same manner.

4.3 Evaluating FD-Detection Methods

In this section, we evaluate the quality of functional dependency detection methods discussed in Section 3.2. The goal of the evaluation is to measure how different FD-detection methods affect the extension result. We use 3 methods in this evaluation: *Error Threshold (ET)*, *Violation Detectability (VD)*, and *Violation Detectability with Column Grouping (VDCG)*

We compare the quality of table extension using each of the aforementioned FD-detection methods. The queries were selected randomly from the set of columns in the table corpus. However, since in 75% of the queries, all 3 methods yield the same result for a query, we set a condition to select a query only if it yields different results for at least 2 methods.

Table 4.2 shows the evaluation of different FD-detection methods. We notice that *VDCG* extends the query table with more relevant columns than *VD*, but less relevant columns than *ET*. This is due to the fact that some of those approximate FDs that are accepted by *ET* have no duplicate values on their right-hand-sides and are rejected by *VD*. However, the attributes of some of those rejected FDs occur in other tables, and those FDs are recovered after a column grouping in VDCG. The same trend can be seen in the average number of source columns that are selected from tables matching a non-key column with the query column. As shown in Table 4.2, *ET* selects 35.45 columns on average, *VD* rejects 22.05 columns on average from those returned by *ET*, and *VDCG* recovers 8.3 of those rejected columns.

	ET	**VD**	**VDCG**
Avg # of Unique Relevant Columns	7.55	3.45	4.90
Avg # of Source Columns Picked from Tables with $\hat{Q}$ under their Non-Key Column	35.45	13.40	21.70
Avg Grouping Factor	1.00	1.00	1.00
Avg Table Size	17.05	7.65	9.75
Precision of Extension	0.44	0.45	0.50

Table 4.2: Comparison between FD-Detection methods

Despite the fact that table extension using *ET* augments the query table with more relevant columns (compared to other FD-detection methods), we cannot claim that *ET* is a better method overall. Selecting columns that are thought to be functionally dependent on the query column without additional evidence from other tables may result in picking more irrelevant source columns and rejecting functional dependencies with no duplicates in the *rhs* of the FD may result in losing relevant source columns. *VDCG* is a trade-off that compromises between these two extremes. This is also shown in Table 4.2, where the average table size of a table extension using *VDCG* is smaller than the average table size of an extension using *ET* and larger than that of *VD* but its average precision of extension is the highest among these methods.

4.4 Column Ranking Analysis

As discussed in Section 3.5, users prefer to see extended columns that are more related to the query column first. Hence the proposed method returns a ranked list of extended columns ordered by their relevance to the query. In this section we evaluate the *Column Ranking* step in our proposed solution. We evaluate the column ranking step using each of the pre-selected queries in 3 settings: The relatedness between an extended column and the query column based on their (1) *Header Relatedness*, (2) *Row Relatedness*, and (3) the combination of the two, which were all discussed in Section 3.5. The goal of this evaluation is to measure how effective the column ranking method is.

To assess the effectiveness of *Column Ranking*, we calculate the *R-precision* (a.k.a. *the precision at R*) [38] of the relevant columns for each ranking setting as below:

$$R\text{-}Precision = \frac{\#relevant\ columns\ in\ the\ top\text{-}R\ extended\ columns}{R}. \tag{4.4}$$

For each query, R is chosen as the total number of unique relevant columns for that query. In this experiment, since R-precision is measured for each query, we average this quantity over queries in our query set to compare the results of the ranking settings.

Table 4.3 shows the R-precision for each of the pre-selected queries in each column ranking setting. Based on the results, when the relatedness score is based on Row Relatedness, the average R-precision is higher than the Header Relatedness setting. This is partly due to the fact that some of the columns in the dataset lack header names or contain header names that are not informative for the column. Plus, by using row relatedness, more evidence can be found in the table corpus

that shows whether the values of 2 columns co-occur with each other or not. However, based on the evaluation results, the best ranking setting is when header relatedness and row relatedness are combined and used as a single relatedness measure which leverages the benefits of both measures.

