InfoGather: Entity Augmentation and Attribute Discovery By Holistic Matching with Web Tables

Mohamed Yakout[*]
Purdue University
myakout@cs.purdue.edu

Kris Ganjam
Microsoft Research
krisgan@microsoft.com

Kaushik Chakrabarti
Microsoft Research
kaushik@microsoft.com

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

ABSTRACT

The Web contains a vast corpus of HTML tables, specifically entity-attribute tables. We present three core operations, namely entity augmentation by attribute name, entity augmentation by example and attribute discovery, that are useful for "information gathering" tasks (e.g., researching for products or stocks). We propose to use web table corpus to perform them automatically. We require the operations to have high precision and coverage, have fast (ideally interactive) response times and be applicable to any arbitrary domain of entities. The naive approach that attempts to directly match the user input with the web tables suffers from poor precision and coverage.

Our key insight is that we can achieve much higher precision and coverage by considering indirectly matching tables in addition to the directly matching ones. The challenge is to be robust to spuriously matched tables: we address it by developing a holistic matching framework based on topic sensitive pagerank and an augmentation framework that aggregates predictions from multiple matched tables. We propose a novel architecture that leverages preprocessing in MapReduce to achieve extremely fast response times at query time. Our experiments on real-life datasets and 573M web tables show that our approach has (i) significantly higher precision and coverage and (ii) four orders of magnitude faster response times compared with the state-of-the-art approach.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services

General Terms

Algorithms, Design, Experimentation

1. INTRODUCTION

The Web contains a vast corpus of HTML tables. In this paper, we focus on one class of HTML tables: entity-attribute tables (also referred to as relational tables [5, 4] and 2-dimensional tables [20]). Such a table contains values of multiple entities on multiple attributes, each row corresponding to an entity and each column corresponding to an attribute. Cafarella et. al. reported 154M such tables from a snapshot of Google's crawl in 2008; we extracted 573M such tables from a recent crawl of Microsoft Bing search engine. Henceforth, we refer to such tables as simply web tables.

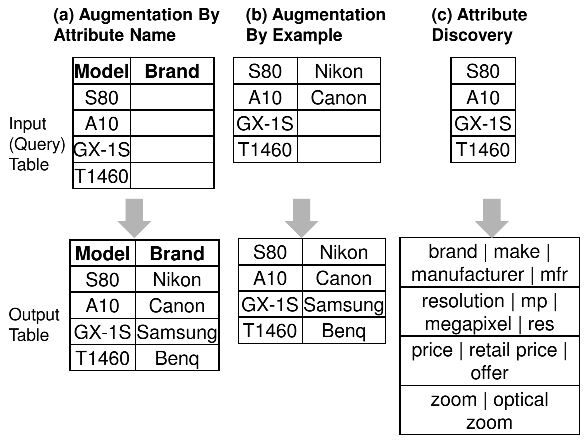

(a) Augmentation By Attribute Name
(b) Augmentation By Example
(c) Attribute Discovery
Input (Query) Table
Output Table
Model
Brand
S80
A10
GX-1S
T1460
Nikon
Canon
Samsung
Benq
brand | make | manufacturer | mfr
resolution | mp | megapixel | res
price | retail price | offer
zoom | optical zoom

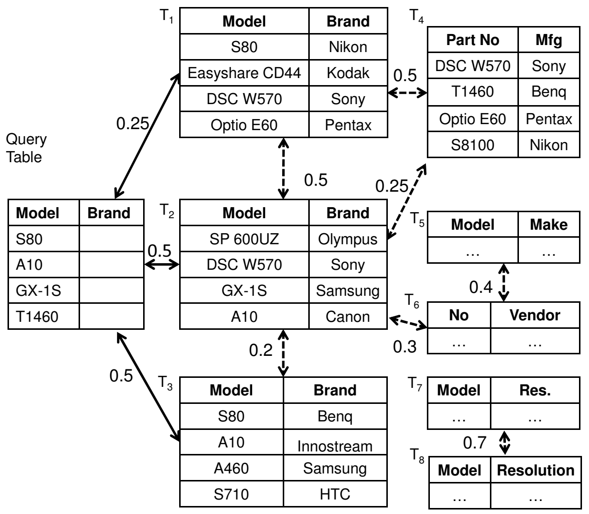
Figure 1: APIs of the 3 core operations

Consider a user researching for products or stocks or an analyst performing competitor analysis. One of the most labor-intensive subtasks of such tasks is gathering information about the "entities" of interest. We identify two such subtasks: finding the values of attributes of one or more entities and finding the relevant attributes of an entity type. *We propose to automate them using the extracted web tables.* We formalize these subtasks using the following three core operations:

• **Augmentation By Attribute Name (ABA)**: Consider a user researching digital cameras. She collects the names of models she is interested in into a spreadsheet (e.g., Excel). She would like to find their values on various attributes such as brand, resolution, price and optical zoom based on which she can decide which one to buy. Here the entities which the user wants to gather information for are the camera models; we henceforth refer to them simply as entities. We refer to this operation as *augmentation by attribute name* and these attributes as *augmenting attributes*. This was originally proposed as an operator, called EXTEND, in [4]. Figure 1(a) shows example input and output for this operation applied to camera model entities with one augmenting attribute (brand). Such augmentation would be difficult to perform using an enterprise database or an ontology because the entities can be *from any arbitrary domain*. Today, users try to manually find the web sources containing this information and assemble the values. Assuming that this informa-

[*]Work done while visiting Microsoft Research.

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.
Copyright 2012 ACM 978-1-4503-1247-9/12/05 ...$10.00.



Model	Brand
S80	Nikon
Easyshare CD44	Kodak
DSC W570	Sony
Optio E60	Pentax

Part No	Mfg
DSC W570	Sony
T1460	Benq
Optio E60	Pentax
S8100	Nikon

Model	Brand
S80	
A10	
GX-1S	
T1460	

Model	Brand
SP 600UZ	Olympus
DSC W570	Sony
GX-1S	Samsung
A10	Canon

Model	Make
...	...

No	Vendor
...	...

Model	Brand
S80	Benq
A10	Innostream
A460	Samsung
S710	HTC

Model	Res.
...	...

Model	Resolution
...	...

Figure 2: ABA operation using web tables

tion is available, albeit scattered, in various web tables, we can save a lot of time and effort if we can perform this operation automatically.

• **Augmentation by Example (ABE)**: A variant of ABA is to provide the *values on the augmenting attribute(s) for a few entities* instead of providing the name of the augmenting attribute(s). We refer to this operation as *augmentation by example*. Figure 1(b) shows example input and output for this operation applied to camera model entities and one augmenting attribute (brand).

• **Discovery Of Important Attributes (AD)**: Often, the user may not know enough about the domain; in such cases, she would like to know the most important attributes for the given set of entities. She can then select the ones that matter the most to her and request augmentation for those. Figure 1(c) shows example input and output for this operation applied to camera model entities. If we can use the web tables to discover the relevant attributes automatically, we can save the user's time and effort in trying to discover them manually.

The requirements for these core operations are: (i) *high precision* ($\frac{\#corraug}{\#aug}$) and *high coverage* ($\frac{\#aug}{\#entity}$) where $\#corraug$, $\#aug$ and $\#entity$ denote the number of entities correctly augmented, the number of entities augmented and the number of entities, respectively. (ii) *fast (ideally interactive) response times* and (iii) applicability to entities of any *arbitrary domain*. The focus of the paper is to perform these operations using web tables such that the above requirements are satisfied.

Baseline Technique: We present the baseline technique and our insights in the context of the ABA operation; they apply to all the core operations as discussed in Section 5. For simplicity, we consider only one augmenting attribute. As shown in Figure 1(a), the input can be viewed as a binary relation with the first column corresponding to the entity name and the second corresponding to the augmenting attribute. The first column is populated with the names of entities to be augmented while the second column is empty. We refer to this table as the *query table* (or simply the *query*). The baseline technique first identifies web tables that semantically "matches" with the query table using schema matching techniques (we consider simple 1:1 mappings only) [2]. Subsequently, we look each entity up in those web tables to obtain its value on the augmenting attribute. The state-of-the-art entity augmentation technique, namely Octopus, implements a variant of this technique using the search engine API [4].

EXAMPLE 1. *Consider the query table Q in Figure 2. For simplicity, assume that, like the query table, all the web tables are entity-attribute binary (EAB) relations with the first column corresponding to the entity name and the second to an attribute of the entity. Note that for both the query table and web table the first column is approximately the key column. Using traditional schema matching techniques, a web table matches Q iff (i) data values in its first column overlaps with those in the first column of Q and (ii) name of its second column is identical to that of the augmenting attribute. We refer to such matches as "direct matches" and the approach as "direct match approach" (DMA). In Figure 2, only web tables T_1, T_2 and T_3 directly matches with Q (shown using solid arrows). A score can be associated with each direct match based on the degree of value overlap and degree of column name match; such scores are shown in Figure 2. We then look the entities up in T_1, T_2 and T_3. For $S80$, both T_1 and T_3 contain it but the values are different (Nikon and Benq respectively). We can either choose arbitrarily or choose the value from the web table with the higher score, i.e., Benq from T_3. For $A10$, we can choose either Canon from T_2 or Innostream from T_3 (they have equal scores). For $GX-1S$, we get Samsung. We fail to augment $T1460$ as none of the matched tables contains that entity.*

DMA suffers from two problems:
(i) **Low precision**: In the above example, T_3 contains models and brands of cell phones, not cameras. The names of some of the cell phone models in T_3 are identical to those of the camera models in the query table, hence, T_3 get a high score. This results in 2 (out of 3) wrong augmentations: $S80$ and $A10$ (assuming we choose Innostream from T_3 for $A10$). Hence, the precision is 33%. Such ambiguity of entity names exist in all domains as validated by our experiments. Note that this can mitigated by raising the "matching threshold" but this leads to poor coverage.
(ii) **Low coverage**: In the above example, we fail to augment $T1460$. Hence, the coverage is 75%. This number is much lower in practice, especially for tail domains. For example, the Octopus system (which implements a variant of DMA) reports a coverage of 33%. This primarily happens because tables that can provide the desired values either do not have column names or use different column name as the augmenting attribute name provided by the user.

