Materializing Multi-Relational Databases from the Web using Taxonomic Queries

Matthew Michelson
Fetch Technologies
841 Apollo St., Ste. 400
El Segundo, CA 90245
mmichelson@fetch.com

Sofus A. Macskassy
Fetch Technologies
841 Apollo St., Ste. 400
El Segundo, CA 90245
sofmac@fetch.com

Steven N. Minton
Fetch Technologies
841 Apollo St., Ste. 400
El Segundo, CA 90245
sminton@fetch.com

Lise Getoor
Dept. of Computer Science
University of Maryland,
College Park
College Park, MD 20742
getoor@cs.umd.edu

ABSTRACT

Recently, much attention has been given to extracting tables from Web data. In this problem, the column definitions and tuples (such as what "company" is headquartered in what "city,") are extracted from Web text, structured Web data such as lists, or results of querying the deep Web, creating the table of interest. In this paper, we examine the problem of extracting and discovering *multiple* tables in a given domain, generating a truly multi-relational database as output. Beyond discovering the relations that define single tables, our approach discovers and leverages "within column" set membership relations, and discovers relations across the extracted tables (e.g., joins). By leveraging within-column relations our method can extract table instances that are ambiguous or rare, and by discovering joins, our method generates truly multi-relational output. Further, our approach uses taxonomic queries to bootstrap the extraction, rather than the more traditional "seed instances." Creating seeds often requires more domain knowledge than taxonomic queries, and previous work has shown that extraction methods may be sensitive to which input seeds they are given. We test our approach on two real world domains: NBA basketball and cancer information. Our results demonstrate that our approach generates databases of relevant tables from disparate Web information, and discovers the relations between them. Further, we show that by leveraging the "within column" relation our approach can identify a significant number of relevant tuples that would be difficult to do so otherwise.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Recently there has been significant interest in extracting relations from the Web. In this problem, relations such as what "company" is headquartered in what "city," are extracted from Web text (e.g., [2, 18, 4, 1, 15, 22]), structured Web data such as lists (e.g., [11, 9]), or queries to the Deep Web (e.g., [20]). In all cases, the goal is to generate a single table of extractions for each relation.

However, previous approaches treat relations in isolation. Even though multiple relations may be extracted, they are processed serially and do not take into account the myriad of relations that exist within a single domain. This leads to a number of problems.

First, there is an issue of deciding upon the correctness of an extracted relation, in isolation. For instance, assume a linguistic extractor includes the pattern "X located in Y" where X is a company and Y is a city, and encounters a sentence such as "Skyscraper located in Los Angeles!" The system must reason as to whether "Skyscraper" is a company. Although it fits the pattern, the term skyscraper is an ambiguous extraction because it could refer to a building or a company. There are methods to deal with the issue of extraction correctness (e.g., [8]), but the larger issue is the difficulty that arises when considering extractions in isolation from the (sub)set of representative values. Instead, it would be preferable to discover and exploit a set membership relation directly. Given a column of values, implying set membership, if this column contains a few well-known companies, and contains "Skyscraper" then it is likely to be a company too.[1]

[1]According to previous work, "instances that co-occur very frequently in the same column/row with seeds S are often found to be correct (e.g., a column containing the seeds 'Brad Pitt' and 'Tom Hanks' will likely contains other actors)" [16].

Second, since traditionally the extracted relations are isolated from one another, relations *across* the various tables of extracted values are not considered. One table might extract companies and their cities, and another companies and their CEOs. However, the joins between these tables are not discovered and considered.

Finally, many methods start by using input "seeds," which are example instances of relations (e.g.,[18, 4, 1, 15, 22, 11]). However, starting with seeds has two issues. First, previous work demonstrated that when extracting similar items based on seeds, the accuracy can vary significantly depending on the choice of seeds [19]. This puts significant pressure on the human user to generate seeds that are not too prototypical and which provide good domain coverage [19]. Second, choosing good seeds may require significant domain knowledge. In many cases, a user wants the machine to automatically compile information precisely because he/she wants to learn more about the domain. In this case, the user does not have enough domain knowledge to create useful seed relations. Instead, it would be preferable to relax this restriction such that a person only needs knowledge about the domain, rather than specific instances.

In this paper we describe an approach to extracting relations that addresses the problems outlined above. Our work is motivated by the hypothesis that there is a significant amount of semi-structured data on the Web that is *independently organized around subsets of categories.* For instance, one person (or company) builds a Webpage that lists NBA teams and their arenas, while another person builds a Webpage with legendary players, their teams, their rookie year, and the points they scored. Therefore, the goal is for the system to discover, extract, and relate these independent sources of relational information, thereby materializing a single database representing these sources.

Our paper makes the following contributions:

- We formalize and define a novel and challenging problem that generalizes from the traditional problem of relation extraction. We also present an end-to-end approach that serves as baseline solution for this problem.
- Our method starts with phrases representing domain knowledge (called "taxonomic queries"), rather than seed instances. For example, a user interested in materializing a database about NBA basketball does not need to develop a seed list of teams and their associated players, coaches, etc., but rather he/she uses the taxonomic query, "NBA teams," "NBA players," instead. Taxonomic queries require less domain knowledge since they are organized around entity types, and they overcome the issue of seed sensitivity.
- Our approach extracts whole sets of structured data from each source, which we represent as tables, and we show that we can therefore exploit the within-column set-membership relation to find highly relevant columns. Further, this within-column relation identifies instances that would be hard to discover in isolation due to their rarity or ambiguity.
- Our method finds joins *across* relational data extracted from disparate sources, materializing a truly multi-relational database.

Figure 1 shows the architecture for our approach. To start, a user provides a few "taxonomic queries," and the DiscoveryEngine retrieves the top-K webpages associated with each taxonomic query. Next, the AutoWrap component extracts structured data from the returned webpages, and turns it into tables. AutoWrap creates tables from all of the structured data it can find (lists, tables, etc.), and therefore it includes data that is not relevant to the domain. Therefore, the ColumnScrubber next projects only the clean columns from the extracted tables, resulting in a set of cleaned tables. Finally, the JoinFinder finds both "explicit" joins and "implicit" joins across the cleaned tables. The result is a materialized database related to the set of input taxonomic queries.

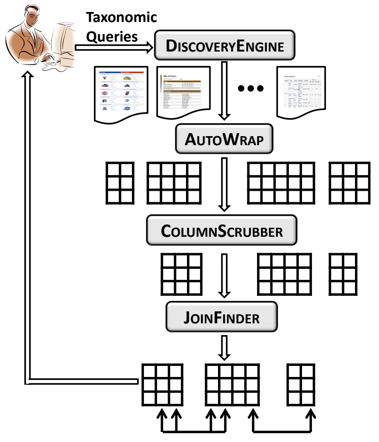


Figure 1: Our overall approach

The rest of this paper is as follows. Section 2 formally describes our problem and method. Section 3 describes the details of our approach, and Section 4 presents experiments validating our method. Section 5 outlines related research, and Section 6 presents our conclusions.

