Exploiting Content Redundancy for Web Information Extraction

Pankaj Gulhane Rajeev Rastogi Srinivasan H Sengamedu
Yahoo! Labs, Bangalore
{pankajeg, rrastogi, shs}@yahoo-inc.com

Ashwin Tengli
Microsoft IDC, Bangalore
ashwint@microsoft.com

ABSTRACT

We propose a novel extraction approach that exploits *content redundancy* on the web to extract structured data from *template-based* web sites. We start by populating a seed database with records extracted from a few initial sites. We then identify values within the pages of each new site that match attribute values contained in the seed set of records. To match attribute values with diverse representations across sites, we define a new similarity metric that leverages the templatized structure of attribute content. Specifically, our metric discovers the matching pattern between attribute values from two sites, and uses this to ignore extraneous portions of attribute values when computing similarity scores. Further, to filter out noisy attribute value matches, we exploit the fact that attribute values occur at fixed positions within template-based sites. We develop an efficient Apriori-style algorithm to systematically enumerate attribute position configurations with sufficient matching values across pages. Finally, we conduct an extensive experimental study with real-life web data to demonstrate the effectiveness of our extraction approach.

1. INTRODUCTION

A significant fraction of pages on the web are dynamically generated by populating *fixed page templates* with content from a backend DBMS. Previous studies [12] estimate that as high as 40-50% of the content on the web is template content. Popular web sites with thousands or even millions of template-based pages include retailer site `amazon.com`, aggregator site `yelp.com`, news site `cnn.com`, video site `youtube.com`, and so on.

Template-based pages contain a wealth of information about real-world entities like businesses (e.g., address, phone, category, hours of operation), products (e.g., description, price, reviews), restaurants, books, etc. Extracting this information and integrating it across pages from multiple sites can allow us to create extensive databases of entities. These databases can then be queried by users to access product features and reviews collated from different web sites, order products by price, determine the phone numbers of restaurants in a specific location, etc.

1.1 Problem

Proceedings of the VLDB Endowment, Vol. 3, No. 1

Consider a set of template-based web sites $\mathcal{W} = \{W_1, \ldots, W_m\}$ belonging to a specific vertical (e.g., restaurants, books). Each page within a web site W_i contains attribute values for a specific real-world entity. Let $A = \{a_1, \ldots, a_q\}$ denote the set of entity attributes within the vertical. In this paper, we address the following extraction problem: *Given a set of web sites $\mathcal{W}$, from each page of web sites $W_i \in \mathcal{W}$, extract a single record containing the attribute values for the corresponding entity.* We restrict our attention to extracting "string-valued" attributes from "single-entity" pages in this paper.

1.2 Our Approach

In this paper, we develop a new approach to extract entity records from multiple template-based web sites. Our extraction solution delivers high accuracy while incurring minimal manual effort, and leverages the following two key properties of template-based sites:
Property 1: Multiple sites contain pages for the same entity. Further, the values of an attribute across the various pages for an entity are "textually" similar.
Property 2: Pages within a web site have a similar structure conforming to a common template.

Property 1 implies that there is redundant content across web sites. For example, multiple sites like `yelp.com`, `superpages.com`, etc. contain entity pages for overlapping sets of restaurants, and the values of key attributes like name, address, and phone number are similar across the various pages for a restaurant. We exploit this content redundancy by using extractions from one site to identify attribute values in the pages of overlapping entities in a different site. Property 2 implies that attribute values occur at fixed positions within pages of a site. Thus, once we have identified attribute values in a few pages of a site, we can infer their positions, and use this to extract attribute values from the remaining pages of the site.