One way to address the coverage issue is to use synonyms of the augmenting attribute [18, 16]. Traditionally, schema-matchers have used hand-crafted synonyms; this is not feasible in our setting where the entities can be from any arbitrary domain. Automatically generating attribute synonyms for arbitrary domains, as proposed in [5], typically result in poor quality synonyms. Our experiments show that these are unusable without manual intervention.

Main Insights and Contributions: Our key insight is that many tables *indirectly match* the query table, i.e., via other web tables. These tables, in conjunction with the directly matching ones, can improve both coverage and precision. We first consider coverage. Observe that in Figure 2, table T_4 contains the desired attribute value of $T1460$ (Benq) but we cannot "reach" it using direct match. Using schema matching techniques, we can find that T_4 matches with T_1 (i.e., there is 1:1 mapping between the two attributes of the two relations) as well as T_2 (as it has 2 records in common with T_1 and 1 in common with T_2). Such schema matches among web tables are denoted by dashed arrows; each such match has a score representing the degree of match. Since T_1 and/or T_2 (approximately) matches with Q (using DMA) and T_4 (approximately) matches with T_1 and T_2 (using schema matching among web tables), we can conclude T_4 (approximately) matches with Q. We refer to T_4

as an *indirectly matching table*; using it, we can correctly augment $T1460$. This improves coverage from 75% to 100%.

Many of the indirectly matching tables are spurious matches; using these tables to predict values would result in wrong predictions. The challenge is to be robust to such spurious matches. We address this challenge in two ways. First, we perform *holistic matching*. We observe that *truly matching tables match with each other and with the directly matching tables, either directly or indirectly while spurious ones do not*. For example, T_1, T_2 and T_4 match directly with each other while T_4 only matches weakly with T_2. If we compute the overall matching score of a table by aggregating the direct match as well as *all* indirect matches, the true matching tables will get higher scores; we refer to this as *holistic matching*[1]. In the above example, T_1, T_2 and T_4 will get higher score compared with T_3; this leads to correct augmentations for $S80$ and $A10$ resulting in a precision of 100% (up from 33%). Second, for each entity, we obtain predictions from *multiple* matched tables and *"aggregate"* them; we then select the "top" one (or k) value(s) as the final predicted value(s).

This gives rise to additional technical challenges: (i) We need to compute schema matches *between pairs of web tables*; we refer to this as the *schema matching among web tables (SMW) graph* . How do we build an accurate SMW graph over $573M \times 573M$ pairs of tables? (ii) How do we model the holistic matching? The model should take into account the scores associated with the edges in the SMW graph as well as those associated with the direct matches. (iii) How do we augment the entities efficiently at query time?

We have built the INFOGATHER system based on the above insights. Our contributions can be summarized as follows:

- We develop a novel holistic matching framework based on *topic sensitive pagerank* (TSP) over the SMW graph (Section 2). We argue that by considering the query table as a topic and web tables as documents, we can efficiently model the holistic matching as TSP (details are in Section 2.4). To the best of our knowledge, this is the first paper to propose holistic matching with web tables.
- We present a novel architecture for the INFOGATHER system that leverages preprocessing in MapReduce to achieve extremely fast (interactive) response times at query time. Our architecture overcomes the limitations of the prior architecture (viz., Octopus) that uses the search API: its inability to perform indirect/holistic matches and its high response times (Section 3).
- We present a machine learning-based technique for building the SMW graph. Our key insight is that the text surrounding the web tables is important in determining whether two web tables match or not. We propose a novel set of features that leverage this insight. Furthermore, we develop MapReduce techniques to compute these (pairwise) features that scales to 573M tables. Finally, we propose a novel approach to automatically generate training data for this learning task; this liberates the system designer for manually producing labeled data (Section 4).
- We describe how our holistic matching framework can benefit the other core operations, namely augmentation-by-example and attribute-discovery (Section 5).
- We perform extensive experiments on six real-life query datasets and 573M web tables (Section 7). Our experiments show that our holistic matching framework has significantly higher precision and coverage compared with both direct matching approach as well as the state-of-the-art entity augmentation technique, Octopus. Furthermore, our technique have four orders of magnitude faster response times compared with Octopus.

[1] This is different from holistic matching proposed in [12] as discussed in Section 8.

2. HOLISTIC MATCHING FRAMEWORK

We present the data model, the general augmentation framework and its two specializations: direct matching and holistic matching frameworks. We present them in the context of ABA operation. How we leverage these frameworks for the other core operations (ABE and AD) are discussed in Section 5.

2.1 Data Model

For the purpose of exposition, we assume that the query table is an entity-attribute binary (EAB) relation, i.e., a query table Q is of the form $Q(K, A)$, where K denotes the entity name attribute and A is the augmenting attribute. Since $Q.K$ is approximately the key attribute, we refer to it as the query table key attribute and the entities as keys. The key column is populated while the augmenting attribute column is empty. An example of the query table satisfying the above properties is shown in Figure 2.

We assume that all web tables are EAB relations as well. For each web table $T \in \mathcal{T}$, we have the following: (1) the EAB relation $T_R(K, B)$ where K denotes the entity name attribute and B is an attribute of the entity; as in the query table, since $T.K$ is approximately the key attribute, we refer to it as the web table key attribute, (2) the url T_U of the web page from which it was extracted, and (3) its context T_C (i.e., the text surrounding the table) in the web page from which it was extracted. For simplicity, we denote $T_R(K, B)$ as $T(K, B)$ when it is clear from the context. Figure 2 shows four web tables (T_1,T_2,T_3,T_4) satisfying the EAB property.

The ABA problem can be stated as follows.

DEFINITION 1. *Augmentation By Attribute Name (ABA): Given a query table $Q(K, A)$ and a set of web tables $\langle T(K, B), T_U, T_C \rangle \in \mathcal{T}$, predict the value of each query record $q \in Q$ on attribute A.*

In practice, not all web tables are EAB relations; we show how our framework can be used for general, n-ary web tables in Section 6. Furthermore, the query table can have more than one augmenting attribute; we assume that those attributes are independent and perform predictions for one attribute at a time.

2.2 General Augmentation Framework

Our augmentation framework consists of two main steps: First, identify web tables that "match" with the query table. Second, use each matched web table to provide value predictions for the particular keys that happen to overlap between the query and the web table; then aggregate these predictions and pick the top value as the final predicted value. We describe the two steps in further detail.

- **Identify Matching Tables**: Intuitively, a web table $T(K, B)$ matches the query table $Q(K, A)$ if $Q.K$ and $T.K$ refer to the same type of entities and $Q.A$ and $Q.B$ refers to the same attribute of the entities. In this paper, we consider simple 1:1 mappings only. Each web table T will be assigned a score $S(Q, T)$ representing the matching score to the query table Q. Since Q is fixed, we omit Q from the notation and simply denote it as $S(T)$. There are many ways to obtain the matching scores between the query table and web tables; we consider two such ways in the next two subsections.
- **Predict Values**: For each record $q \in Q$, we predict the value $q[Q.A]$ of record q on attribute $Q.A$ from the matching web tables. This is done by joining the query table $Q(K, A)$ with each matched web table $T(K, B)$ on the key attribute K. If there exists a record $t \in T$ such that $q[Q.K] \approx t[T.K]$ (where $\approx$ denotes either exact or approximately equality of values), then we say that the web table T

predicted the value $v = t[T.B]$ for $q[Q.A]$ with a prediction score $S_T(v) = S(T)$ and return $(v, S_T(v))$.

After processing all the matched tables, we end up with a set $\mathcal{P}_q = \{(x_1, S_{T_1}(x_1)), (x_2, S_{T_2}(x_2)), \dots\}$ of predicted values for $q[Q.A]$ along with their corresponding prediction scores. We then perform fuzzy grouping [7] on the x_i's to get the groups $\mathcal{G}_q = \{g_1, g_2, \dots\}$, such that, $\forall x_i \in g_k, x_i \approx v_k$, where v_k is the centroid or the representative of group g_k. We compute the final prediction score for each group representative v by aggregating the predictions scores of the group's members as follows:

$$S(v) = \underset{(x_i, S_{T_i}(x_i)) \in \mathcal{P}_q | x_i \approx v}{\mathcal{F}} S_{T_i}(x_i) \tag{1}$$

where $\mathcal{F}$ is an aggregation function. Any aggregation function such as sum or max can be used in this framework.

The final predicted value for $q[Q.A]$ is the one with the highest final prediction score:

$$q[Q.A] = \underset{v}{\operatorname{argmax}} \; S(v) \tag{2}$$

If the goal is to augment k values for an entity on an attribute (e.g., the entity is a musical band and the goal is to augment it with all its albums), we simply pick the k with the highest final prediction score.