4.5 Summary of Results

Our evaluation reveals that:

- The number of unique relevant columns extended by our proposed solution is on average 3 times higher than that of two state-of-the-art baselines.
- The precision of extending a table using our method is higher than that of both baselines, meaning that fewer irrelevant columns are retrieved.
- The *Error Threshold* method for Functional Dependency detection results in picking more irrelevant source columns and the *Violation Detectability* method results in losing relevant source columns. *Violation Detectability with Column Grouping* is a trade-off that compromises between these two extremes.
- In the *Column Ranking* step, when the relatedness score is based on *Row Relatedness*, the average R-precision is higher than the *Header Relatedness* setting. The best ranking setting is when these two measures are combined and used as a single relatedness measure, which leverages the benefits of both measures.

Relatedness Score = HeaderRelatedness				
Query	**#Rel @Top #TotalRel**	**#TotalRel**	**TotalExtended**	**R-Precision**
Games	3	4	9	0.75
Films	0	2	13	0.00
Companies	0	2	8	0.00
Currencies	0	2	4	0.00
Languages	2	2	6	1.00
Cities	3	7	19	0.43
Countries	5	10	59	0.5
			Average	**0.38**

(a) Column ranking evaluation when *Relatedness Score* is solely based on *HeaderRelatedness*

Relatedness score = RowRelatedness				
Query	**#Rel @Top #TotalRel**	**#TotalRel**	**TotalExtended**	**R-Precision**
Games	3	4	9	0.75
Films	1	2	13	0.00
Companies	1	2	8	0.00
Currencies	1	2	4	0.00
Languages	2	2	6	1.00
Cities	3	7	19	0.43
Countries	4	10	59	0.5
			Average	**0.58**

(b) Column ranking evaluation when *Relatedness Score* is solely based on *RowRelatedness*

Relatedness Score = Avg(HeaderRelatedness, RowRelatedness)				
Query	**#Rel @Top #TotalRel**	**#TotalRel**	**TotalExtended**	**R-Precision**
Games	3	4	9	0.75
Films	1	2	13	0.00
Companies	1	2	8	0.00
Currencies	1	2	4	0.00
Languages	2	2	6	1.00
Cities	4	7	19	0.43
Countries	5	10	59	0.5
			Average	**0.62**

(c) Column ranking evaluation when *Relatedness Score* is based on the average of *HeaderRelatedness* and *RowRelatedness*

Table 4.3: Column Ranking Evaluation

Chapter 5

Conclusion

In this thesis, we propose a framework of table extension, where given a query table and a corpus of web tables, the goal is to find a ranked list of additional columns from the table corpus that describe the entities of the query table. The framework consists of 5 steps: (1) *Table Search*, which extracts tables from the corpus that are potential candidates for having suitable source columns to be used in an extended column, (2) *Column Selection*, which selects suitable columns for performing a table extension from the candidate tables returned by the previous step, (3) *Column Grouping*, which merges columns that represent the same concept but are coming from different tables into separate clusters, (4) *Grouped Column Consolidation*, which consolidates each column-cluster from the previous step into a single extended column by selecting a single value or a set of values for each column-cluster and query value, and (5) *Column Ranking*, which ranks the extended columns based on their relatedness to the query column.

In the *Column Selection* step, we use the notion of *functional Dependency* in order to capture the relatedness between suitable columns and the query column. To detect *Functional Dependencies*, we present 3 methods: (1) *Error Threshold*, which is based on the number of violations in a table for an FD to be satisfied, (2) *Violation Detectability*, which tries to detect positive evidence in a table for an FD to hold over a schema, and (3) *Violation Detectability with Column Grouping*, which tries to find positive evidence from other tables for an FD to hold over a schema. We study the effects of each method on table extension results. The study shows that *Error Threshold* results in picking more irrelevant source columns and *Violation Detectability* results in losing relevant source columns. *Violation Detectability with Column Grouping* is a trade-off that compromises between these two extremes.

We introduce methods for ranking an extended column based on the *Header Relatedness* and *Row Relatedness* between the query column and the extended column. We show that when the relatedness score is based on *Row Relatedness*, the average R-precision is higher than the *Header Relatedness* setting. The best ranking setting is when these two measures are combined together, which leverages the benefits of both measures.

Our evaluation demonstrates that the number of unique relevant columns extended by our proposed solution is on average 3 times higher than that of two state-of-the-art baselines. Plus, the precision of extending a table using our method is higher than that of both baselines, meaning that fewer irrelevant columns are retrieved.

As a possible direction for future research, one can improve on the matching method between the values of table columns. Currently, we are using a character-based matching. However, in web tables, there are multiple cases where this method may not work for matching two values. For example, matching a word with its abbreviated form may not be done via this method, since the character-wise similarity between the two words may not be high enough. As another example, character-based matching may not work for synonyms where the words do not have enough shared characters.