Our extraction procedure starts by populating a seed database R of records from a few initial sites. These records are extracted from the sites by having human editors annotate attribute values in a few sample pages from each site, and learning wrappers for the sites. Note that each seed record in R contains attribute values for an entity from a single entity page. Now, a new site $W_i \in \mathcal{W}$ will typically contain pages for entities that already have records in R. Thus, we scan the pages in W_i to find values that match attribute values in the seed records. These matching attribute values within the pages of overlapping entities are used to learn wrappers that are subsequently used to extract records from the remaining pages of the new site. The seed database is augmented with the newly extracted records (some of which will be for new entities) and the process is repeated for other new sites.

Note that our extraction approach is largely *unsupervised*. Ex-

	Name	Address
r_1	Beijing Bites	120 Lexington Avenue New York, NY 10016
r_2	China Club	312 W 34th Street New York, NY 10001

(a) Seed database R.

Bejing Bites

120 Lexington Ave
(between 28th and 29th St)
New York, NY 10016

Nearest Transit:
Lexington Ave
New York, NY

Related Restaurants:
China Club
China Grill

(b) Page p_1.

China Club

312 W 34th St
(between 8th and 9th Ave)
New York, NY 10001

Nearest Transit:
Penn Station
New York, NY

Related Restaurants:
Bejing Bites
China Grill

(c) Page p_2.

Figure 1: Web extraction example.

cept for a few initial sites where human annotations are needed to generate the seed database, it eliminates the need for manually annotating attribute values in pages of web sites. Moreover, by continuously expanding the initial seed database with extracted records for new entities, it ensures that there is sufficient overlap between the seed database and new sites. In the following subsections, we describe some of the key challenges with our content matching-based approach, and our proposed solutions to overcome them.

1.3 Key Challenges

Our extraction approach faces two major challenges. First, it relies heavily on the fact that attribute values across pages for the same real-world entity are textually similar. However, in practice, entity attribute values can vary between sites due to data entry errors, abbreviations, heterogeneous representations, etc. This can lead to attribute value matches going undetected during extraction. Second, web pages are inherently noisy and contain extraneous values that can lead to spurious attribute value matches. The following example, inspired by real-life web data, illustrates the two challenges.

EXAMPLE 1.1. *Consider the seed database R in Figure 1(a) with two records r_1 and r_2 containing the name and address attributes of restaurants. Consider the two restaurant pages p_1 and p_2 in Figures 1(b) and (c), respectively, from a new web site. The name and address of the restaurants are at the top of the pages, and the nearest transit subway station and related restaurants with similar cuisine are listed at the bottom. It is straightforward to see that record r_i and page p_i ($i = 1, 2$) refer to the same real-world entity. Observe that, in the name attribute, "Beijing" is misspelled as "Bejing" in pages p_1 and p_2. Similarly, in the address attribute, the terms "Avenue" and "Street" are abbreviated to "Ave" (in p_1) and "St" (in p_2), respectively, and an additional line starting with "(between" is uniformly inserted in both pages.*

Now, suppose we use the Jaccard coefficient [19] to measure the similarity between attribute values, where each value is treated as a bag of words (using space as delimiter), and for two sets S_1 and S_2, the Jaccard similarity is defined as $JC(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$. Due to the different conventions for recording the address attribute, the Jaccard similarity between the address values in r_1 and p_1 belonging to the same entity is only $\frac{6}{13}$, which is less than 0.5. In contrast, the nearest transit value "Lexington Ave New York, NY" at the bottom of p_1 has a higher similarity score of $\frac{4}{8}$ with the address value in r_1. Similarly, the name values in r_1 and p_1 have a similarity score of only $\frac{1}{3}$, while the string "China Club" under "Related Restaurants:" in p_1 has a similarity of 1 with the name value in r_2. Thus, extraneous values at the bottom of p_1 have much higher similarity scores with name and address values in the seed records which can lead to false positive matches. □

1.4 Our Contributions

In this paper, we make the following contributions to solve the above-mentioned problems.

1) We propose a novel extraction approach that exploits content redundancy across sites and structural similarity among template-based pages to extract attribute values with high precision and minimal manual intervention.