2. PROBLEM DEFINITIONS

As stated in the Introduction, we believe relevant Web information is sometimes organized around "taxonomic queries," which can be used to find and extract relational data from semi-structured sources (e.g., Webpages). We define a domain $\mathcal{D}$ as the category that describes collections of various entities (grouped by their types) and their relations. Our goal is to try to reconstruct a multi-relational description of $\mathcal{D}$ seeded by the names of the entity types. We define a *taxonomic query* as a phrase consisting of the domain name D and one of the entity types $e \in E$. For example, given $\mathcal{D}$ ="NBA Basketball," D ="NBA" and $e \in${players, teams, ...}, we generate the set of taxonomic queries: Q ={"NBA players," "NBA teams," ...}. Taxonomic queries can cover a large range of data types, ranging from our NBA example, to cancer information,[2] to country information.[3] Each taxonomic query represents a distinct entity type, and therefore users will sometimes organize data on their Webpages around them.

For each taxonomic query $q_y \in Q$, our approach generates a set of Tables $T_k^{q_y}$ built by extracting data from structured pieces of the Webpages (e.g,. the lists, tables, etc. that

[2] D ="Cancer" and $e \in${medicines, treatments, types, ...}
[3] D ="United States" and $e \in${states, capitals, ...}

occur on the page). For clarity, we will sometimes remove the superscript q_y from our definitions, implying that all T_k are generated by the same taxonomic query. We define t_k^j as the jth tuple of T_k, and c_k^{ij} as the ith column instance of the jth tuple. Therefore, we define C_k as all of the columns of T_k (e.g., $C_k = \bigcup c_k^i$).

However, not all columns for each extracted table are relevant to the domain (e.g., noise). Therefore, the next goal is to project the relevant columns from T_k. C_k decomposes into two disjoint sets C_k^R, the set of relevant columns, and C_k^N, the set of noisy columns (such that $C_k^R \cup C_k^N \equiv C_k$). Therefore, a cleaned table T_k^* is defined as:

$$T_k^* : \pi_{\{\forall c : c \in C_k^R \setminus C_k^N\}}(T_k)$$

Given a set of cleaned tables $T^* = \{T_1^*, \ldots, T_k^*\}$, the last step discovers the joins between them (again we assume all tables come from the same taxonomic query, so we ignore the superscript q_y). There are two types of joins, *explicit* joins and *implicit* joins. Explicit joins between clean tables are defined as natural joins (though we also describe how to support "fuzzy" matches as joins). Given clean tables T_x^* and T_y^*, an explicit join is just $T_x^* \bowtie_{x,y} T_y^*$.

An implicit join is a join that requires outside information to perform the natural join. That is, two tables may not have any columns that match for an explicit join, but by using an outside information source (such as the Web), join equivalence can be established. For example, one table may have NBA players and teams, and another may have NBA coaches and arenas. By using the Web, we can figure out how to assign NBA teams to arenas, and use this assignment to join the tables. Therefore, we define an implicit join as:

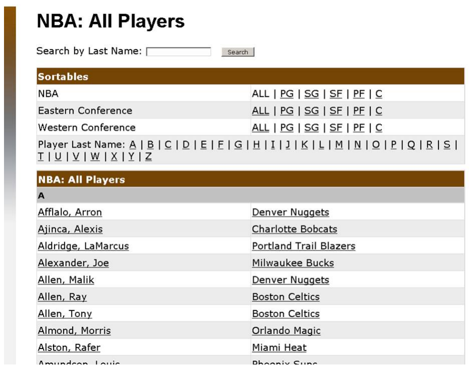
$$T_x^* \bowtie_{(\text{IMP}_{x,y})} T_y^*$$

where $\text{IMP}_{x,y}$ is a look-up table mapping the implicit joins from T_x^* to T_y^*. While our example shows a functional dependency, we note that implicit joins could represent multi-valued dependencies, as we describe below.

Overall, our problem takes as input the taxonomic queries $q_z \in Q$, and outputs three sets: the cleaned, extracted tables $T^{*Q} = \{\forall q_z \in Q : \cup T^{*q_z}\}$, the discovered explicit joins $J_e = \{(x,y) : T_x^{*Q} \bowtie_{x,y} T_y^{*Q}\}$, and the discovered implicit joins $J_i = \{(x,y) : T_x^{*Q} \bowtie_{(\text{IMP}_{x,y})} T_y^{*Q}\}$. Therefore, the resulting output represents a materialized multi-relational database composed of the cleaned tables and the joins between them, all related to the few, taxonomic queries.

3. OUR APPROACH

In this section, we present the details of each component of our algorithm shown in Figure 1. We motivate our approach with the hypothesis that people will independently organize data on the Web around taxonomic queries. Consider the example data from Figures 2 and 3 that show two different sets of NBA data. Figure 2 contains a list of players (and their associated teams), while Figure 3 shows a table of some of the current NBA coaches. These pages were built independently, but by extracting the structured data representing the players and coaches, and then discovering the join across the resulting extracted tables (using the team name), we can materialize a database relating players, teams, and coaches from the NBA.



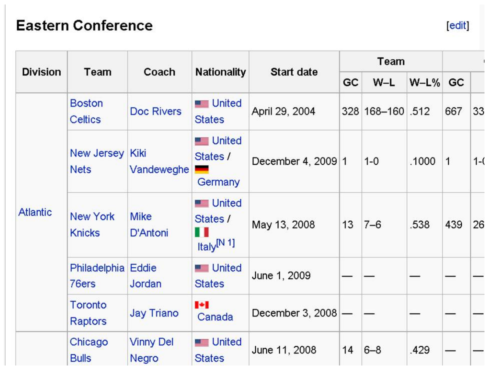
NBA: All Players

Search by Last Name: Search

Sortables	
NBA	ALL | PG | SG | SF | PF | C
Eastern Conference	ALL | PG | SG | SF | PF | C
Western Conference	ALL | PG | SG | SF | PF | C
Player Last Name: A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z	

NBA: All Players	
A	
Afflalo, Arron	Denver Nuggets
Ajinca, Alexis	Charlotte Bobcats
Aldridge, LaMarcus	Portland Trail Blazers
Alexander, Joe	Milwaukee Bucks
Allen, Malik	Denver Nuggets
Allen, Ray	Boston Celtics
Allen, Tony	Boston Celtics
Almond, Morris	Orlando Magic
Alston, Rafer	Miami Heat
Amundson, Louis	Phoenix Suns

Figure 2: List of NBA Players from espn.com



Eastern Conference [edit]

Division	Team	Coach	Nationality	Start date	Team				
					GC	W–L	W–L%	GC	
Atlantic	Boston Celtics	Doc Rivers	United States	April 29, 2004	328	168–160	.512	667	33
	New Jersey Nets	Kiki Vandeweghe	United States / Germany	December 4, 2009	1	1-0	.1000	1	1-
	New York Knicks	Mike D'Antoni	United States / Italy[N 1]	May 13, 2008	13	7–6	.538	439	26
	Philadelphia 76ers	Eddie Jordan	United States	June 1, 2009	—	—	—	—	—
	Toronto Raptors	Jay Triano	Canada	December 3, 2008	—	—	—	—	—
	Chicago Bulls	Vinny Del Negro	United States	June 11, 2008	14	6–8	.429	—	—