EXAMPLE 2. *Consider the example in Figure 2. Using the table matching scores shown, for the query record* $S80$, $\mathcal{P}_q = \{(Nikon, 0.25), (Benq, 0.5)\}$ *(predicted by tables* T_1 *and* T_3 *respectively). The final predicted values are* $Nikon$ *and* $Benq$ *with scores* 0.25 *and* 0.5 *respectively, so the predicted value is* $Benq$.

2.3 Direct Match Approach

One way to compute the matching web tables and their scores is the direct match approach (DMA) discussed in Section 1. The prediction step is identical to that in the general augmentation framework. Using traditional schema matching techniques, DMA considers a web table T to match with the query table Q iff (i) data values in $T.K$ overlaps with those $Q.K$ and (ii) the attribute name $T.B$ matches $Q.A$ (denoted by $T.B \approx Q.A$). DMA computes the matching score $S(T)$ between Q and T, denoted as $S_{DMA}(T)$, as follows:

$$S_{DMA}(T) = \begin{cases} \frac{|T \cap_K Q|}{\min(|Q|,|T|)} & \text{if } Q.A \approx T.B \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

where $|T \cap_K Q| = |\{t \mid t \in T \;\&\; \exists\, q \in Q \; s.t. \; t[T.K] \approx q[Q.K]\}|$. For example, in Figure 2, the scores for T_1, T_2 and T_3 are $\frac{1}{4}$, $\frac{2}{4}$ and $\frac{2}{4}$ respectively as they have 1, 2 and 2 matching keys respectively, $\min(|Q|, |T|) = 4$ and $Q.A \approx T.B$; the score for T_4 is 0 because $Q.A \not\approx T.B$.

2.4 Holistic Match Approach

To overcome the limitations of the DMA approach as outlined in Section 1, we study the holistic approach to compute matching tables and their scores. The prediction step remains the same as above. We model the holistic matching using TSP. We start by reviewing the definitions of personalized pagerank (PPR) and TSP; and then make the link to our problem in Section 2.4.2.

2.4.1 *Preliminaries: Personalized and Topic Sensitive Pagerank*

Consider a weighted, directed graph $G(V, E)$. We denote the weight on an edge $(u, v) \in E$ with $\alpha_{u,v}$. Pagerank is the stationary distribution of a random walk on G that at each step, with a probability ϵ, usually called the teleport probability, jumps to a random node, and with probability $(1 - \epsilon)$ follows a random outgoing edge from the current node. Personalized Pagerank (PPR) is the same as Pagerank, except all the random jumps are done back to the same node, denoted as the "source" node, for which we are personalizing the Pagerank.

Formally, the PPR of a node v, with respect to the source node u, denoted by $\pi_u(v)$, is defined as the solution of the following equation:

$$\pi_u(v) = \epsilon\delta_u(v) + (1 - \epsilon) \sum_{\{w | (w,v) \in E\}} \pi_u(w)\alpha_{w,v} \tag{4}$$

where $\delta_u(v) = 1$ iff $u = v$, and 0 otherwise. The PPR values $\pi_u(v)$ of all nodes $v \in V$ with respect to u is referred to as the PPR vector of u.

A "topic" is defined as a preference vector $\vec{\beta}$ inducing a probability distribution over V. We denote the value of $\vec{\beta}$ for node $v \in V$ as β_v. Topic sensitive pagerank (TSP) is the same as Pagerank, except all the random jumps are done back to one of the nodes u with $\beta_u > 0$, chosen with probability β_u. Formally, the TSP of a node v for a topic $\vec{\beta}$ is defined as the solution of the following equation [11]:

$$\pi_{\vec{\beta}}(v) = \epsilon\vec{\beta} + (1 - \epsilon) \sum_{\{w | (w,v) \in E\}} \pi_{\vec{\beta}}(w)\alpha_{w,v} \tag{5}$$

2.4.2 *Modeling Holistic Matching using TSP*

First, we draw the connection between the PPR of a node with respect to a source node and the holistic match between two web tables. Then, we show how the holistic matching between the query table and a web table can be modeled with TSP.

Consider two nodes u and v of any weighted, directed graph $G(V, E)$. *The PPR* $\pi_u(v)$ *of* v *with respect to* u *represents the holistic relationship of* v *to* u *where* E *represents the direct, pairwise relationships*, i.e., it considers all the paths, direct as well as indirect, from u to v and "aggregates" their scores to compute the overall score. PPR has been applied to different types of relationships. When the direct, pairwise relationships are hyperlinks between web pages, $\pi_u(v)$ is the holistic importance conferral (via hyperlinking) of v from u; when the direct, pairwise relationships are direct friendships in a social network, $\pi_u(v)$ is the holistic friendship of v from u.

In this paper, we propose to use PPR to compute the holistic semantic match between two web tables. Therefore, we build the weighted graph $G(V, E)$, where each node $v \in V$ corresponds to a web table and each edge $(u, v) \in E$ represents the direct pairwise match (using schema matching) between the web tables corresponding to u and v. Each edge $(u, v) \in E$ has a weight $\alpha_{u,v}$ which represents the degree of match between the web tables u and v (provided by the schema matching technique). We discuss building this graph and computing the weights in detail in Section 4.1. We refer to this graph as the *schema matching graph among web tables* (SMW graph). Thus, the PPR of $\pi_u(v)$ of v with respect to u over the SMW graph models the *holistic semantic match* of v to u.

Suppose the query table Q is identical to a web table corresponding to the node u, then the holistic match score $S_{Hol}(T)$ between Q and the web table T is $\pi_u(v)$, where v is the node corresponding to T. However, the query table Q is typically not identical to a web table. In this case, how can we model the holistic match of a web table T to Q? Our key insight is to consider Q as a "topic" and model the match as the TSP of the node v corresponding to T to the topic. In the web context where the relationship is that of importance conferral, the most important pages on a topic are used to model the topic (the ones included under that topic in Open Directory Project); in our context where the relationship is semantic match, the top matching tables should be used to model the topic of Q. We use the set of web tables $\mathcal{S}$ (referred to as *seed tables*) that directly matches with Q, i.e., $\mathcal{S} = \{T | S_{DMA}(T) > 0\}$ to model it. Furthermore, we use the direct matching scores $S_{DMA}(T) | T \in \mathcal{S}$ as the preference values $\vec{\beta}$:

$$\beta_v = \begin{cases} \frac{S_{DMA}(T)}{\sum_{T \in S} S_{DMA}(T)} \text{if } T \in \mathcal{S} \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

where v corresponds to T. For example, β_v are $\frac{0.25}{1.25}$, $\frac{0.5}{1.25}$ and $\frac{0.5}{1.25}$ for T_1, T_2 and T_3 respectively and 0 for all other tables. Just like the TSP score of web page representing the holistically computed importance of a page to the topic, $\pi_{\vec{\beta}}(v)$ over the SMW graph models the holistic semantic match of v to Q. Thus, we propose to use $S_{Hol}(T) = \pi_{\vec{\beta}}(v)$ where v corresponds to T.

3. SYSTEM ARCHITECTURE

Suppose the SMW graph G has been built upfront. The naive way to compute the holistic matching score $S_{Hol}(T)$ for each web table is to run the TSP computation algorithm over G at augmentation time. This results in prohibitively high response times. We leverage the following result to overcome this problem:

THEOREM 1. *(Linearity [11]) For any preference vector $\vec{\beta}$, the following equality holds:*

$$\pi_{\vec{\beta}}(v) = \sum_{u \in V} \beta_u \times \pi_u(v) \tag{7}$$

If we can precompute the PPR $\pi_u(v)$ of every node v with respect to every other node u (referred to as Full Personalize Pagerank (FPPR) computation) in the SMW graph, we can compute the holistic matching score *for any query table* $\pi_{\vec{\beta}}(v)$ efficiently using Eq. 7. This leads to very fast response times at query time.

INFOGATHER architecture has two components as shown in Figure 3. The first component performs offline preprocessing for the web crawl to extract the web tables, build the SMW graph and compute the FPPR. For all these offline steps, our techniques need to scale to hundreds of millions of tables. We propose to leverage the MapReduce framework for this purpose. The second component concerns the query time processing, where we compute the TSP scores for the web tables and aggregate the predictions from the web tables. In the following, we give more details about each component:

Preprocessing: There are five main processing steps in this component:

• P1: *Extract the HTML web tables* from the web crawl and use a classifier to distinguish the entity attribute tables from the other types of web tables, (e.g., formatting tables, attribute value tables, etc.). Our approach is similar to the one proposed in [6]; we do not discuss this step further as it is not the focus of the paper.

• P2: *Index the web tables* to facilitate faster identification of the seed tables. We use three indexes: (i) An index on the web tables' key attribute values (WIK). Given a query table Q, WIK(Q) returns the set of web tables that overlaps with Q on at least one of the keys. (ii) An index for the web tables complete records (that is key and value combined) (WIKV). WIKV(Q) returns the set of web tables that contain at least one record from Q. (iii) An index on the web tables attributes names (WIA), such that, WIA(Q) returns the set of web tables $\{T | T.B \approx Q.A\}$

• P3: *Build the SMW graph* based on schema matching techniques as we describe in Section 4.1.