A limitation of our proposed framework in detecting Functional Dependencies is that, after applying any of the FD-detection methods, there are still some accidental FDs that are accepted as *true* FDs. As another area for future research, one can improve the FD-detection method in order to identify and reject accidental FDs with more accuracy. Also, one can improve on the time complexity of the proposed framework by proposing a more efficient column grouping method than the hierarchical clustering in which the time complexity is $O(n^3)$.

References

[1] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 1090–1101, Aug. 2009, ISSN: 2150-8097. DOI: 10.14778/1687627.1687750. [Online]. Available: https://doi.org/10.14778/1687627.1687750. 1, 11, 14

[2] J. Eberius, M. Thiele, K. Braunschweig, and W. Lehner, "Top-k entity augmentation using consistent set covering," in *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '15, La Jolla, California: ACM, 2015, 8:1–8:12, ISBN: 978-1-4503-3709-0. DOI: 10.1145/2791347.2791353. [Online]. Available: http://doi.acm.org/10.1145/2791347.2791353. 1, 12

[3] O. Lehmberg, D. Ritze, P. Ristoski, R. Meusel, H. Paulheim, and C. Bizer, "The mannheim search join engine," *Web Semant.*, vol. 35, no. P3, pp. 159–166, Dec. 2015, ISSN: 1570-8268. DOI: 10.1016/j.websem.2015.05.001. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2015.05.001. 1, 10, 11, 15, 33

[4] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri, "Infogather: Entity augmentation and attribute discovery by holistic matching with web tables," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12, Scottsdale, Arizona, USA: ACM, 2012, pp. 97–108, ISBN: 978-1-4503-1247-9. DOI: 10.1145/2213836.2213848. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213848. 1, 8, 11, 12, 15, 3

[5] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," English, *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015. DOI: 10.3233/SW-140134. [Online]. Available: https://madoc.bib.uni-mannheim.de/37476/. 7, 33

[6] M. Zhang and K. Chakrabarti, "Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13, New York, New York, USA: ACM, 2013, pp. 145–156, ISBN: 978-1-4503-2037-5. DOI: 10.1145/2463676.2465276. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465276. 10

[7] C. S. Bhagavatula, T. Noraset, and D. Downey, "Methods for exploring and mining tables on wikipedia," in *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics*, ser. IDEA '13, Chicago, Illinois: ACM, 2013, pp. 18–26, ISBN: 978-1-4503-2329-1. DOI: 10.1145/2501511.2501516. [Online]. Available: http://doi.acm.org/10.1145/2501511.2501516. 11

[8] G. Penn, Jianying Hu, Hengbin Luo, and R. McDonald, "Flexible web document analysis for delivery to narrow-bandwidth devices," in *Proceedings of Sixth International Conference on Document Analysis and Recognition*, Sep. 2001, pp. 1074–1078. DOI: 10.1109/ICDAR.2001.953951. 13

[9] H.-H. Chen, S.-C. Tsai, and J.-H. Tsai, "Mining tables from large scale html texts," in *Proceedings of the 18th Conference on Computational Linguistics - Volume 1*, ser. COLING '00, Saarbrücken, Germany: Association for Computational Linguistics, 2000, pp. 166–172, ISBN: 1-55860-717-X. DOI: 10.3115/990820.990845. [Online]. Available: https://doi.org/10.3115/990820.990845. 13

[10] Y. Wang and J. Hu, "A machine learning based approach for table detection on the web," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02, Honolulu, Hawaii, USA: ACM, 2002, pp. 242–250, ISBN: 1-58113-449-5. DOI: 10.1145/511446.511478. [Online]. Available: http://doi.acm.org/10.1145/511446.511478. 13

[11] M. J. Cafarella and E. Wu, "Uncovering the relational web," in *Proceedings of the 11th International Workshop on Web and Databases*, ser. WebDB '08, 2008. 13

[12] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer, "A large public corpus of web tables containing time and context metadata," in *Proceedings of the 25th International Conference Companion on World Wide Web*, ser. WWW '16 Companion, Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 75–76, ISBN: 978-1-4503-4144-8. DOI: 10.1145/2872518.2889386. [Online]. Available: https://doi.org/10.1145/2872518.2889386. 13, 30