2) To cope with different attribute formatting conventions across sites, we define a new similarity function. Our new function leverages the fact that attribute values in template-based pages have a templatized structure and uses this to improve matching accuracy. Our proposed metric discovers repeating patterns among the matching portions of attribute value pairs from two sites, and uses this to filter out non-matching portions when computing the similarity score between attribute value pairs. In Figure 1, our similarity function detects that the lines beginning with "(between" are extraneous across the address values in p_1 and p_2, and so ignores them when matching the values with addresses in the seed database. This boosts the similarity scores (Jaccard similarity $= \frac{3}{4}$) between address values in pages p_1 and p_2, and their corresponding values in records r_1 and r_2, respectively, and ensures that they are matched correctly. Thus, our new similarity metric is able to match attribute values (with diverse representations) belonging to the same entity while keeping the number of false positive matches low.

3) In order to further filter out noisy matches, we match values for multiple attributes and also exploit the fact that attribute values occur at fixed positions within the pages of a template-based web site. Thus, suppose that for a configuration of attribute positions, we define the *support* as the number of pages for which the values at the positions match seed record values. Then, we can essentially prune configurations with insufficient support. We propose an efficient Apriori-style [3] algorithm to systematically enumerate attribute position configurations with sufficient support. In Figure 1, our algorithm will prune the spurious match between the string "China Club" under "Related Restaurants:" in p_1 and the name attribute in r_2 since the address value in page p_1 does not match the address value in record r_2 for restaurant "China Club".

4) We conduct an extensive experimental study with real-life web datasets from different verticals. Our extraction approach performs well delivering greater than 95% precision and more than 80% recall for extracted records from a wide range of web sites.

2. SIMILARITY MODEL

A basic primitive for our extraction task is to be able to find matching attribute value pairs between two sets U and V of values for an attribute a. The attribute values in a set can either come from records in the seed database, or they can be text values at a particular position within the pages of a web site. Observe that this is different from previous work which primarily focuses on determining if two values are similar.

In the following subsections, we first consider the case of com-

Algorithm 1 STRONGSIM

Input: Sets of values U, V, Attribute a;
Output: Strong similarity scores $ssim_a$ for value pairs $u \in U$ and $v \in V$;

Let $WS = \{(u, v) : u \in U \wedge v \in V \wedge sim_a(u, v) \geq T_w\}$;
Initialize $B()$ to $\emptyset$;
for each weakly similar pair $(u, v) \in WS$ **do**
  Construct a bipartite graph G with two sets of vertices corresponding to words in u and v;
  for each word pair $w \in u$, $w' \in v$ **do**
    if $JC(qg(w), qg(w')) \geq 0.9$ **then**
      Add edge (w, w') with weight $JC(qg(w), qg(w'))$ to G;
    end if
  end for
  Compute the max-weight bipartite matching $M(u, v)$ of G;
  $P(u, v)$ = MATCHINGPATTERN$(u, v, M(u, v))$;
  Let $u'(v')$ be the subsequence of $u(v)$ containing only the matching words in $M(u, v)$;
  Add the pair (u', v') to $B(P(u, v))$;
end for
for each pair (u, v), $u \in U, v \in V$ **do**
  if (u, v) is weakly similar and $|B(P(u, v))| \geq \alpha \cdot |WS|$ **then**
    $ssim_a(u, v) = sim_a(u', v')$;
  else
    $ssim_a(u, v) = sim_a(u, v)$;
  end if
end for
return $ssim_a$;

puting the similarity between a pair of values, and subsequently, we use this to determine the similar value pairs between two sets of attribute values.

2.1 Similarity Between Value Pairs

We treat each attribute value string as a sequence of words separated by special characters such as space, tab, hyphen, comma, etc. which are considered as word delimiters. We normalize each word by converting all the letters to lower case.