• P4: *Compute the FPPR* and store the PPR vector for each web

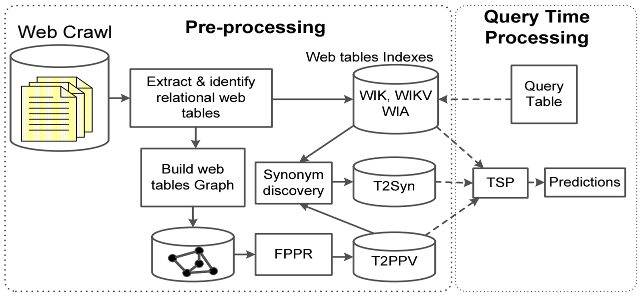


Figure 3: INFOGATHER System Architecture

table (we store only the non-zero entries). We refer to this as the *T2PPV index*. For any web table T, T2PPV(T) returns the PPR vector of T. We discuss the technique we use to compute the FPPR in Section 4.2.

• P5: *Discover the synonyms* of attribute B for each web table $T(K, B)$. We give the details of this step while discussing the attribute discovery operation in Section 5.3. We refer to this as the *T2Syn index*. For any web table T, T2Syn(T) returns the synonyms of attribute B of table T.

The indexes (WIK, WIKV, WIA, T2PPV and T2Syn) may either be disk-resident or reside in memory for faster access.

Query Time Processing: The query time processing can be abstracted in three main steps. The details of each step depends on the operation. We provide those details for each operation in Section 5.

• Q1: *Identify the seed tables:* We leverage the WIK, WIKV and WIA indexes to identify the seed tables and compute their DMA scores.

• Q2: *Compute the TSP scores:* We compute the preference vector $\vec{\beta}$ by plugging the DMA matching scores in Eq. 6. According to Theorem 1, we can use $\vec{\beta}$ and the stored PPR vectors of each table to compute the TSP score for each web table. Note that only the seed tables have non-zero entries in $\vec{\beta}$. Accordingly, we need to retrieve the PPR vectors of only the seed tables using the T2PPV index. Furthermore, we do not need to compute TSP scores for all web tables in the retrieved PPR vectors. We need to compute it only for the tables that could be used in the aggregation step: the one that have at least one key overlapping with the query table. We refer to them as *relevant tables*. These can be identified efficiently by invoking WIK(Q). These two optimizations are important to compute the TSP scores efficiently.

• Q3: *Aggregate and select values:* In this step, we collect the predictions provided by the relevant web tables T along with the scores $S_{Hol}(T)$. The predictions are then processed, the scores are aggregated and the final predictions are selected according to the operation.

4. BUILDING THE SMW GRAPH AND COMPUTING FPPR

This section discusses the major preprocessing steps of the web tables, namely, building the SMW graph (P3) and computing the FPPR (P4). The preprocessing step (P5) is discussed in Section 5 while discussing the AD operation.

4.1 Building the SMW Graph

First, we give details on how we match a pair of web tables and then address the scalability challenges in building the SMW graph.

4.1.1 Matching Web Tables

In the SMW graph, there is an edge between a pair $(T(K, B),$ $T'(K', B'))$ of web tables if T *matches* with T', i.e., $T.K$ and

$T.K'$ refer to the same type of entities and $T.B$ and $T'.B'$ refers to the same attribute of those entities. Our problem can be formally stated as follows:

DEFINITION 2. *Pairwise web tables matching problem: Given two web tables $\langle T(K, B), T_U, T_C\rangle$ and $\langle T'(K', B'), T'_U, T'_C\rangle$, determine whether T matches with T' and compute the score of the mapping $T.K \rightarrow T'.K'$, $T.B \rightarrow T'.B'$.*

In schema matching [18, 2], the problem of matching two schemas S and S' is normally framed as follows: Given the two schemas, for each attribute A of S, find the best corresponding attribute A' of S', possibly with an associated matching score.

The problems are similar enough so that the techniques used in standard schema matching problem can be used for ours as well. Schema matching techniques first identify information about each element of each schema that is relevant to discovering matches. For each pair of elements, one from each schema, they compute a set of "feature scores" where each feature score represent a match between the pair on a different aspect. Finally, they combine those feature scores into a single score based on which they decide whether the element pair matches or not. The combination module can either use machine learning-based techniques or non-learning methods [15, 8]; we use machine learning-based techniques in this paper.

Traditionally, the focus is on schema level features (e.g., attribute names matching) and instance level features (e.g., attribute data values matching). Specifically for web tables, [4] suggested to use two specific features: (i) the average columns width similarity and (ii) the similarity of the text of the table content without considering the columns and rows structure.

But for web tables it may not be sufficient to rely only on these traditional schema and instance level information. For example, consider tables T_2 and T_3 in Figure 2. At the schema level, they both share the same column names, and moreover, at the instance level both share the Model `A10` and share the Brand `Samsung`. Despite all these similarities, these web tables are not a match, because T_2 is about cameras and T_3 is about cell phones. On the other hand, consider web tables T_2 and T_4. They neither share schema level nor instance level similarities. However, both T_2 and T_4 contain digital camera models with their brands and should have high matching score. Furthermore, many web tables do not have column names [19]; this further exacerbates the problem.

Our main insight is that there is additional information about the web tables that can be leveraged to overcome the above limitations. We propose 4 novel feature scores based on this insight:

• **Context similarity:** The *context* or the text around the web table in the web page provides valuable information about the table. Suppose, the context for T_3 is "Mobile phones or cell phones, ranging from ...", while that for T_2 and T_4 are "Ready made correction data for cameras and lenses" and "Camera Bags Compatibility List" respectively. This indicates that T_2 and T_4 are probably about cameras while T_3 about phones. Clearly, sharing the term 'cameras' indicates that similarity between T_2 and T_4. We capture this intuition using a context similarity feature which is computed using the tf-idf cosine similarity of the text around the table.

• **Table-to-Context similarity:** The context of a table may contain keywords that overlap with values inside another web table. This provides evidence that the web pages containing the tables are about similar subjects, and hence, the tables may be about similar subjects as well. We capture this intuition using table-to-context similarity feature, which is computed using the tf-idf cosine similarity of the text around the first table and the text inside the second table.

Table 1: Web tables matching features as documents.

Feature name	Document
Context	Terms in the text around the web table with idf weights
Table-to-Context	The table content as text and context text with idf weights
URL	The terms in the URL with idf weight computed from all URL set
Tuples	All the distinct table rows (or key-value pairs) form terms of a document with equal weights
Attributes name	The terms mentioned in the column names with equal weights
Column values	All the distinct values in a column form terms of a document with equal weights
Table-to-Table	The table content as text with idf weights

• **URL similarity:** The URL of the web page containing the table can help in matching with another table. Sometimes, a web site lists the records from the same original large table in several web pages. For example a web site may list the movies in several web pages by year, by first letter, etc. In this case, matching the URLs of the web pages is a good signal in the matching of the web tables. We capture this intuition using a URL similarity feature, computed using cosine similarity of the URL terms.

• **Tuples similarity:** The web tables that we consider are EAB relations and the correspondences between the attributes are frozen ($T.K \rightarrow T'.K'$ and $T.B \rightarrow T'.B'$); we just need to determine the strength of the correspondence. Hence, the number of tuples that overlaps between the two tables will be a strong evidence to decide upon the tables matching. Note that this is different from the instance level feature, which consider the data values of each attribute individually.

We use the above similarities as features in a classification model. Given the features, the model predicts the match between two tables with a probability, which is used as the weight on the edge between them. The set of features include the newly proposed features, namely, (1) Context similarity, (2) Table-to-Context similarity, (3) URL similarity, and (4) Tuples similarity; in addition to the traditional schema and instance level features, namely, (5) attribute names similarity, (6) column values similarity, (7) Table-to-Table similarity as a bag of words, (8) columns widths similarity.

There are two major challenges in building the SMW graph: (i) computing the pairwise features that scales to hundreds of millions of tables and, (ii) getting labeled pairs of web tables to train a classifier. We address these challenges in the following two subsections.

4.1.2 Scalable Computation of Pairwise Features

Note that we are computing these features for 573M $\times$ 573M web table pairs and, obviously, we cannot do the cross product computation. Our key insight here is that, for each of the mentioned features, the web table can be considered as a bag of words (or a document). We can then leverage scalable techniques for computing pairwise document similarities over a large document collection. Table 1 describes the mapping of a web table to a document for each feature.

We leverage the technique described in [9] to compute the document similarity matrix of a large document set using MapReduce. The technique can be summarized as follows: Each document d contains a set of terms and can be represented as a vector W_d of term weights $w_{t,d}$. The similarity between two documents is the inner product of the term weights as $sim(d_1, d_2) = \sum_{t \in d_1 \cup d_2} w_{t,d_1} \cdot w_{w,d_2}$. The key observation here is that a term t will contribute to the similarity of two documents d_1, d_2 iff $t \in d_1$

and $t \in d_2$. If we have an inverted index I, we can easily get the documents $I(t)$ that contain a particular term t. For each pair of document $\langle d_i, d_j \rangle \in I(t) \times I(t)$, $sim(d_i, d_j)$ is incremented by $(w_{t,d_i} \cdot w_{t,d_j})$. By processing all the terms we have computed the entire similarity matrix without the expensive cross-product computations.

This can be implemented directly as two MapReduce tasks: (1) *Indexing*: The mapper processes each document d and emits for each term $t \in d$ (key = t, value = $(d, w_{t,d})$). The reducer outputs the term as the key and the list of documents containing that key (key = t, value =$I(t)$). (2) *Similarity computation*: The mapper processes each term with its list of documents, $(t, I(t))$, and emits for each pair of documents $\langle d_i, d_j \rangle \in I(t) \times I(t)$ and $i < j$ (key = $\langle d_i, d_j \rangle$, value = $w_{t,d_i} \cdot w_{t,d_j}$). Finally, the reducer does the summation to output the $sim(d_i, d_j)$ (key= $\langle d_i, d_j \rangle$, value=$sim(d_i, d_j) = \sum_{t \in (d_i \cap d_j)} w_{t,d_i} \cdot w_{t,d_j}$). For more efficiency, a df-cut notion is used to eliminate terms with high document frequency [9].