[13] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "Webtables: Exploring the power of tables on the web," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 538–549, Aug. 2008, ISSN: 2150-8097. DOI: 10.14778/1453856.1453916. [Online]. Available: http://dx.doi.org/10.14778/1453856.1453916. 14, 27

[14] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12, Scottsdale, Arizona, USA: ACM, 2012, pp. 817–828, ISBN: 978-1-4503-1247-9. DOI: 10.1145/2213836.2213962. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213962. 14, 27

[15] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu, "Recovering semantics of tables on the web," *Proc. VLDB Endow.*, vol. 4, no. 9, pp. 528–538, Jun. 2011, ISSN: 2150-8097. DOI: 10.14778/2002938.2002939. [Online]. Available: http://dx.doi.org/10.14778/2002938.2002939. 14, 16

[16] G. Limaye, S. Sarawagi, and S. Chakrabarti, "Annotating and searching web tables using entities, types and relationships," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1338–1347, Sep. 2010, ISSN: 2150-8097. DOI: 10.14778/1920841.1921005. [Online]. Available: http://dx.doi.org/10.14778/1920841.1921005. 14, 15

[17] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later.," *PVLDB*, vol. 4, no. 11, pp. 695–701, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/pvldb/pvldb4.html#BernsteinMR11. 15

[18] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, Dec. 2001, ISSN: 1066-8888. DOI: 10.1007/s007780100057. [Online]. Available: http://dx.doi.org/10.1007/s007780100057. 15

[19] Z. Bellahsene, A. Bonifati, and E. Rahm, *Schema Matching and Mapping*, 1st. Springer Publishing Company, Incorporated, 2011, ISBN: 9783642165177. 15

[20] J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy, "Corpus-based schema matching," in *Proceedings of the 21st International Conference on Data Engineering*, ser. ICDE '05, Washington, DC, USA: IEEE Computer Society, 2005, pp. 57–68, ISBN: 0-7695-2285-8. DOI: 10.1109/ICDE.2005.39. [Online]. Available: https://doi.org/10.1109/ICDE.2005.39. 15

[21] G. Weikum and M. Theobald, "From information to knowledge: Harvesting entities and relationships from web sources," in *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '10, Indianapolis, Indiana, USA: ACM, 2010, pp. 65–76, ISBN: 978-1-4503-0033-9. DOI: 10.1145/1807085.1807097. [Online]. Available: http://doi.acm.org/10.1145/1807085.1807097. 15

[22] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14, New York, New York, USA: ACM, 2014, pp. 601–610, ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623623. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623623. 15

[23] X. Zhang, Y. Chen, J. Chen, X. Du, and L. Zou, "Mapping entity-attribute web tables to web-scale knowledge bases," in *Database Systems for Advanced Applications*, W. Meng, L. Feng, S. Bressan, W. Winiwarter, and W. Song, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 108–122, ISBN: 978-3-642-37450-0. 15

[24] Y. A. Sekhavat, F. di Paolo, D. Barbosa, and P. Merialdo, "Knowledge base augmentation using tabular data," in *7th Workshop on Linked Data on the Web*, CEUR-WS.org, 2014. [Online]. Available: http://ceur-ws.org/Vol-1184/. 15

[25] D. Ritze, O. Lehmberg, and C. Bizer, "Matching html tables to dbpedia," in *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '15, Larnaca, Cyprus: ACM, 2015, 10:1–10:6, ISBN: 978-1-4503-3293-4. DOI: 10.1145/2797115.2797118. [Online]. Available: http://doi.acm.org/10.1145/2797115.2797118. 15, 30

[26] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1502–1513, Aug. 2017, ISSN: 2150-8097. DOI: 10.14778/3137628.3137657. [Online]. Available: https://doi.org/10.14778/3137628.3137657. 16, 30

[27] X. Ling, A. Halevy, F. Wu, and C. Yu, "Synthesizing union tables from the web," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI '13, Beijing, China: AAAI Press, 2013, pp. 2677–2683, ISBN: 978-1-57735-633-2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2540128.2540514. 16

[28] R. Gupta and S. Sarawagi, "Answering table augmentation queries from unstructured lists on the web," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 289–300, Aug. 2009, ISSN: 2150-8097. DOI: 10.14778/1687627.1687661. [Online]. Available: https://doi.org/10.14778/1687627.1687661. 16