For our web extraction scenario, we require a similarity metric that is robust to typographical errors, word re-orderings, abbreviations, etc. A number of similarity functions for approximately matching strings have been proposed in the research literature. Popular measures include the Jaccard coefficient and Cosine similarity metrics from information retrieval (IR) [19, 8], extensions (of Cosine similarity) to use *q-grams* instead of words [17], and the edit distance family of functions [10, 24, 18, 22]. We use $sim_a(u, v)$ to denote the similarity between strings u and v when u and v are considered as values of the attribute a. We use a variant of the "Cosine similarity over q-grams" similarity function proposed in [17] because it can handle both spelling errors as well as word rearrangements, and is computationally efficient. Appendix A gives the details of $sim_a(u, v)$ computation. We refer to $sim_a(u, v)$ as the *weak similarity* score between u and v. Further, we say that values u and v are *weakly similar* if $sim_a(u, v) \geq T_w$ where T_w is the weak similarity threshold.

2.2 Similarity Between Sets of Values

A straightforward approach to find similar value pairs from two sets $U = \{u_1, u_2, \ldots\}$ and $V = \{v_1, v_2, \ldots\}$ is to simply consider individual pairs of values (u_i, v_j) and check if they are weakly similar. But this may not work in many cases because different attribute representations between two sites can cause the similarity between a pair of attribute values for the same entity to be low. In Figure 1, lines beginning with "(between" are inserted into the address values in pages p_1 and p_2. These values will not be judged as similar to the corresponding address values in seed records r_1 and r_2 unless the weak similarity threshold is set to a small value. However, setting the weak similarity threshold too low can lead to false positive matches between extraneous values (e.g., nearest transit) in pages and attribute values in the seed database.

To prune the spurious matches resulting from a small weak similarity threshold, our solution exploits the fact that within template-based sites, attribute content also follows a templatized format. So there is typically a fixed pattern of matching segments between attribute value pairs for the same entity. For instance, in Figure 1, if we ignore the line beginning with "(between", then the two lines consisting of the street name, city, state, and zip code in the address values in p_1 and p_2 match the address values in records r_1 and r_2, respectively.

We exploit this property of template-based sites to define a *strong similarity* metric between attribute value pairs. Suppose that a pair of attribute values (u, v) is weakly similar and the pattern of segment matches between u and v occurs in a sufficient number of other attribute value pairs. Then we can compute a stronger similarity score between u and v by ignoring all segments in u and v that don't match, and only considering the matching segments between u and v. Thus, unlike existing similarity functions, our strong similarity metric takes into account the matching pattern between other value pairs when determining the similarity score for a specific pair of values.

Strong similarity essentially boosts similarity scores between sets of values that have a templatized structure. Since our attribute value content within a page is templatized, its similarity will be boosted, while the similarity scores of noisy values in a page will not be boosted to the same extent. Thus, by setting a high enough strong similarity threshold, we can retain the legitimate attribute value matches and filter out the false positive matches.

Procedure STRONGSIM describes the steps for computing strong similarity scores between value pairs from two sets U and V of values for an attribute a. For each weakly similar pair (u, v), it computes the segment matching pattern by first identifying the potentially matching word pairs (w, w') with Jaccard similarity $JC(qg(w), qg(w')) = \frac{|qg(w) \cap qg(w')|}{|qg(w) \cup qg(w')|} \geq 0.9$. (Here $qg(w)$ is the set of q-grams associated with w. See Appendix A for details.) It then narrows this down to a set of disjoint matching word pairs by computing a max-weight matching $M(u, v)$. Now, since attribute values for different entities can contain a variable number of words, the matching pattern over words may not be consistent across attribute value pairs (u, v) belonging to the same entity. So STRONGSIM invokes a separate procedure MATCHINGPATTERN to compute the matching pattern over segments, which are essentially contiguous subsequences of words.