Figure 3: List of NBA Coaches from Wikipedia

3.1 Source discovery with Taxonomic Queries

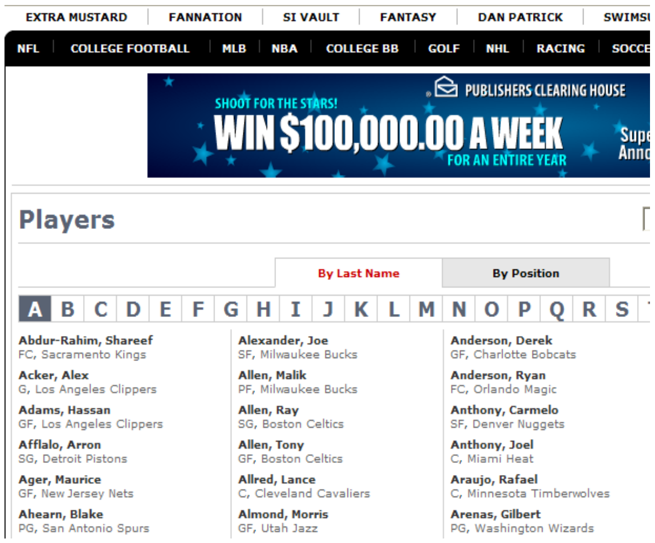
Our method uses the DiscoveryEngine to find sources related to the taxonomic queries. Since each taxonomic query is essentially a phrase relating a domain name and entity type, our approach passes each taxonomic query to a search engine (Google), and then retrieves the top-K returned pages. It then passes the retrieved pages to the next component, *AutoWrap*.

3.2 Turning Structured Data from Webpages into Tables

Our key idea is that taxonomic queries are useful because when people organize data around taxonomic queries, they tend to (semi-)structure it (e.g., build relational data on the Web). For instance, many times people will include lists or tables on Webpages to represent a grouping of data. Such grouped, semi-structured data is generally more complete, in that it usually contains many instances, and is already structured. Therefore, an important component of our approach is discovering the structured data on a Webpage and extracting it, thereby turning the into a table. That is, we want to turn all of the possible structured pieces of a Webpage, such as the lists, tables, text-delimited values, etc., into tables. To extract tables from semi-structured sources we leverage previous work called *AutoWrap* [10].

AutoWrap does just this: taking semi-structured data, such as a Webpage, as input, and extracting the relational data from the page, creating tables. AutoWrap identifies generic lists and tables on sites, regardless of their format. That is, it recognizes repeated patterns on pages that allow it to extract lists and tables, without relying on any specific HTML formatting tags, such as "<table>" tags. It functions by analyzing websites for their common structure, relying on a set of heterogeneous software "experts." Each expert analyzes a certain type of generic structure, such as the layout of a site's pages, or the structure of a site's URLs.

For example, Figure 4 shows a web pages about NBA players, from SI.com (Sports Illustrated). The page is composed of multiple lists of NBA players, and each item in the list consists a player's name, position, and team. Therefore, AutoWrap realizes this page represents separate lists to be appended together to create a table, extracting the fields for each item in the list to create the rows in the table. Figure 5 shows a screenshot of the console output produced by AutoWrap for the webpage of Figure 4, showing tabular data that the system identified. In the console output each table is given an id, and vertical bars are used to separate the individual columns. As a close inspection of the figure shows, the system successfully extracts the list of players from the site, creating a table with the player's name, position, and team. Of course, AutoWrap also creates many additional tables that are not particularly relevant to reconstructing the NBA domain, such as the list of tab names from the page "Extra Mustard, Fannation, SI Vault, . . ."



Abdur-Rahim, Shareef
FC, Sacramento Kings

Acker, Alex
G, Los Angeles Clippers

Adams, Hassan
GF, Los Angeles Clippers

Afflalo, Arron
SG, Detroit Pistons

Ager, Maurice
GF, New Jersey Nets

Ahearn, Blake
PG, San Antonio Spurs

Alexander, Joe
SF, Milwaukee Bucks

Allen, Malik
PF, Milwaukee Bucks

Allen, Ray
SG, Boston Celtics

Allen, Tony
GF, Boston Celtics

Allred, Lance
C, Cleveland Cavaliers

Almond, Morris
GF, Utah Jazz

Anderson, Derek
GF, Charlotte Bobcats

Anderson, Ryan
FC, Orlando Magic

Anthony, Carmelo
SF, Denver Nuggets

Anthony, Joel
C, Miami Heat

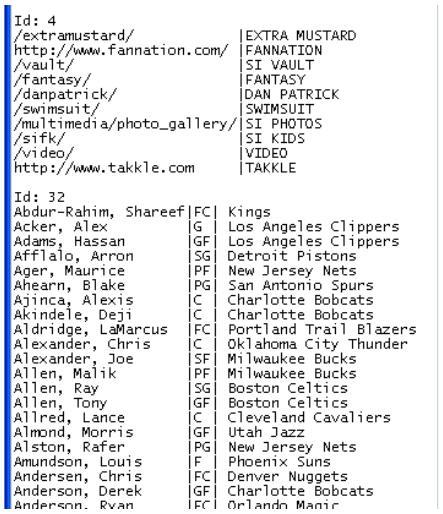
Araujo, Rafael
C, Minnesota Timberwolves

Arenas, Gilbert
PG, Washington Wizards

Figure 4: NBA Players on SI.com

AutoWrap can also union together tables from multiple pages if the repeated structure used to identify the table is the same.[4] That is, while the figure shows all players whose names start with "A," AutoWrap can generate the full list of players from SI.com by extracting the same table (exploiting the same redundancy in structure) for pages with players whose name start with "B," "C," etc., and then unioning those tables together.

AutoWrap generates tables for all possible relational data it finds on webpages, and therefore it can generate a significant number of tables per page, some of which will not be relevant for reconstructing the domain (e.g., a table of advertisement links). Therefore, extraction must be followed



```
Id: 4
/extramustard/                |EXTRA MUSTARD
http://www.fannation.com/     |FANNATION
/vault/                       |SI VAULT
/fantasy/                     |FANTASY
/danpatrick/                  |DAN PATRICK
/swimsuit/                    |SWIMSUIT
/multimedia/photo_gallery/|SI PHOTOS
/sifk/                        |SI KIDS
/video/                       |VIDEO
http://www.takkle.com         |TAKKLE

Id: 32
Abdur-Rahim, Shareef|FC| Kings
Acker, Alex         |G | Los Angeles Clippers
Adams, Hassan       |GF| Los Angeles Clippers
Afflalo, Arron      |SG| Detroit Pistons
Ager, Maurice       |PF| New Jersey Nets
Ahearn, Blake       |PG| San Antonio Spurs
Ajinca, Alexis      |C | Charlotte Bobcats
Akindele, Deji      |C | Charlotte Bobcats
Aldridge, LaMarcus  |FC| Portland Trail Blazers
Alexander, Chris    |C | Oklahoma City Thunder
Alexander, Joe      |SF| Milwaukee Bucks
Allen, Malik        |PF| Milwaukee Bucks
Allen, Ray          |SG| Boston Celtics
Allen, Tony         |GF| Boston Celtics
Allred, Lance       |C | Cleveland Cavaliers
Almond, Morris      |GF| Utah Jazz
Alston, Rafer       |PG| New Jersey Nets
Amundson, Louis     |F | Phoenix Suns
Andersen, Chris     |FC| Denver Nuggets
Anderson, Derek     |GF| Charlotte Bobcats
Anderson, Ryan      |FC| Orlando Magic
```

Figure 5: Data extracted by AutoWrap

by a component for cleaning the extracted tables, leaving only data relevant for the domain.