4.1.3 *Getting labeled pairs of web tables:*

We mentioned earlier that we rely on a classification model to get the matching score of two tables given their similarity features vector. The challenge here is to obtain labeled examples to train the classifier. One way is to use a human to manually label a random set of pairs of web tables. However, this is going to be painful and time consuming. We propose an automatic way to obtain labeled pairs of web tables.

To label a pair of web tables (T, T') as a positive example, our hypothesis is that T and T' may not have records in common, but a third web table T'' have some records in common with T and T' individually (we call it a *labeling web table*). For example, consider tables T_2 and T_4 in Figure 2. T_1 is found to be a labeling web table for them. T_1 overlaps with T_2 on one record (`DSC W570, Sony`), as well as, it overlaps with T_4 on one record (`Optio E60, Pentax`).

We formalize our hypothesis as follows: A pair of tables $T_i(K_i, B_i)$ and $T_j(K_j, B_j)$ is a true example pair, if $\exists$ a web table $T_L(K_L, B_L)$ (labeling web table) such that (i) the set of overlapping records $|T_L \cap T_i| \geq \theta$ and $|T_L \cap T_j| \geq \theta$, and (ii) for each record $t_L \in T_L$ and $\exists$ record $t_i \in T_i(t_j \in T_j)$, s. t., if $t_L[K_L] = t_i[K_i](t_L[K_L] = t_j[K_j])$, then $t_L[B_L] = t_i[B_i](t_L[B_L] = t_j[B_j])$. The second condition guarantees that if T_i shares a key with T_L then, the value of the other attribute must match. If we do not find such labeling web table, then the web table pair is considered as a negative example.

It may come to mind that the web table labeling approach can be used to generate all the pairwise semantic matches to build the SMW graph, but this is too expensive to be done for 573M × 573M pairs of web tables. However, using the labeling web table approach to generate a few thousand examples to train a classifier is feasible.

4.2 Computing FPPR on SMW Graph

Once the SMW graph is constructed, we compute the full personalized pagerank matrix. There are two broad approaches to compute personalized pagerank. The first approach is to use linear algebraic techniques, such as Power Iteration [17]. The other approach is Monte Carlo, where the basic idea is to approximate Personalized Pagerank by directly simulating the corresponding random walks and then estimating the stationary distributions with the empirical distributions of the performed walks.

We use the recently proposed MapReduce algorithm to compute the FPPR [1]. It is based on the Monte Carlo approach. The basic idea is to very efficiently compute single random walks of a given length starting at each node in the graph. Then these random walks are used to efficiently compute the PPR vector for each node.

Algorithm 1 ABA(Query table $Q(K, A)$)

```
1: $\forall\, q \in Q$ , $\mathcal{P}_q = \{\}$
2: $\mathcal{R} = WIK(Q)$.
3: $\mathcal{R} = \mathcal{R} \cap WIA(Q)$ {Relevant web tables.}
4: for all $T \in \mathcal{R}$ do
5:     for all $q \in Q$ and $t \in T$, s.t., $q[Q.K] \approx t[T.K]$ do
6:         $\mathcal{P}_q = \mathcal{P}_q \cup \{(\ v = t[T.B]\ ,\ S_T(v) = S(T)\ )\}$
7: $\forall\, q \in Q$, Fuzzy group $\mathcal{P}_q$ to get $\mathcal{G}_q$
8: for all $q \in Q$ do
9:     $\forall\, g \in \mathcal{G}_q$, s.t., $v$ =centroid($g$),
           $S(v) = \mathcal{F}_{(x_i, S_{T_i}(x_i)) \in \mathcal{P}_q | x_i \approx v}\ \ S_{T_i}(x_i)$
10: $\forall\, q \in Q,\ \ q[Q.A] = \text{argmax}_v\ \ S(v)$
```

5. SUPPORTING CORE OPERATIONS

We discuss how we support the core operations using our holistic matching framework. Note that for each operation, we re-define the DMA score.

5.1 Augmentation-By-Attribute (ABA)

We have discussed the ABA operation already in Section 2. Here, we mention the details of the 3 query time steps abstracted in Section 3. We present the pseudo code for the ABA operation in Algorithm 1.

• Q1: *Identifying the seed table:* The seed tables for $Q(K, A)$ are identified using the WIK and WIA indexes such that a web table $T(K, B)$ is considered if there is at least one key overlap and $Q.A \approx T.B$. The DMA scores are computed using Eq. 3.

• Q2: *Computing the tables TSP scores:* is identical to Step 2 in Section 3.

• Q3: *Aggregating and processing values:* This step is identical to the predict values step in the augmentation framework discussed in Section 2.

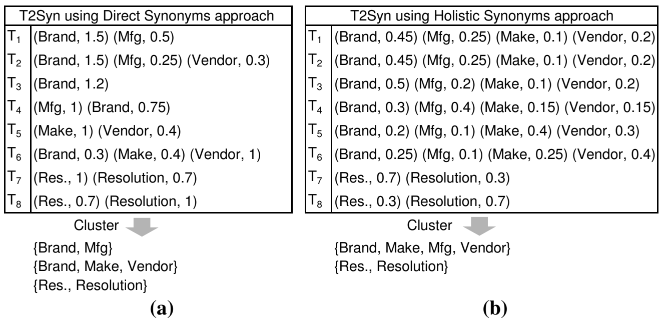
5.2 Augmentation-By-Example (ABE)

This is a variation of ABA operation. Instead of providing the augmenting attribute name, the user provides the query table with some complete records as examples, i.e., for some of the keys, she provides the values on the augmenting attribute (e.g., Figure 1(b)).

DEFINITION 3. *Augmenting-By-Example (ABE): Given query table $Q(K, A) = Q^c \cup Q^e$, where Q^c denotes the set of records $\{q^c \in Q \mid q^c[A] \neq null\}$ (referred as example complete records) and Q^e denotes the set of records $\{q^e \in Q \mid q^e[A] = null\}$ (referred as incomplete records), predict the value of each incomplete record $q^e \in Q^e$ on attribute A.*

The 3 query time steps for the ABE operation is identical to those of ABA operation, except for the way we identify the seed tables and compute the DMA scores. DMA considers a web table T to match the query table Q iff the records Q^c overlaps with those in T. For example, in Figure 2, table T_1 is considered a seed table for the query table illustrated in Figure 1(b), because they overlap on the record (`S80, Nikon`). Given the query table, we use the WIKV index to get the seed tables efficiently.

Intuitively, a web table T should be assigned a high DMA score if, for each shared key between T and Q^c, the two tables agree on the value of the augmenting attribute as well. Accordingly, we



T2Syn using Direct Synonyms approach	
T_1	(Brand, 1.5) (Mfg, 0.5)
T_2	(Brand, 1.5) (Mfg, 0.25) (Vendor, 0.3)
T_3	(Brand, 1.2)
T_4	(Mfg, 1) (Brand, 0.75)
T_5	(Make, 1) (Vendor, 0.4)
T_6	(Brand, 0.3) (Make, 0.4) (Vendor, 1)
T_7	(Res., 1) (Resolution, 0.7)
T_8	(Res., 0.7) (Resolution, 1)

Cluster

{Brand, Mfg}
{Brand, Make, Vendor}
{Res., Resolution}

(a)

T2Syn using Holistic Synonyms approach	
T_1	(Brand, 0.45) (Mfg, 0.25) (Make, 0.1) (Vendor, 0.2)
T_2	(Brand, 0.45) (Mfg, 0.25) (Make, 0.1) (Vendor, 0.2)
T_3	(Brand, 0.5) (Mfg, 0.2) (Make, 0.1) (Vendor, 0.2)
T_4	(Brand, 0.3) (Mfg, 0.4) (Make, 0.15) (Vendor, 0.15)
T_5	(Brand, 0.2) (Mfg, 0.1) (Make, 0.4) (Vendor, 0.3)
T_6	(Brand, 0.25) (Mfg, 0.1) (Make, 0.25) (Vendor, 0.4)
T_7	(Res., 0.7) (Resolution, 0.3)
T_8	(Res., 0.3) (Resolution, 0.7)

Cluster

{Brand, Make, Mfg, Vendor}
{Res., Resolution}

(b)

Figure 4: Example of web tables synonyms discovery using DMA and Holistic approaches

redefine the DMA matching score as the fraction of the shared keys that agree on the value of the augmenting attribute;

$$S_{DMA}(T) = \frac{|Q^c \cap_{KV} T|}{|Q^c \cap_K T|} \tag{8}$$

where $|Q^c \cap_{KV} T|$ denotes the number of overlapping records between the complete records of the query table Q and the web table T. Recall that $|Q^c \cap_K T|$ denotes the number of shared keys.

5.3 Attributes Discovering (AD)

Another operation of our framework is to discover important attributes of a given set of entities (e.g., Figure 1(c)). Here, we are given a query table with a list of keys (i.e., entities names) and we predict attributes for the query table that can be populated from the web tables. We also provide synonyms for each of the discovered attributes.