Procedure MATCHINGPATTERN takes as input the set of matching word pairs $M(u, v)$, and uses this to decompose u and v into segments $s_1, \ldots, s_y$ and $s'_1, \ldots, s'_z$, respectively. Each segment in $u(v)$ is either the longest contiguous subsequence of words with a corresponding matching subsequence in $v(u)$, or the longest contiguous subsequence of words with no matching words in $v(u)$. The final segment matching pattern $P(u)$ for u is then a sequence of positive integers in which the i^{th} element is "0" if segment s_i in u does not match any segment in v; else, it is the index "j" of the segment s'_j in v that matches s_i. The segment matching pattern $P(v)$ for v is similarly derived, and the two are concatenated to derive the final matching pattern $P(u, v)$ for the attribute value pair

Algorithm 2 MATCHINGPATTERN

Input: Values $u = w_1, \ldots w_k$ and $v = w'_1, \ldots, w'_l$, Matching $M(u, v)$;
Output: Matching pattern $P(u, v)$ between u and v;

Split u into segments $s_1, \ldots, s_y$, where each segment s is the longest contiguous subsequence of words $w_i, w_{i+1}, \ldots$ in u such that either

- There is a corresponding contiguous subsequence of words $w'_j, w'_{j+1}, \ldots$ in v such that the word pairs $(w_i, w'_j), (w_{i+1}, w'_{j+1}), \ldots$ are in $M(u, v)$, or
- None of the words $w_i, w_{i+1}, \ldots$ appear in $M(u, v)$.

Segment v into segments $s'_1, \ldots, s'_z$ similar to u;
$P(u) = \epsilon$;
for $i = 1$ to y **do**
  if words in s_i do not appear in $M(u, v)$ **then**
    Append a "0" to $P(u)$;
  else
    Let s'_j be the segment in v that matches s_i in u (that is, consecutive word pairs from s_i and s'_j are contained in $M(u, v)$);
    Append the index "j" for s'_j to $P(u)$;
  end if
end for
Compute $P(v)$ in a similar manner to $P(u)$;
return $P(u) \cdot P(v)$;

(u, v).

Now, for a (u, v) pair with matching pattern $P(u, v)$, let u' and v' be the subsequences containing words from only the matching segments in u and v, respectively. For weakly similar (u, v) pairs for the same entity, we would like to boost the similarity scores to $sim_a(u', v')$ which essentially ignores the non-matching portions of u and v during similarity computation. We use the matching pattern $P(u, v)$ and matching subsequences u', v' to determine if a pair (u, v) could potentially correspond to the same entity. First, since values are templatized, we require that a sufficiently large number of pairs (for other entities) also have the pattern $P(u, v)$; otherwise the pattern may just be noise. Second, the matching subsequences for the value pairs with the pattern $P(u, v)$ must be sufficiently diverse. For example, if the matching subsequences u', v' for all address value pairs with a specific pattern is simply a generic string like "new york ny", then it is possible that the address values in each pair are for different entities, and we may not want to boost the similarity scores for such pairs. On the other hand, if the matching subsequences u', v' are distinct strings containing the street number and street name, then each address value pair with the pattern is very likely for the same entity, and so we can go ahead and boost the similarity scores between values in each pair by ignoring the non-matching portions.

So, for each pattern, STRONGSIM keeps track of the number of distinct matching subsequences u', v' for (u, v) pairs with the pattern in the set $B()$. For a weakly similar pair (u, v), if the number of distinct matching subsequences in $B(P(u, v))$ is at least a fraction α of the weakly similar pairs, then we boost the similarity score between u and v by performing similarity computation using u', v' instead of u, v. Note that, to accommodate noise in the matching patterns, we can cluster similar patterns (e.g., using edit distance as the similarity metric). Then for each pattern, the set of matching subsequences $B()$ can be obtained by taking the union of all the $B()$ sets for patterns in its cluster.