[29] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 908–919, Jun. 2012, ISSN: 2150-8097. DOI: 10.14778/2336664.2336665. [Online]. Available: http://dx.doi.org/10.14778/2336664.2336665. 16

[30] M. Kifer, A. Bernstein, and P. M. Lewis, "Database design with the relational normalization theory," in *Database Systems: An Application Oriented Approach, Complete Version (2Nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005, pp. 193–250, ISBN: 0321268458. 19

[31] Y. Wang and Y. He, "Synthesizing mapping relationships using table corpus," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17, Chicago, Illinois, USA: ACM, 2017, pp. 1117–1132, ISBN: 978-1-4503-4197-4. DOI: 10.1145/3035918.3064010. [Online]. Available: http://doi.acm.org/10.1145/3035918.3064010. 20

[32] J. Kivinen and H. Mannila, "Approximate dependency inference from relations," in *Database Theory — ICDT '92*, J. Biskup and R. Hull, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 86–98, ISBN: 978-3-540-47360-2. 20

[33] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Comput. Linguist.*, vol. 16, no. 1, pp. 22–29, Mar. 1990, ISSN: 0891-2017. [Online]. Available: http://dl.acm.org/citation.cfm?id=89086.89095. 23, 24

[34] L. Rokach and O. Maimon, "Clustering methods," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Boston, MA: Springer US, 2005, pp. 321–352, ISBN: 978-0-387-25465-4. DOI: 10.1007/0-387-25465-X_15. [Online]. Available: https://doi.org/10.1007/0-387-25465-X_15. 25

[35] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and efficient fuzzy match for online data cleaning," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03, San Diego, California: ACM, 2003, pp. 313–324, ISBN: 1-58113-634-X. DOI: 10.1145/872757.872796. [Online]. Available: http://doi.acm.org/10.1145/872757.872796. 26

[36] A. L. Gentile, P. Ristoski, S. Eckel, D. Ritze, and H. Paulheim, "Entity matching on web tables: A table embeddings approach for blocking," English, in *Advances in Database Technology - EDBT 2017 : 20th International Conference on Extending Database Technology, Venice, Italy, March 21?24, 2017, Proceedings*, Online-Ressource, Konstanz: OpenProceedings, 2017, pp. 510–513. DOI: 10.5441/002/edbt.2017.57. [Online]. Available: http://ub-madoc.bib.uni-mannheim.de/41887/. 30

[37] *Wdc dataset instructions*, http://webdatacommons.org/webtables/2015/downloadInstructions.html, Accessed: 2019-04-01. 31

[38] N. Craswell, "R-precision," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 2453–2453, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_486. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_486. 40

Appendix A

Dataset Detailed Statistics

Table Labels	#Tables	Level 2 DBPedia Class	Level 1 DBPedia Class
Company	17	Organization	Agent
PoliticalParty	13	Organization	Agent
Scientist	9	Person	Agent
Monarch	9	Person	Agent
BaseballPlayer	6	Person	Agent
RadioStation	5	Organization	Agent
Saint	3	Person	Agent
Wrestler	2	Person	Agent
Person	2	Person	Agent
Airline	2	Organization	Agent
FictionalCharacter	1	FictionalCharacter	Agent
University	1	Organization	Agent
GolfPlayer	1	Person	Agent
Country	40	Populated Place	Place
Mountain	13	Natural Place	Place
Lake	12	Natural Place	Place
Hospital	9	Architectural Structure	Place
City	9	Populated Place	Place

Airport	7	Architectural Structure	Place
Museum	7	Architectural Structure	Place
Building	3	Architectural Structure	Place
AdministrativeRegion	1	Populated Place	Place
Hotel	1	Architectural Structure	Place
Animal	6	Eukaryote	Species
Bird	5	Eukaryote	Species
Plant	3	Eukaryote	Species
VideoGame	19	Software	Work
Film	17	Film	Work
Newspaper	4	Written Work	Work
Book	2	Written Work	Work
AcademicJournal	2	Written Work	Work
Work	1		Work
TelevisionShow	1	TelevisionShow	Work