3.3 Cleaning the Extracted Tables

Since AutoWrap extracts all possible relational data from sites, the next challenge is determining which of the created tables and columns from the extracted data are relevant and which are not. To do this, our ColumnScrubber algorithm prunes away the tables and columns that are not relevant (as we define below). It prunes irrelevant columns one at a time, such that if all of the columns are removed from a table, this prunes away the entire table.

To clean the tables, our ColumnScrubber algorithm exploits the set-membership relation defined by the column's instances. Intuitively, if some of the instances c_k^{ij} in column c_k^i are definitely relevant (not noise), then the whole column c_k^i is likely to be relevant (not noise), despite the fact that other members of the column are not obviously relevant. We define the relevance of a column instance based upon the conditional probability (CP) of generating the domain name D given the instance c_k^{ij}.[5] For this calculation we define a function f("phrase") that returns the number of hits from a search engine (Google) for the given "phrase," and divides it by the size of the search index (assuming 10B pages). Therefore, for a given instance c_k^{ij} and domain name D, we define:

$$CP(D|c_k^{ij}) = \frac{f(D \wedge c_k^{ij})}{f(c_k^{ij})}$$

As $CP(D|c_k^{ij})$ approaches 1, it suggests that the column instance occurs mostly with the domain name, suggesting a strong relationship between the two.[6] As it approaches zero,

[4]Since we don't spider from the pages returned by the DiscoveryEngine, we do not use this feature in our current method.

[5]We use the domain name rather than the specific taxonomic query for the column, since relevant columns can be related to a domain without being specifically related to the taxonomic query. For example, the query "NBA players" may generate a table with columns of both players and teams. The teams column is related to the domain but not as related to the taxonomic query that generated it.

[6]One can add the constraint that $f(c_k^{ij}) > F_{min}$, a minimum frequency, to avoid artefacts. We omit this constraint here.

it suggests one of two cases. Either the column instance is quite common and does not often occur solely with the domain name, and so the relationship is weaker (we call this an "ambiguous" instance), or it so rarely occurs on the Web in general, it barely ever occurs with the domain name (the "rare" instance).

Based upon the above formulation for an instance's relevance, we define the column's relevance as the probability of the column containing relevant instances. More formally, given a column $c_k^i \in C_k$ (C_k is all columns), we define $Relevance(c_k^i)$ as the fraction of instances where $CP(D|c_k^{ij})$ is above tm, the "match threshold."

$$Relevance(c_k^i) = \frac{1}{\|c_k^i\|} * \sum_{c_k^{ij} \in c_k^i} \begin{cases} 1, & CP(D|c_k^{ij}) > tm \\ 0, & \text{Otherwise} \end{cases}$$

Our definition of relevance exploits the structure of list membership by relying on the few obviously relevant instances. In contrast, if we were to try and classify the relevant column instances in isolation, then the rare and ambiguous instances would be more difficult to classify as relevant due to their lack of Web evidence. Anecdotally, we found a number of cases, such as NBA players and teams, where the hits for column instances follow a long-tail distribution (many rare instances), motivating this approach. Such long-tail distributions for various classes have been noted previously, for example, Wu, et al. [22] show that on Wikipedia both the number of articles in categories and the lengths of articles themselves follow a long tail distribution where a few categories dominate, and there are a few very long articles and many "stub" articles that lack most of the information.

Table 1: Cleaning columns from extracted tables

```
ColumnScrubber(Tables $T$, Domain name $D$,
      Tail Size $ts$, Match Threshold $tm$)
1.  $T^C \leftarrow \{\}$
2.  $\forall T_k \in T$
3.      $C_k^R \leftarrow \{\}$
4.      $C_k^N \leftarrow \{\}$
5.      $\forall C_k \in T_k$
6.          $\forall c_k^i \in C_k$
7.              if( $Relevance(c_k^i) > ts$ )
8.                  $C_k^R \leftarrow C_k^R \cup c_k^i$
9.              else
10.                 $C_k^N \leftarrow C_k^N \cup c_k^i$
11.     $T_k^* \leftarrow \pi_{\{\forall c : c \in C_k^R \setminus C_k^N\}}(T_k)$
12.     $T^C \leftarrow T^C \cup T_k^*$
```

The whole ColumnScrubber algorithm for pruning tables using this method is given in Table 1. As in Section 2, for clarity we remove superscripts referencing the taxonomic queries. This algorithm iterates over all of the columns from table T_k and splits them into the disjoint set of relevant columns C_k^R and noisy columns C_k^N. It then assigns a clean table, T_k^*, as the projection of relevant columns from the original table. Beyond the "match threshold" parameter, tm, this algorithm also defines a "tail size" (ts) parameter that defines a threshold for the relevance probability, above which a column is considered relevant enough to keep. The ColumnScrubber operation acts as a projection of the clean columns from all of hte columns that are discovered.

3.4 Relations Across Tables

The cleaned tables that result from ColumnScrubber can then be related to one another via joins. There are two types of joins our approach discovers across these cleaned tables. First, there are "explicit" joins. An *explicit* join is one that can be discovered using only the information provided by the tables. For instance, we can join the table generated from Figure 2 with that extracted from Figure 3, by joining on the team name explicitly. Explicit joins are not limited to using single columns, but may involve linking multiple columns across the tables (e.g., matching both player names and teams across tables).[7] For our purposes, we define an explicit join across two (or more) columns if the instances of one column are completely subsumed by the other column (e.g., all of the values from column c_k^a appear in column C_l^b, such that $c_k^a \subseteq c_l^b$).

In this manner, even multi-column joins can be identified because the overlap is defined on a per-column basis. That is, one table may contain NBA players from a specific team, with columns of the player's name, team, and coach. Another table may contain data about all NBA teams, including the team and coach, and therefore, we can join the table of players to the team based on the fact that the team and coach values of the first table will be subsumed by the second.

However, there are also useful joins that are not explicit. The organization of data around taxonomic queries leads to tables that would be useful if linked together, but may be independent of one another. For instance, one page may be organized around NBA players, and another around NBA arenas, but neither page mentions entities from the other (precluding an explicit join). Nonetheless, relating players to their home arenas is a useful relation for the domain. We call these "implicit" joins. An *implicit* join is one that can be discovered using information outside of that provided solely by the tables. In our case, the Web as our outside information.