EXAMPLE 2.1. *Let U be the set of addresses in the seed records r_1 and r_2 in Figure 1, and V be the set of address values at the top of pages p_1 and p_2. Let the addresses in r_1 and r_2 be weakly similar to the address values in p_1 and p_2, respectively. We trace the execution of our strong similarity computation algorithm on inputs U and V. Consider the (u, v) value pair where u and v are the address values in record r_1 and page p_1, respectively. The matching words in u and v are "120", "lexington", "new", "york", "ny" and "10016". The value u is split into 3 segments: s_1 = "120 lexington", s_2 = "avenue", and s_3 = "new york ny 10016". Value v is also split into 3 segments: s'_1 = "120 lexington", s'_2 = "ave between 28th and 29th st" and s'_3 = "new york ny 10016". The matching patterns $P(u)$ for u and $P(v)$ for v are both "1 0 3" since s_1 matches s'_1, s_2 and s'_2 do not match any other segments, and s_3 matches s'_3. Thus, the matching pattern for the (u, v) pair is "1 0 3 1 0 3", and the matching subsequences u' and v' are both "120 lexington new york ny 10016".*

The matching pattern for the address values in r_2 and p_2 can also be shown to be "1 0 3 1 0 3" but with a different matching subsequence "312 w 34th new york ny 10001". Thus, B("1 0 3 1 0 3") contains two distinct matching subsequences which is the same as the number of weakly similar pairs, and so the strong similarity between u and v is $ssim_{addr}(u, v) = sim_{addr}(u', v') = 1$.

Now, suppose U is the set of addresses in seed records as before, but V is the set of nearest transit values in p_1 and p_2. Further, let all 4 (u, v) pairs be weakly similar, and $\alpha = 0.3$. We show that the strong similarity scores for all the (u, v) pairs are identical to their original similarity scores. Consider the (u, v) pair with u equal to the address value in r_1 and v equal to the nearest transit value in p_1. The matching patterns $P(u)$ and $P(v)$ are "0 1 0 3 0" and "2 0 4", respectively, and the matching subsequences are both "lexington new york ny". For the remaining 3 (u, v) pairs, the matching patterns $P(u)$ and $P(v)$ are "0 2 0" and "0 2", respectively, with a single matching subsequence "new york ny". Thus, the number of distinct subsequences in the $B()$ sets for the two matching patterns is 1 and this is less than α times 4, the number of weakly similar pairs. Thus, for all the (u, v) pairs, $ssim_{addr}(u, v) = sim_{addr}(u, v)$, which is low in most cases because of the small word overlap between u and v values. □

We say that attribute values u and v are *strongly similar* if $ssim_a(u, v)$ is greater than or equal to a strong similarity threshold T_s. In our experiments (see Section 4), we found that with weak and strong similarity threshold settings of $T_w = 0.5$ and $T_s = 0.9$, and $\alpha = 0.1$, we were able to match attribute values (with different formats) for the same entity without too many false positive matches.

3. EXTRACTION ALGORITHM

We are now ready to describe our algorithm that leverages strongly similar content between the seed database R and the web pages of sites in $\mathcal{W}$ for structured data extraction.

Notation: Consider the DOM tree representation of a web page p. For a node n in the page, let x be the unique path from the root to n in the DOM tree. We treat the path x as the position of node n in page p. Further, we use $p[x]$ to denote the value of the node n at position x in page p. For a leaf node, the value is essentially the text string contained in it. If n is an internal node, then its value is the concatenated sequence of text from the leaves of the subtree rooted at n (in the order in which the nodes appear in the DOM tree). For a seed record r in R, we denote the value of attribute $a_i \in A$ in r by $r[a_i]$. With each record r, we associate the web site from which the record was extracted, denoted by $W(r)$.

Algorithm: Our extraction algorithm scans the pages of a new

web site W to find node values that match attribute values in the seed database R, and uses this to infer the node position for each attribute within the pages of W. It then extracts entity records from pages of the web site by extracting the attribute values at the identified positions.

A key challenge here is to find the correct node values within a page that match the attribute values within a seed record. Even with strong similarity, due to noise and extraneous information in web pages, we may still have spurious matches within a page. For instance, in Figure 1, values like "china club" under "Related Restaurants:" can match name values in seed records.