DEFINITION 4. *Attributes Discovery (AD): Given the keys of a query table Q, find a set of attribute names, with their synonyms, that can be augmented using the web tables.*

A naïve approach is to simply take the union of the attribute names $T.B$ from the relevant web tables $T(K, B)$. Recall that relevant web tables are the ones that overlap with the query table on the keys. For example in Figure 2, tables $T_1 \dots T_8$ overlap with the query table on the keys. In this case, the list of discovered attributes is {Brand, Mfg, Make, Vendor, Resolution, Res.}. These represent two distinct attributes with their synonyms, but there is no way to distinguish between the distinct attributes names and their synonyms.

Our main insight is that if we have the synonyms for $T.B$ of each web table T (presumably pre-computed), we can perform clustering based on the notion of set overlap of these synonyms, and thus, make the distinction between the distinct attributes and their synonyms. For example, in Figure 4(b), if these are the synonyms for each table, then using the set overlap we can cluster these synonyms to get two clusters representing the Brand and Resolution attributes. We start by discussing how we pre-compute synonyms for web tables attributes and then we show the details of the query time processing steps for the AD operation.

Synonyms discovery for each web table: This is the preprocessing step (P5) discussed in Section 3. To discover synonyms of $T.B$ of a web table $T(K, B)$, we propose to first find all tables $T'(K, B')$ that match T with their matching score $S(T, T')$ (using schema matching techniques). Then, each web table T' predicts B' as a synonym for B with the corresponding score $S(T, T')$ and returns $(B', S(T, T'))$. We end up with a set of predictions $\mathcal{P}_T = \{(B_1, S(T, T_1)), (B_2, S(T, T_2)), \dots\}$. Finally, we aggregate the scores similar to Eq. 1 and insert the top d synonyms into the synonyms index T2Syn(T) as synonyms of T (we use $d = 20$ in our evaluations).

Algorithm 2 AD(Query table Q, Web tables)

```
1: $\mathcal{R}$ =WIK($Q$) {Relevant tables using the inverted index}
2: AllSynSet = { }
3: for all $T \in \mathcal{R}$ do
4:     AllSynSet = AllSynSet $\cup$ $(T2Syn(T), S_{Hol}(T))$
5: return Cluster(AllSynSet)
```

A simple way to find the matching tables T' is to use the direct neighbors of T in the SMW graph. We call this approach as the *direct synonyms* approach. In our example, the discovered direct synonyms for each web table are shown in Figure 4(a). Note that for T_1, direct synonyms approach misses "Make" and "Vendor" as synonyms from tables T_5, T_6, because they are not direct neighbors to T_1. This results in bad quality AD. For example, when we cluster the discovered direct synonyms, we end up with 3 clusters (as shown in Figure 4(a)), 2 of them represent the same attribute "Brand", which is an undesirable output of AD.

On the other hand, if we use our holistic approach to get the matching tables, we obtain more complete synonym set for each web table (*holistic synonyms approach*). Continuing with our example, in Figure 4(b), T_5 and T_6 are reachable from T_1 and hence the PPR scores for T_5 and T_6 w.r.t to T_1 will be non-zero, and both T_5 and T_6 will contribute to the synonym set of T_1. The discovered holistic synonyms for each web table are shown in Figure 4(b). The more complete synonym set results in better quality AD. In this case, if we cluster the holistic synonyms, there will be two clusters (or synonym sets) representing the "Brand" and "Resolution" attributes, which is accurate and meaningful.

AD query time steps: We describe here the 3 query time processing steps for the AD operation. We present the pseudo code of this operation in Algorithm 2.

• Q1: *Identifying the seed tables:* Here we are provided only with the keys of the query table, and therefore, DMA identifies the relevant tables as the seed tables (those that overlap with the query table on the key attribute). Accordingly, we redefine the DMA matching score of Q and T as the fraction of the overlapping keys;

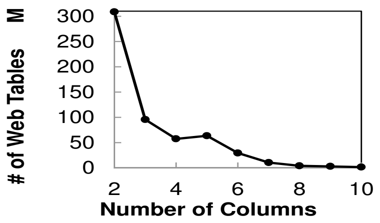
$$S_{DMA}(T) = \frac{|Q \cap_K T|}{\min\{|Q|, |T|\}} \tag{9}$$

For example, in Figure 2, table T_1 is a seed table as it overlaps with the query table on the key S80. $S_{DMA}(T_1) = \frac{1}{4}$. We use the WIK index to get the seed tables efficiently.

• Q2: *Computing the TSP scores:* is identical to Step 2 in Section 3.

• Q3: *Aggregation and processing values:* Each web table T outputted by Step 2 predicts a set of synonyms T2Syn(T) with $S_{Hol}(T)$ as the score and returns (T2Syn(T), $S_{Hol}(T)$). After processing all matched tables, we end up with a set of synonym sets along with their corresponding prediction scores. The sets of synonyms are then clustered and the prediction scores are aggregated for each cluster. Finally, we return a sorted list of clusters—each cluster represents a discovered attribute with its synonyms.

For the clustering, we follow the standard agglomerative clustering steps. Initially, each synonym set T2Syn(T) is a cluster. There is a score associated with each cluster; initially, it is $S_{Hol}(T)$. Within each synonym set, the elements are stored with their score (as shown in Figure 4), and therefore, we use the cosine similarity to compute the similarity between two clusters. When we merge two clusters, (i) we compute the resulting cluster score by aggregating their individual cluster scores, and (ii) we follow a standard document clustering paradigm by constructing a new cluster con-



(a) columns # dist.

# of web tables	573M
Avg. # columns	3.09
Avg. # rows	26.54

(b) Statistics

Figure 5: The distribution of the number of columns per web table and statistics about the relational web tables and

taining the union of the elements in both original clusters, while aggregating the elements' scores (i.e., adding the elements' weights if the element is shared between the two clusters) and then normalize the scores within the new cluster. Finally, at the end, we return a ranked list of clusters.

6. HANDLING N-ARY WEB TABLES

Throughout our discussion, we assume that the web tables are entity-attribute binary (EAB) relations. The result is working with a simpler graph with a single score among the nodes, and this enables us to model the problem as a TSP problem. If we consider n-ary web tables and use a single score among the nodes, a matching score between the query table and a web table will not say which column of the web table is the desired augmenting attribute.

In practice, not all the tables on the web are binary relations. Fortunately, relational tables on the web are meant for human consumption and usually it has a subject column [5, 19]. According to [5], there are effective heuristics to identify web table's subject. For example, using the web search log where the subject column name will have high search query hits (i.e., the subject column name appears in the search query that hits the web page containing the web table), and also usually the subject column appears in the left most column. If the subject column can be identified, then we *split* the table into several EAB relations, i.e., the subject column with each of the other columns comprise a set of EAB relations. The main assumption that we make on the web table is that the subject appears in a single column and we do not consider multiple columns as subjects.

In this paper, we do not assume anything and in this case, we split an n-ary web table into $(n-1)^2$ EAB relations—all possible pairs of columns are considered EAB relations. Our study shows the feasibility of doing that. Figure 5(a) shows the distribution of the number of columns per a relational web table. The average is 3.1 and about 54% are binary tables and 70% are either binary or ternary relations.

7. EXPERIMENTAL EVALUATION

We present an experimental evaluation of the techniques proposed in the paper for the three core operations. The goals of the study are:

• To compare holistic matching approach with DMA, DMA with attribute synonyms and the state-of-the-art approach, Octopus [4], in terms of precision and coverage for the ABA operation
• To compare holistic matching approach with DMA in terms of precision and coverage for the ABE operation
• To study the sensitivity of quality (precision and coverage) of the approaches to "head" vs "tail" query entities
• To study the sensitivity of quality to the number of example complete records for ABE operation
• To evaluate the quality of the attribute discovery algorithm based on holistic approach
• To evaluate the (direct) impact of our novel features (context, table-to-context, URL, tuples similarities) on the quality of the SMW graph as well as its (indirect) impact on the quality of ABA operation
• To evaluate the holistic approach in terms of query response times and compare with Octopus

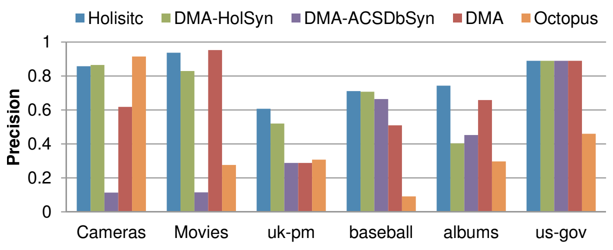
Table 2: Query entity domains and Augmenting Attributes

Dataset name	Entity (Key attribute)	Augmenting attribute
Cameras	Camera model	Brand
Movies	Movie	Director
baseball	Baseball team	Player
albums	Musical band	Album
uk-pm	UK parliament party	Member of parliament
us-gov	US state	Governor

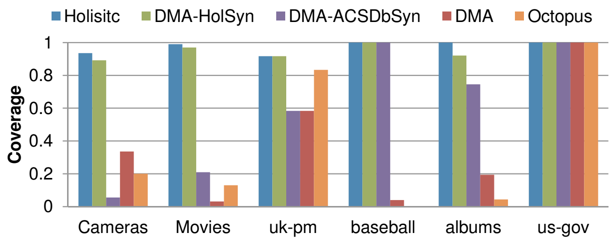
7.1 Experimental Setting

Implementation: We implemented the INFOGATHER system described in Figure 3. In the offline preprocessing step, we extracted 573M entity-attribute HTML tables from a recent snapshot (July, 2011) of Microsoft Bing search engine; such snapshots are available in the internal MapReduce clusters within Microsoft. We then built the WIK, WIKV and WIA indexes, built the SMW graph, computed T2PPV index and the T2Syn index. We performed all these steps in our MapReduce clusters as discussed in Section 3. To build the SMW graph, we use the solution described in [9] and used a df-cut of 99.9%. We stored the indexes (WIK, WIKV, WIA, T2PPV and T2Syn) on a single machine for query processing. We used an Intel x64 machine with 8 2.66GHz Intel Xeon processors and 32GB RAM, running Windows 2008 Server for this purpose.
Datasets: We conducted experiments on 6 datasets shown in Table 2. For example, for the cameras dataset: the ABA operation augments the brand given a set of camera model names and the string "brand", the ABE operation augments the brands of a set of camera model names given a set of (model, brand) pairs and the AD operation discovers attributes given a set of camera model names. Toy examples of inputs and outputs for this dataset is shown in Figure 1.