Table A.1: Dataset Detailed Overview

Row Labels	**Sum**
Country	
rdf-schema#label	40
capital	17
populationTotal	14
currency	12
currencyCode	8
language	6
grossDomesticProduct	6
governmentType	6

formerName	5
foundingYear	4
PopulatedPlace/area	4
iso31661Code	4
subdivision	3
capitalCoordinates	2
circle	2
frenchName	1
PopulatedPlace/populationDensity	1
GeopoliticalOrganisation/populationDensity	1
giniCoefficient	1
diocese	1
lifeExpectancy	1
topLevelDomain	1
officialLanguage	1
fipsCode	1
perCapitaIncome	1
code	1
PopulatedPlace/areaTotal	1
income	1
creationYear	1
infantMortality	1
region	1
catholicPercentage	1
timeZone	1
isoCode	1
date	1
continent	1

Country Total	154
VideoGame	
rdf-schema#label	19
computingPlatform	11
releaseDate	10
developer	9
genre	9
publisher	8
year	6
category	2
Software/fileSize	1
manufacturer	1
usk	1
cost	1
numberOfPlayers	1
number	1
VideoGame Total	80
Film	
rdf-schema#label	17
releaseDate	17
director	17
country	1
writer	1
duration	1
rating	1
Film Total	55
Company	
rdf-schema#label	17

industry	17
sales	4
assets	4
revenue	3
symbol	3
collectionSize	2
country	2
founder	1
headquarter	1
Company Total	54
City	
populationTotal	9
country	9
rdf-schema#label	9
mayor	9
populationMetro	9
City Total	45
Lake	
rdf-schema#label	12
location	6
areaTotal	5
elevation	4
country	2
depth	1
Lake/volume	1
length	1
Lake Total	32
Mountain	

rdf-schema#label	13
elevation	8
mountainRange	4
locatedInArea	2
firstAscentYear	2
continent	1
Mountain Total	30
Airport	
city	7
rdf-schema#label	7
iataLocationIdentifier	7
Airport Total	21
Animal	
rdf-schema#label	6
class	3
child	2
otherName	2
description	1
type	1
range	1
family	1
water	1
movement	1
origin	1
Animal Total	20
Museum	
location	7
rdf-schema#label	7

numberOfVisitors	2
Museum Total	16
Hospital	
rdf-schema#label	9
location	3
owner	1
circle	1
bedCount	1
country	1
Hospital Total	16
Scientist	
rdf-schema#label	9
doctoralAdvisor	1
knownFor	1
birthDate	1
almaMater	1
deathDate	1
Scientist Total	14
PoliticalParty	
rdf-schema#label	13
PoliticalParty Total	13
RadioStation	
rdf-schema#label	5
programmeFormat	2
frequency	2
owner	1
broadcastArea	1
city	1

RadioStation Total	12
Building	
location	4
rdf-schema#label	3
floorCount	2
elevation	2
openingDate	1
Building Total	12
Monarch	
rdf-schema#label	9
alias	2
spouse	1
Monarch Total	12
BaseballPlayer	
rdf-schema#label	6
team	2
statisticValue	1
activeYearsStartDate	1
activeYearsEndDate	1
BaseballPlayer Total	11
Bird	
rdf-schema#label	5
conservationStatus	2
genus	2
synonym	1
Bird Total	10
Book	
author	2

rdf-schema#label	2
religion	1
releaseDate	1
Book Total	6
Plant	
rdf-schema#label	3
commonName	1
family	1
Plant Total	5
Saint	
rdf-schema#label	3
deathYear	1
Saint Total	4
Airline	
rdf-schema#label	2
iataAirlineCode	2
Airline Total	4
Wrestler	
rdf-schema#label	2
Person/height	1
Person/weight	1
Wrestler Total	4
Newspaper	
rdf-schema#label	4
Newspaper Total	4
Work	
artist	1
rdf-schema#label	1

genre	1
Work Total	3
University	
city	1
type	1
rdf-schema#label	1
University Total	3
AcademicJournal	
rdf-schema#label	2
publisher	1
AcademicJournal Total	3
Person	
rdf-schema#label	2
Person Total	2
FictionalCharacter	
rdf-schema#label	1
portrayer	1
FictionalCharacter Total	2
Hotel	
rdf-schema#label	1
address	1
Hotel Total	2
AdministrativeRegion	
rdf-schema#label	1
AdministrativeRegion Total	1
GolfPlayer	
rdf-schema#label	1
GolfPlayer Total	1

TelevisionShow	
rdf-schema#label	1
TelevisionShow Total	1
Grand Total	**652**

Table A.2: Mapped Attributes to DBpedia