Our method for discovering implicit joins across the tables is motivated by the "AltaVista mutual information" metric [17], which calculates the point-wise mutual information between terms based on the number of hits returned from the AltaVista search engine. However, given the vastness of the Google search engine's page index, we use Google rather than AltaVista. (Previous researchers have used Google similarly, for example to determine term taxonomies [13].) Further, we note that while Turney [17] made explicit use of the "NEAR" operator between search terms for AltaVista queries, the Google search syntax seems to support NEAR implicitly through its AND keyword syntax, so we focus solely on using AND-based co-occurrence. We call this method "GMI" (Google Mutual Information) for simplicity.

More specifically, given two different columns, c_k^a and c_l^b, from two different cleaned tables T_k^C and T_l^C respectively, we define GMI over two instances, $c_k^{ai} \in c_k^a$ and $c_l^{bj} \in c_l^b$ from the columns as:[8]

$$GMI(c_k^{ai}, c_l^{bj}) = f(\text{"}c_k^{ai} \text{ AND } c_l^{bj}\text{"}) * \log \frac{f(\text{"}c_k^{ai} \text{ AND } c_l^{bj}\text{"})}{f(c_k^{ai}) f(c_l^{bj})}$$

[7]In fact, joins where instances do not match exactly can also be found efficiently [6]. However, we focus on exactly matching instances in this study.

[8]GMI uses the f("phrase") function to define hit counts.

Given the full set of GMI scores for the cross product $c_k^a \times c_l^b$, we can then assign implicit joins by assigning the instance c_k^{ai} for each instance c_l^{bj} that maximizes the GMI score. This can result in both 1-1 and 1-N joins across the columns (for M-N style joins we can simply keep the set from c^{bj} that have either scores for c_k^{ai} above some threshold, or that are maximally scoring instances).

There are a number of properties that motivate the choice of this method. First, mutual information is a symmetric relation (that is, GMI(i,j) = GMI(j,i)), which is a requirement for the discovered joins (i.e., NBA players and NBA teams are related symmetrically). Second, this selection mechanism defines 1-n relations from the chosen matches. For example, multiple players play for a single team. It is easy to discover the order of the cardinality of the relations by simply analyzing the results (e.g., NBA players become the "many" side of the relation, while NBA teams are the "one"). Therefore, we can assign cardinality information to the resulting implicit joins if required. Now that we have defined both explicit and implicit joins, Table 2 describes the *Join-Finder* algorithm from Figure 1 in Table 2.

We note that our approach compares all pairs of tables for the joins. However, scalability is not much of an issue because many of the tables and columns are cleaned away. Further, previous efforts demonstrate efficient mechanisms for finding explicit joins [6], and scalability for implicit joins can be improved further by first sampling pairs and testing to see if any implicit join matches are discovered before testing the whole cross product of the columns. Finally, we note that tables that have one type of join (e.g., an explicit join) are not considered for implicit joins, relieving redundant computations.

Table 2: Finding joins across cleaned tables

```
JOINFINDER(Tables $T$)
1. Explicit joins $J_e \leftarrow \{\}$
2. Implicit joins $J_i \leftarrow \{\}$
3. $\forall T_x^C, T_y^C \in T, x \neq y$
4.     $J_{e_{xy}} \leftarrow$ EXPLICITJOINS($T_x^C, T_y^C$)
5.     if( $|J_{e_{xy}}| > 0$)
6.         $J_e \leftarrow J_e \cup J_{e_{xy}}$
7.     else
8.         $J_{i_{xy}} \leftarrow$ IMPLICITJOINS($T_x^C, T_y^C$)
9.         if( $|J_{i_{xy}}| > 0$)
10.            $J_i \leftarrow J_i \cup J_{i_{xy}}$
```

4. EXPERIMENTS

Our experiments focus on two diverse, real-world domains: NBA basketball and cancer information. Table 3 lists each domain and the taxonomic queries for the domains (we set K=5 for the DiscoveryEngine, resulting in 5 returned pages per query). This table also shows the results describing the extracted tables (prior to cleaning), including and the number of tables, the total number of columns, the average number of columns per table, the total number of rows, and the average number of rows per table.

Our first set of experiments tests our approach to pruning the noise from the many extracted tables and columns. Therefore, we manually labeled columns as relevant to a domain or not. In the NBA domain 509 of the 3,356 columns are relevant (15.17%) and for Cancer 35 of the 429 columns are relevant (8.16%).

The 509 relevant NBA columns are distributed across 216 distinct tables generated by the taxonomic queries, and the 35 relevant Cancer columns are distributed across 27 different tables. The relevant columns are therefore distributed across many tables, implying that databases of relevant information can be compiled from *independently* generated semi-structured sources discovered from the Web. Since relevant columns are distributed across distinct relevant tables from disparate sources, the data is indeed organized around taxonomic queries. Therefore, taxonomic queries can prove effective for bootstrapping the discovery of relational information.

Table 3: Domains with extracted-table results

NBA Domain					
Taxonomic Query	**# Tables**	**Total Cols**	**Avg. Cols**	**Total Rows**	**Avg. Rows**
NBA Arenas	53	291	5.49	1,238	23.36
NBA Coaches	53	496	9.36	2,080	39.25
NBA Players	85	1,061	12.48	6,929	81.52
NBA Teams	85	1,508	17.74	8,477	99.73
Total	276	3,356	12.16	18,724	67.84
Cancer Domain					
Taxonomic Query	**# Tables**	**Total Cols**	**Avg. Cols**	**Total Rows**	**Avg. Rows**
Cancer Medicines	30	70	2.33	1,044	34.80
Cancer Treatments	40	144	3.60	1,264	31.60
Cancer Types	43	215	5.00	1682	39.12
Total	113	429	3.80	3,990	35.31

4.1 Finding Relevant Columns

Next, we demonstrate our ability to find relevant columns, and that column membership is useful for discovering instances related to the domain. One benefit of attacking the extraction problem as one of finding and compiling tables (rather than isolated instances from text) is that for members of the long tail (e.g., unknown NBA players) or ambiguous members (the NBA team "Jazz"), column membership can be exploited to aid in deciding whether or not the member is a relevant instance. In particular, as Downey, et al. [8] stated "a fundamental problem for both supervised [information extraction] and [unsupervised information extraction] is assessing the probability that extracted information is correct." Therefore, we demonstrate the benefit of using column membership as a means for determining relevant column instances. We note that our method is therefore complementary to linguistic based extraction methods as a method to deal with long-tail or ambiguous column instances.

We first demonstrate that relevant column instances are hard to identify in isolation (which also shows the difficulty in selecting relevant columns during cleaning). Specifically, we analyze distributions of the computed CP values for the columns. To generate the distributions, we create bins of the conditional probability (CP) values (0-10%, 10-20%, ...) for each true positive column, where each bin marks the number of instances in a column that have CP values within that bin. This allows us to build a distribution over the bins for each column. Then, using these distributions, we can build further distributions showing the percent of the time a column is composed of a certain bin. That is, we use the distributions over the columns to compute the percentage of time that 60-70% of a column instances contain conditional probabilities between 70-80%. Figures 4.1 and 4.1 show the resulting graph for all CP values and all percentages in each domain, respectively. In the figures, the y-axis represents the fraction of time a bin composes the given percent of the column (which are the x-axis).