We chose 4 datasets (baseball, albums, uk-pm, us-gov) that were also used to evaluate Octopus. We compiled the complete ground truth for these datasets by manually identifying a knowledgebase and extracting the desired information from it. For example, for baseball, we got the "all-time roster" for a randomly chosen set of 12 baseball teams from Wikipedia; for albums, we got all the albums for a randomly chosen set of 14 bands from Freebase. We chose two additional datasets (cameras, movies) for which we had complete ground truth (from Microsoft Bing Shopping product catalog and IMDB database, respectively). One distinguishing characteristic of these two datasets are that the augmenting attribute has 1:1 relationship with the key (as opposed to 1:n in the above 4 datasets). We generate a query table by randomly selecting keys from the ground truth. For the movies we use a query table of 6,000 and for the cameras 1,000. All our results are averaged over 5 such query tables. We use $\mathcal{F} = sum$ in Eq. 1 for all our experiments.
Measures: Since some of the datasets have 1:n relationships between the key and augmenting attribute, we generalize the precision and coverage measures defined in Section 1 as follows. We first compute the precision and coverage *for each key* as follows:
$precision = \frac{\#values\_correctly\_predicted}{\#values\_predicted}$
$coverage = \frac{\#values\_predicted}{\#values\_in\_ground\_truth}$
We average over all the keys in the query table Q to obtain the precision and coverage for Q. Recall that if the ground truth has k values for a key on the augmenting attribute, the augmentation framework selects the top-k values for that key.



(a) ABA precision



(b) ABA coverage

Figure 6: Augmenting-By-Attribute (ABA) evaluation

7.2 Experimental Results

Evaluating Augmentation-By-Attribute (ABA) We implemented five different approaches for ABA:

• *Holistic*: This is our approach using TSP.
• *DMA*: This is the direct matching approach.
• *DMA with attribute synonyms*: This is the DMA approach where we use a set of synonyms for the augmenting attribute. A web table $T(K, B)$ will be used for prediction if its keys overlap with those in the query table and $Q.A$ matches with *any* of the synonyms of $T.B$. We consider two approaches to get the synonyms: (i) The state of the art technique to get the synonyms using the attribute correlation statistics database (ACSDb) as described in [5]. The algorithm requires a context attribute name for each dataset. We provide the key attribute name as the context. We refer to this approach as *DMA-ACSDbSyn*. (ii) In the second approach, the synonyms are obtained using our holistic approach. We take the holistic synonyms T2Syn(T) for each seed table. This represents an alternate way to using our holistic framework and we refer to it as *DMA-HolSyn*. The objective is to demonstrate the impact of using holistic synonyms (in *DMA-HolSyn*) in comparison with state-of-the-art attribute synonyms using web tables.
• *Octopus*: This is the EXTEND operation using the MultiJoin algorithm introduced in [4]. This is the state of the art to do ABA using web tables. Given a query table and an attribute name a, MultiJoin composes a web search query in the form "k a" for each key k in the query table. Then all the web tables in the resulting web pages from all the web search queries are obtained. The resulting web tables are then clustered according to their schema similarity. Finally, the cluster that best cover the query table is selected to join each of cluster member with the query table and augment the values.

Figure 6 reports the precision and coverage. The two approaches based on the holistic framework, *Holistic* and *DMA-HolSyn* significantly outperform all other approaches both in terms of precision and coverage. The average precision (over all 6 datasets) of *Holistic* and *DMA-HolSyn* are 0.79 and 0.70 respectively compared with 0.65, 0.42 and 0.39 for *DMA*, *DMA-ACSDbSyn* and *Octopus* respectively. The average coverage (over all 6 datasets) of *Holistic* and *DMA-HolSyn* are 0.97 and 0.95 respectively compared with 0.36, 0.59 and 0.38 for *DMA*, *DMA-ACSDbSyn* and *Octopus* respectively. This shows that considering indirect matches and computing the scores holistically improves both precision and coverage. The holistically discovered synonyms used in *DMA-HolSyn* are of high quality as shown later in Figure 9. *DMA-HolSyn* has slightly lower precision because it uses the synonyms to "pull in" indirectly matching tables (we simply choose the top 20 synonyms); *Holistic* uses the holistic scores as well.

DMA demonstrates high precision with all the datasets except for cameras where it was 60%; the main limitation of *DMA* is coverage as it does not consider indirectly matching tables.

DMA-ACSDbSyn has lower precision compared to *DMA*, due to the quality of the synonyms used. We manually inspected the synonyms we get from the ACSDb; there were almost no meaningful synonyms in the top 20 for the cameras and movies datasets. This is because *DMA-ACSDbSyn* uses only schema-level correlations to compute synonyms; attribute names are often ambiguous (e.g., the attribute name "name") leading to many spurious synonyms. *DMA-HolSyn*, on the other hand, uses both instance level and schema level information and performs holistic match; it discovers much better quality synonyms as shown in Figure 9.

Octopus demonstrates low precision as well as low coverage for all the datasets, except for the cameras where the precision was high and for the us-gov dataset where the coverage was high. On average the coverage is about 33%, which matches the results reported in [4]. Octopus uses the web search API to retrieve matching tables; since web search is not meant for matching tables, in many cases, the top 1000 returned urls did not provide *any* matching tables. Furthermore, Octopus's architecture does not consider indirectly matching tables and does not perform holistic matching.

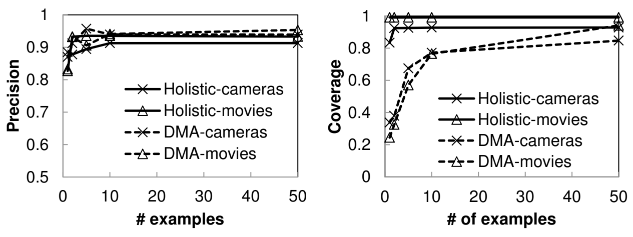
Evaluating Augmentation-By-Example (ABE): We study the sensitivity to the number of example complete records, as well as, the sensitivity to the nature of the provided examples in terms of being famous (head) or rare (tail) examples. Head (tail) examples are those records that show up in a high (low) number of web tables.

In Figure 7, we report the precision and coverage of the augmented values for the query table using the *Holistic* and *DMA* approaches as we increase the number of example complete records between 1 and 50. We report the results for the cameras and movies datasets; for the other datasets, we observe quite similar results.

In Figure 7(a), the reported precision is high for both datasets using the two approaches. However, in Figure 7(b), the *Holistic* significantly outperforms the *DMA* in coverage when there are very few example complete records (between 1 and 10). The *Holistic* provides values for about 99% and 93% of the incomplete records for the movies and cameras datasets, respectively, while the *DMA* provides coverage in the range between 20% and 75% for up to 10 examples. This shows that even with small number of example complete records the *Holistic* can get enough tables to augment the query table while the *DMA* does not.

The number of example complete records is not the only factor impacting the coverage. The frequency of the examples in the web tables also impacts the performance. We note that the query records distribution on the web tables follow the power law. Hence, if the example complete records appear in a lot of web tables (head or famous query records), then we will directly match a lot of web tables to increase the coverage. On the other hand, if the examples are tail records, there will be very few direct matching web tables.

In Figure 8, we do joint sensitivity analysis of both the number of example complete records and the nature of the records (i.e., being head, tail or mid—records in the middle). We also report the results for number of examples of 2, 10, and 50. The precision results were high and similar for all the datasets. The figure shows the coverage for the movies dataset; we observed similar behavior for the other datasets.



(a) Precision (b) Coverage

Figure 7: Sensitivity of the precision and coverage to the number of examples. The Holistic shows high precision and maintains high coverage in comparison to DMA.

The *DMA* is sensitive to both the number of example records and their nature. The coverage degrades as we decrease the number of examples; and it degrades further if the examples are tail records. On the other hand, the *Holistic* does well and maintains a coverage of 99% even in the hard situations with small number of example tail records.

Attributes Discovery Evaluation: In Figure 9, we show a sample of the discovered attributes with their synonyms for 4 of the datasets. We found these discovered synonyms to be of much higher quality than the ones generated by the state-of-the-art technique (using ACSDb [5]).

Impact of new features for building SMW graph: The objective of this experiment is to evaluate the usefulness of our proposed set of features for matching web tables in comparison to other features used in the literature before. In our evaluation, we compare four techniques: (1)*SMW Graph*: This represents all our proposed set of features, (2)*Traditional*: This represents the schema and instance level features. We use the features that represent the similarity between the attributes names, as well as, the similarity between the values in each column. (3) *WTCluster*: This represents the features that were introduced in [4] for matching web tables, namely, the table text similarity and columns widths similarity. (4) *Traditional & WTCluster*: This combines both the previous two sets of features.

We first compare the above techniques on the quality of the SMW graph. In this experiment, we randomly picked 500 web tables relevant to each of the datasets, and then we computed the features values and identified labels for each pair of web tables as being a match or not (using our automatic labeling technique described in Section 4.1). We created a balanced set of examples (i.e., almost equal negative and positive examples). We trained a classifier using the examples and reported in Figure 10(a) the model's accuracy per dataset in addition to the overall accuracy. The displayed results are average of 5 different runs to the above procedure.

In general, the web tables matching accuracy using our set of features, *SMW Graph*, shows the best performance. *SMW Graph* has about 6% improvement over the *Traditional* and about 10% improvement over *ClusterWT*. *SMW graph* also outperforms the combined set of features *Traditional & WTCluster* by about 3%. These results prove the importance of the new introduced features for matching web tables.

The *SMW graph* technique is consistently outperforming the other techniques, however, the other techniques are not consistently reliable. For example, in the uk-pm dataset, the *Traditional* technique performs better than *WT-Cluster*. The situation is reversed in the us-gov dataset.

In Figure 10(b), we evaluate the impact of our features on the quality of the ABA operation; that is whether the improved quality of the SMW graph translates in better quality ABA (indirect evaluation of the features). Here, we report only the precision as we obtain the same coverage using each of the features set. Note the

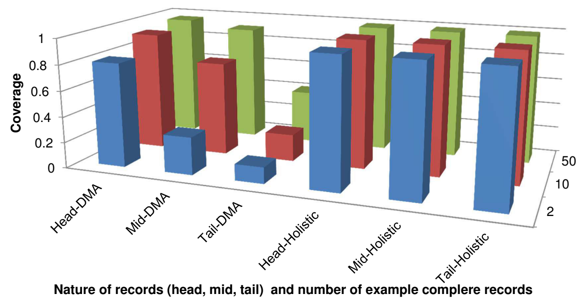


Figure 8: Joint sensitivity analysis to the number of examples and the head vs. tail records in the web tables. The Holistic is robust in comparison to the DMA.

similarity between the two figures 10(a) and 10(b). The *SMW graph* features get better quality SMW graph, and hence, better precision.

Efficiency evaluation: We evaluate in Figure 10(c) the efficiency of our approach and architecture in comparison with the Octopus approach for the ABA operation for the cameras dataset. We obtained similar performance with the other datasets. Our implementation of Octopus involves using a web search engine API. We report the query response time as we increase the size of the query table. Our approach takes milliseconds to respond and is 4 orders of magnitude faster than Octopus. Our fast response time is due to the fast computations of the TSP scores using the pre-computed PPR vectors and indexes that we introduce in Section 3. For Octopus, most of the time goes in processing the web search queries as it is based on SOAP request/response kind of communication. As mentioned in [4], Octopus can be implemented more efficiently if web search engines support Octopus-specific operations; however, current search engines do not support such operations.

In summary, our experiments show that our holistic matching framework and proposed system architecture can support the three core operations with high precision, high coverage and interactive response times.

8. RELATED WORK

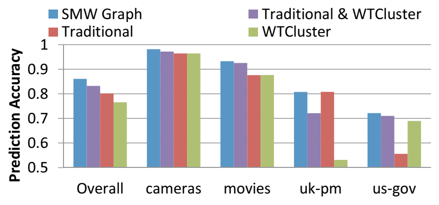
Our work is most related to the Octopus system developed by Cafarella et. al. [4]. The EXTEND operator proposed by Octopus is identical to our ABA core operation. However, the approach proposed in this paper is fundamentally different from Octopus (specifically, the recommended MultiJoin algorithm) both in terms of matching semantic, as well as, the system architecture. Octopus uses the web search API to retrieve matching tables; this does not have any well-defined semantics. On the other hand, INFOGATHER matches with the *extracted tables* themselves using schema matching techniques and TSP-based framework; and it has a well-defined semantics. Since web search is not meant for matching tables, in many cases, the top 1000 returned urls did not provide *any* matching tables, let alone indirectly and holistically matching tables. As a result, INFOGATHER provides significantly better precision and coverage (Figure 6). Secondly, INFOGATHER performs most of the "heavy lifting" at preprocessing time and performs only the aggregation at query time; Octopus needs to invoke the search API for each query record and perform clustering of web tables at query time. As a result, INFOGATHER is four orders of magnitude faster than Octopus (Figure 10(c)).

Researchers have developed techniques to annotate web tables with column names and names of relationships [19, 14]. This is complementary to our techniques and can help to build a better SMW graph.

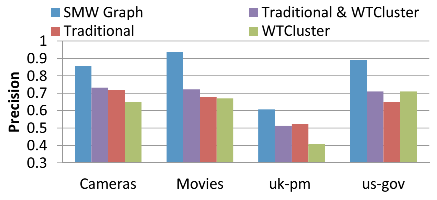
Our work on building the SMW graph is related to the vast body of work on schema matching [18, 3, 2]. Most modern approaches

cameras	movies	uk-pm	baseball
price | price each | offer	starring | character | actor | hero | name	member of parliament | mp | elected mp	players | player | name | player name
brand | manufacturer | make | vendor	director | directed by | regista | director(s)	constituency | ward | region | seats	position | d | pos | rhp | ac | category
model | product | series | modelo	film | english title | title used	date | election | year	school | hometown/school | drafted from
type | type of product | device type	nazione | country | producer(s) | pays | origine	% | votes | percentage	year | debut | season | contract term
resolution | megapixels | mp | res	time | length | playing time | min. | run time		salary | $$/year avg | $ per win | avg. salary
estimated msrp | total msrp	format | media | formato | muoto | tipo | system		organizational ranking | overall ranking
zoom | optical zoom	genere | genre | category | type		league | lg | mlb | hi lvl | position | flg
ecl or e2 | cleaning fluid | liquido	studio/distributor | company | production		stadium | ballpark | venue
output | output voltage | ouput	attori | cast | cast (subject of documentary)		debut | final game | ml debut | year

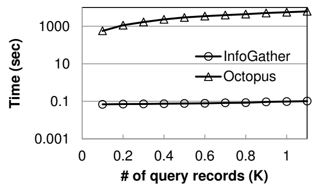
Figure 9: Example of discovered attributes with their synonyms



(a) Direct schema matching evaluation using the proposed features set



(b) The impact of the schema matching quality on the ABA operation



(c) Response time evaluation

Figure 10: Web tables matching accuracy and response time evaluation

uses several *base techniques* such as linguistic matching of attribute names and detecting overlap of data instances and combines them to determine the final matchings; the base techniques as well as the combiner can either be machine learning-based techniques or non-learning methods [8, 15]. We adopt machine learning based techniques in this paper. We propose new features that do not arise for enterprise tables. Furthermore, we need to consider the scalability challenges that arise in our scenario.

He and Chang proposed holistic matching of schemas associated with deep web query interfaces [12]. The goal is to integrate n schemas all at once such that all matching elements of the n schemas are represented only once in the integrated (mediated) schema. While our approach shares the same name as He and Chang's approach, it is different from the latter: we still perform pairwise matching of web tables but consider other web tables to perform indirect matches. He and Chang do not consider other schemas to perform such indirect and holistic matches.

Recently, Yin et. al. developed the FACTO system for answering "fact lookup queries" in web search engines (e.g., [Barack Obama date of birth]) using web tables [20]. FACTO considers only "attribute-value" tables whereas the focus of this paper is on "entity-attribute" tables.

There exists a rich body of work of leveraging HTML lists for set expansion and table augmentation [13, 10]. These works differ from ours in several aspects. First, they consider HTML lists while we focus on entity-attribute HTML tables. Second, the core operation is different: to discover more entities rather than augment the provided entities.

9. CONCLUSION AND FUTURE WORK

In this paper, we present the INFOGATHER system to automate information gathering tasks, like augmenting entities with attribute values and discovering attributes, using web tables. Our experiments demonstrate the superiority of our techniques compared to the state-of-the-art.

Our work can be extended in multiple directions. We considered three core operations in this paper; there are potentially more operations that can benefit from our framework. Examples include set expansion and answering "fact lookup" queries in web search engines (e.g., "who is the director of Titanic?"). Furthermore, we considered only entity-attribute tables in this paper; other forms of structured web data include HTML lists, attribute-value tables and deep web databases. Incorporating these data sources in our framework is also an open challenge.

10. REFERENCES

[1] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In *SIGMOD*, 2011.

[2] Z. Bellahsene, A. Bonifati, and E. Rahm. *Schema Matching and Mapping*. Springer, 2011.

[3] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. In *VLDB*, pages 695–701, 2011.

[4] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2009.

[5] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 2008.

[6] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *WebDB*, 2008.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[8] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *ACM SIGMOD*, pages 509–520, 2001.

[9] T. Elsayed, J. J. Lin, and D. W. Oard. Pairwise document similarity in large collections with mapreduce. In *ACL*, 2008.

[10] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proc. VLDB Endow.*, pages 289–300, 2009.

[11] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, 2002.

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

[13] Y. He and D. Xin. Seisa: set expansion by iterative similarity aggregation. In *WWW*, pages 427–436, 2011.

[14] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, pages 1338–1347, 2010.

[15] J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *ICDE*, 2005.

[16] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, 2001.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

[18] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, pages 334–350, 2001.

[19] P. Venetis et al. Recovering semantics of tables on the web. *Proc. VLDB Endow.*, pages 528–538, 2011.

[20] X. Yin, W. Tan, and C. Liu. Facto: a fact lookup engine based on web tables. In *WWW*, 2011.