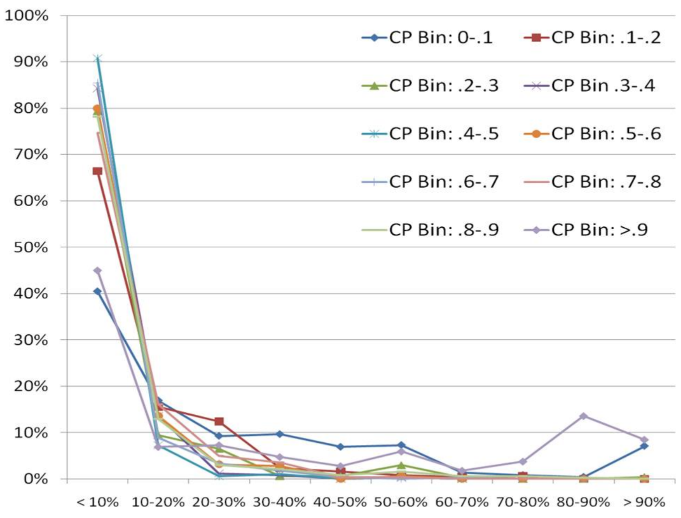


Figure 6: CP Distributions for NBA

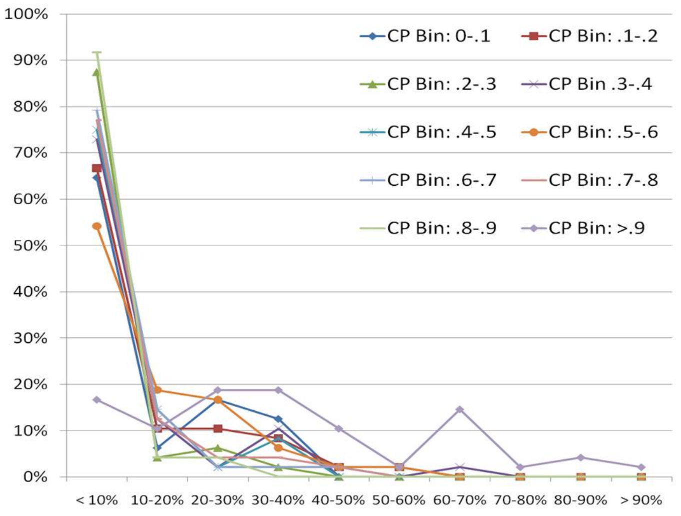


Figure 7: CP Distributions for Cancer

If columns were easily identifiable based solely on the CP values of their instances (e.g., most instances have high CP values), we would expect the graphs in the figures to look like long-tail distributions from right to left (fat-tail on the right) e.g., most columns would fall into high percentages with high CP values. However, for all domains we see the opposite distributions. In fact, these graphs indicate that columns mostly contain a mixture of CP values, and there are very few cases where the columns are filled solely with highly probable instances. Therefore, classifying columns based on the few relevant instances should yield gains in classifying correct instances (shown below).

Our next experimental results show that being able to identify a few easily recognizable instances aids in finding relevant columns. The results are given in Figure 8. The y-axis of this figure shows the F-measures for selecting columns based on our ColumnScrubber algorithm,[9] varying both the match threshold for the CP score, and the tail size (shown as x-axis in the graph). We used CP thresholds of 70%, 80%, 90%, for each domain, representing those instances that are most easily identifiable. For the most part, when only a small tail size is used (e.g., 5%, 10%, 20%) the algorithm performs significantly better than when larger values are required, or even a majority is required (tail sizes greater than 60%). This clearly demonstrates two points: first, the few easily identifiable members can indeed be leveraged for finding relevant columns, even though a large percentage of the column instances may not be easily identifiable. This is demonstrated by the fact that the best F-measures come from the smallest tail sizes. Second, few columns exist in the data that have a majority of the instances as easily identifiable, otherwise the graph would be horizontal, since requiring at least 10% of the instances to have high CP would yield the same F-measure as requiring at least, say, 70%. We note that even though for Cancer the F-measure is below 50%, we emphasize the small priors of a column being relevant for NBA and Cancer are 15.17% and 8.16% respectively. Therefore, even though this is a difficult task, we are able to successfully identify relevant columns.

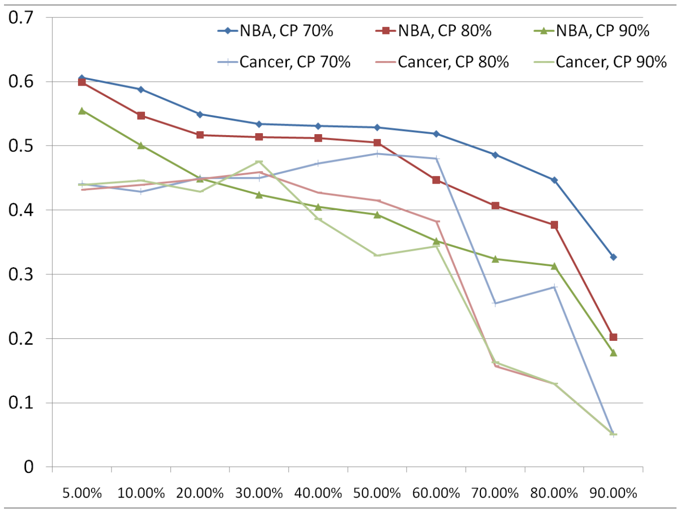


Figure 8: F-measures for selecting columns

Although the relevant columns are difficult to discover, we next show that leveraging the relevant columns our approach does discover yields significant improvement for labeling relevant instances, as compared to labeling the instances in isolation. In particular, this experiment serves as a demonstration of how using discovered tables complements methods that solely examine extractions in isolation.[10] We assume we have a perfect linguistic based extractor that discovers all of the possible extractions in this true positive set (e.g., an extractor with 100% recall). Next, as described in previous work [8], the challenge is to determine which of the made

[9]F-measure=(2*Precision*Recall)/(Precision+Recall)

[10]For this experiment we labeled the instances of columns as true positives, noting that not all instances in a true positive column are themselves true positives (e.g., a column of NBA players may have a header for the column with the term "players").

extractions should be kept and which should be ignored as noise. To simulate this decision, this extractor classifies any instance with a CP value greater than the match threshold as relevant. These would be the extractions that a system could identify in isolation using information such as Web co-occurence with the domain. We compare the F-measure of labeling instances in this manner, which simulates extracting instances in isolation, with our approach that labels all instances as relevant if they belong to a column that is selected as relevant.

To be clear, given an input match threshold of 70%, we first classify all instances as correct if their individual CP score is at least 70%. We then compare this to classifying all instances as correct if the column they belong to is classified as correct based upon this same threshold. Table 4 shows these comparative results, showing the percent difference in F-measure between labeling instances by exploiting column membership versus labeling in isolation. We varied the conditional probability thresholds from 70% to 90%, reflecting instances that run the spectrum of easily identifiable, and we vary the tail size from 5%-15% for our column selection, since those are the areas of the curves of Figure 8 where our algorithm does the best selecting relevant columns.

Table 4: Increase in F-Measure

		NBA	Cancer
Tail Size	CP Thresh	% Diff in F-Mes.	% Diff in F-Mes.
5%	70%	+8.94	+20.22
	80%	+12.95	+28.32
	90%	+13.43	+33.24
10%	70%	+4.73	+20.24
	80%	+8.27	+28.97
	90%	+10.71	+33.90
15%	70%	+0.13	+22.61
	80%	+4.94	+30.94
	90%	+7.96	+35.10

Our method yields an average increase of +18.09% in F-measures, across match thresholds and tail sizes. Therefore, there is a significant benefit to labeling instances as relevant when they belong to a relevant column (exploiting the set membership relation) as compared to deciding upon an extraction's relevance in isolation.

While we demonstrated our approach can clean tables, such tables in isolation do not answer our goals of semi-automatically materializing databases. Therefore, we now test whether we can automatically discover useful joins across the tables, making them richer relationally.

4.2 Discovering Joins

Our next experiment describes the discovery of *explicit* joins. These are joins with exact matches between columns across tables. To test our method, we found all of the joins that occur across the tables in our ground truth columns, which allows us to quantify how well we find the joins using the columns selected by ColumnScrubber. We then found the explicit joins (exact matches across columns) from columns selected by our approach, with a CP threshold of 70% and a tail size of 5% (since this produces the best results in selecting columns), and calculate the F-measure. In the NBA domain, the F-measure for finding explicit joins is 80.09%. In the Cancer domain, the F-measure is 20.69%. This is largely due to false positive joins found across noisy columns that were not pruned. However, given the difficulty of each sub-problem (semi-automatically finding sources of relations, extracting those relations, and relating them) these results are quite encouraging.

Our final set of experiments both tests our ability to uncover *implicit* joins, and shows the variability that can result when extracting relations using seeds as a starting point. For this experiment, the goal is to discover the implicit joins (relations) that occur between all pairs of columns represented by the taxonomic queries. That is, we want to find the following joins: for NBA {Players/Teams, Players/Coaches, Players/Arenas, Teams/Arenas, Teams/Coaches, Coaches/Arenas}, for Cancer {Medicines/Types, Medicines/Treatments, Types/Treatments}. Within these joins, the goal is to find the correct relational occurrence between the column instances (e.g., find the correct NBA team for each NBA player). The more correct relational occurrences discovered, the cleaner (and better) the resulting implicit join will be. Since it would be infeasible to analyze all possible pairs of instances within all implicit joins for accuracy, we instead hand select the columns deemed relevant by our system (using a tail size of 5% and match threshold of 70%) that best represent the taxonomic query, and then analyze the resulting GMI-based instance matches across these columns to analyze the quality of the discovered join. This models how well we could do for the representative columns. Note that these thresholds are the same for both domains.

For the NBA relations, we construct ground truth by extracting data from the ESPN website,[11] and for the Cancer domain we extract a subset of ground truth for the Medicines/Types relations from the Chemocare website.[12] Therefore, we can effectively test F-measure for the NBA data, since we know all possible truth values. For the Cancer domain, we calculate precision for Medicine/Types using the truth values (recall is unknown since we do not have an enumerated set), and for the Medicines/Treatments and Types/Treatments we calculate precision by spot-checking 25 values and analyzing their correctness by hand, using Google. In this manner, our ground truth for the joins is mostly constructured by outside, independent sources, and then extracted, rather than creating it ourselves.

Essentially, discovering implicit joins using GMI is the same as discovering the relations between column instances. Therefore, we compare our GMI method to two different approaches, both of which extract relations from text. We developed the first approach, called "Ling-Seed," for extracting relations from unstructured text. It follows recent work on text extraction (e.g., [4, 1, 15]), taking as input X seed relations (e.g., pairs of players and their teams),[13] and then mining the most frequent surface patterns in text where these relations occur. Its corpus is the first 400 returned snippets of text from searching Google for the seeds (which we deduplicate). After mining patterns, it queries Google using the top-K discovered surface patterns (K=10), and analyzes the returned snippets to see which true relations exist in the returned text. For this comparison, we compute the number of correct relations found over the returned pages, which we then reduce to a recall/page metric ("R/Pg"). This essentially averages the number of correct relations we could find, per page, using Ling-Seed, such that if we were to examine 10,000 pages, assuming independence and equal distribu-

[11]http://espn.go.com/nba/players

[12]http://www.chemocare.com/bio/

[13]X=10

Table 5: Implicit Join Results

	Implicit Joins (GMI)			TextRunner	Ling-Seed (R/Pg)	
Relation	*Recall*	*Prec.*	*F-meas.*	*F-Meas.*	*Avg.*	*Var.*
Players/Teams	0.68	0.6	0.63	0.57	0.06	0.03
Players/Coaches	0.61	0.53	0.57	0.13	0.01	0.01
Players/Arenas	0.01	0.01	0.01	0.23	0.16	0.21
Teams/Arenas	1.0	0.61	0.76	0.89	0.001	0.002
Teams/Coaches	0.93	0.87	0.90	0.43	0.01	0.02
Coaches/Arenas	0.89	0.55	0.68	0.07	0.0	0.0
Medicines/Types	-	0.63	-	-	-	-
Medicines/ Treatments	-	0.56	-	-	-	-
Types/Treatments	-	0.72	-	-	-	-

tion, we would expect to find 10,000× R/Pg true relations. We ran 10 distinct trials of Ling-Seed, each time randomly selecting 10 starting seeds from our truth data, and report our average findings. We note that we only run Ling-Seed for the NBA relations, since we know the full enumerated set and can compare recall for the relations.

We also compare against TextRunner [3], an open information extraction system that finds relations on the Web. TextRunner has a query interface where one side of a relation can be submitted as a query, and the other side of the relation can be returned as a result. For instance, a query of "Kobe Bryant" returns the relation "scores" with the right side of the relation as "Los Angeles Lakers." Therefore, to determine whether TextRunner would have found implicit joins between the selected columns, we query it with one side of the relation and see if the returned relations contain the true relations from our truth data. We must note, however, that TextRunner's relations are built from a small snapshot of 500 million webpages harvested in 2008. Our technique, leverages Google's index for CP and GMI calculations, which is much larger. Also, because TextRunner's snapshot is from 2008, we do not necessarily expect it to be up-to-date on the latest information about the domains. Nonetheless, the accomplishments of TextRunner are substantial, so it presents a worthwhile baseline for comparing the ability to find true relations. Again, we only compare against the NBA data, so we can compare F-measures (which require recall).

The results for the implicit joins are given in Table 5. In this table we see columns of recall, precision, and F-measure for the discovered match pairs that compose our implicit joins, the F-measure for finding the joins using TextRunner, the Ling-Seed average R/Pg metric, and the Ling-Seed variance for the R/Pg metric. Again, we note that we examine the F-measure for the matching pairs found by our method, since this indicates the accuracy of the discovered implicit join.

First, our discovered implicit joins outperform TextRunner for 4 of the 6 relations, and do so with a large margin. For the Teams/Arenas relation, TextRunner outperforms our method because of its low precision, which results from the fact that our list of Arenas is actually a superset of all possible arenas including noise introduced during extraction of the table (the column contains team names and page areas from a set of links), and these noisy values are assigned teams as well. Similarly, for Players/Arenas, most of the players are assigned to the arena "NBA draft," which is a noisy instance included in the column. However, we note that we outperform on a number of attributes, and this is using only the top 5 returned pages for the various taxonomic queries. That is, we did not need to examine a full 500M page corpus to discover the lists that compose the relations, but could do so with taxonomic queries.

The Ling-Seed results are also interesting. First, many of the relations return values for R/Pg that are close to 0. This is because many of the relations (e.g., Coaches/Arenas) do not naturally form as surface patterns in text (this is a hard relation to generalize into a sentence). Based on R/Pg, Ling-Seed needs to examine many thousands of pages to generate the same recall as the GMI-based joins, demonstrating that linking together semi-structured columns has an advantage over textual surface patterns in that it can discover relations that are hard to describe in text. For instance, to generate the same recall as GMI for the Teams and Arenas relation, Ling-Seed must examine 30,000 pages, assuming the best case where it can extract all relations. Further, the variance for R/Pg across the 10 trials is quite high for almost all relations, often times exceeding the average. This large variance, due to which seeds are chosen for bootstrapping, confirms previous results showing certain extractors can be sensitive to their input seeds.

5. RELATED WORK

There is quite a bit of work on turning various Web data into tables of relations. The most popular method is to extract relations from Web text [2, 18, 4, 1, 15, 22]). Other methods construct tables by extracting from other tables (e.g., [11]), lists on the Web (e.g., [9, 14]). Although most of these methods start with seed examples, the key difference between these methods and ours is that each of these approaches aims to generate independent tables representing the relations, whereas our approach also attempts to link the tables together. Therefore, we view our approach as complementary to these methods. In particular, while we leverage AutoWrap which generates tables from arbitrary structured pieces of Web pages (such as lists, tables, etc), we should be able to leverage any of the above methods as the extraction component (if we could generate seeds from our taxonomic queries).

Our work is also similar to integrating resources from the Deep Web via schema matching (e.g., [23, 12, 21]) in that joins are discovered by finding matching columns. However, our work differs in a few crucial ways. First, we rely exclusively on the column instances (since we generally do not have column names), and second, our goal is to integrate together tables that are already materialized, rather than integrating together the schema of the tables. Finally, although there is work on finding complex column matches (e.g., [7]), there is not a notion in schema-matching of "implicit" joins, since the goal of schema matching is to find explicit matches across the schema of existing columns. Therefore, we might be able to leverage some of the previous schema matching work for finding the explicit joins, but implicit joins require different methods.

Our work is also related to the OCTOPUS system [5]. OCTOPUS allows users to create data integration plans on-the-fly by executing commands such as SEARCH (which returns tables from the Web related to your search term), and EXTEND (which finds tables that join with the current user defined table). OCTOPUS is certainly related in spirit in that it materializes data from the Web into tables. However, the most important difference is that we focus on automatically materializing the tables, without the user needing to define the integration plan.

6. CONCLUSIONS

In this work we presented a framework for materializing a database about a domain from disparate sources on the Web. We demonstrated the effectiveness of taxonomic queries for discovering the relevant, disparate sources, and showed that we can discover and utilize relations that go beyond those between columns.

We view this work as the preliminary work for a challenging and novel problem: materializing multi-relational from the Web. There are a number of areas for improvement on interesting subproblems. First, our ColumnScrubber approach is based on a conditional dependency between the values in a column and the domain which it generates. However, there are a number of possible improvements to this approach, such as including domain knowledge from a knowledge base, leveraging more linguistic information, etc.

There is also room for improvement when discovering the joins. One interesting future direction is to leverage the redundancy of certain information in the columns as possible training data for more supervised machine learning approaches to this problem. Also, the ability to discover joins could be modeled as a schema matching problem, and perhaps there are unsupervised approaches from that field which we could leverage to improve this work. Another interesting problem along these lines is discovering more complex relations across tables (such as those from [7]). Therefore, we hope that our approach can serve as a baseline for this new and exciting problem and spur further research in this area.

7. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plaintext collections. In *Proceedings of the ACM International Conference on Digital Libraries*, 2000.

[2] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proc. of IJCAI*, 2007.

[3] M. Banko and O. Etzioni. The tradeoffs between open and traditional relation extraction. In *Proc. of ACL*, 2008.

[4] M. J. Cafarella, D. Downey, S. Soderland, and O. Etzioni. Knowitnow: fast, scalable information extraction from the web. In *Proc. of HLT-EMNLP*, 2005.

[5] M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *Proc. VLDB Endow.*, 2(1):1090–1101, 2009.

[6] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proc. of SIGMOD*, 2002.

[7] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: discovering complex semantic matches between database schemas. In *Proc. of SIGMOD*, 2004.

[8] D. Downey, O. Etzioni, and S. Soderland. A probabilistic model of redundancy in information extraction. In *Proceedings of IJCAI*, 2005.

[9] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. In *Proc. of VLDB*, 2009.

[10] B. C. Gazen and S. Minton. Overview of autofeed: An unsupervised learning system for generating webfeeds. In *Proc. of AAAI*, 2006.

[11] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proc. VLDB Endow.*, 2(1):289–300, 2009.

[12] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *Proc. of SIGMOD*, 2003.

[13] M. Makrehchi and M. S. Kamel. Automatic taxonomy extraction using google and term dependency. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, pages 321–325. IEEE Computer Society, 2007.

[14] M. Michelson and C. A. Knoblock. Exploiting background knowledge to build reference sets for information extraction. In *Proceedings of IJCAI*, 2009.

[15] M. Pasca, D. Lin, J. Bingam, A. Lifchits, and A. Jain. Organizing and searching the world wide web of facts - step one: the one-million fact extraction challenge. In *Proceedings of AAAI*, 2006.

[16] M. Pennacchiotti and P. Pantel. Entity extraction via ensemble semantics. In *In Proc. of EMNLP*, 2009.

[17] P. D. Turney. Mining the Web for synonyms: PMI–IR versus LSA on TOEFL. *Lecture Notes in Computer Science*, 2167:491–503, 2001.

[18] P. D. Turney. Expressing implicit semantic relations without supervision. In *In Proc. of ACL*, 2006.

[19] V. Vyas, P. Pantel, and E. Crestan. Helping editors choose better seed sets for entity expansion. In *Proc. of CIKM*, 2009.

[20] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *Proc. of WWW*, 2003.

[21] J. Wang, J.-R. Wen, F. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *Proc. of VLDB*, 2004.

[22] F. Wu, R. Hoffmann, and D. S. Weld. Information extraction from wikipedia: Moving down the long tail. In *In Proc. of SIGKDD*, 2008.

[23] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proc. of SIGMOD*, 2004.