In this section, we use multiple attribute matches and the template structure of pages to filter out additional spurious matches. Consider a subset of attributes $A' = \{a_1, \ldots, a_t\}$. Let $S = \{(a_1, x_1), \ldots, (a_t, x_t)\}$ be a configuration of attribute positions where x_i is the position of attribute a_i. We define the support $sup(S)$ of S to be the number of pages p in site W such that values at positions x_i in page p match values of attributes a_i in some seed record r. Now, the intuition behind our spurious matches pruning strategy is that legitimate attribute position configurations S will generally have high support $sup(S)$. Thus, we can simply go ahead and prune all configurations with inadequate support. Observe that in the definition of $sup(S)$, for each page p that contributes to $sup(S)$, we require the values at the designated positions in p to match the attribute values in a single record r. This ensures that the matched values in p (and r) belong to the same entity.

We can now formally state our attribute position computation problem.

ATTRIBUTE POSITION COMPUTATION PROBLEM: *Given a seed database R, web site W, and a minimum support parameter β, find a maximal set S of (attribute, position) pairs such that $sup(S) \geq \beta$.* □

Notice that we are looking to find positions for as many attributes as possible and not necessarily all attributes – this is to accommodate scenarios in which certain attributes are missing from site W. In case there are multiple sets S with the same number of (attribute, position) pairs and with support at least β, then we select the set S with the maximum support. In our experiments (see Section 4), we found that setting the support parameter $\beta = 10$ is effective at filtering out the spurious matches.

Procedure FINDATTRPOS describes an efficient algorithm for computing the maximal set S of (attribute, position) pairs with support $\geq \beta$ for a new web site W. Similarity computation between a node value and an attribute value in a record serves as a basic building block for determining record-level similarity between attribute values in a page of W and a seed record from R. We keep track of the weakly similar (page, position) and (record, attribute) pairs by storing in $WS(a, x)$ the record, page pairs (r, p) such that $r[a]$ is weakly similar to $p[x]$. For each (attribute, position) pair (a, x) such that there are a sufficient number of pages with weakly similar values in position x, we compute strong similarity scores between $(r[a], p[x])$ pairs. We store in $SS(a, x)$ the (record, page) pairs (r, p) such that $r[a]$ is strongly similar to $p[x]$. Note that $SS(a, x)$ will contain fewer false positive matches compared to $WS(a, x)$.

Due to spurious attribute value matches still contained in $SS(a, x)$, within each page of W, there may be multiple attribute position configurations for which values in the page match (portions of) a record in the seed database R. Across all the pages of W, the number of these attribute position configurations could become really large, even though most of them do not have the required support. Computing support for each of these configurations to determine the maximal configuration with support $\geq \beta$ can turn out to be very inefficient. Instead, we devise an efficient Apriori-style [3] algorithm based on the following observation: *for a pair of (attribute, position) pairs sets S, S', if $S \subseteq S'$, then $sup(S') \leq sup(S)$.* Thus, we can use an iterative Apriori-style algorithm that generates candidate (attribute, position) pairs sets of size k in the k^{th} iteration and stores these in C_k. Candidate sets whose support is less than β are pruned from C_k since any superset of these cannot have support $\geq \beta$. The remaining sets in C_k (after pruning) are used to generate supersets of size $k + 1$ which become candidates for the next iteration.

Algorithm 3 FINDATTRPOS

Input: Seed relation R, Web site W;
Output: Maximal set of (attribute, position) pairs with support $\geq \beta$;

$WS(a, x) = \emptyset$;
for each node n (at position x) in each page p of W **do**
  for each seed record, attribute pair (r, a) such that $sim_a(r[a], p[x]) \geq T_w$ **do**
    $WS(a, x) = WS(a, x) \cup (r, p)$;
  end for
end for
$C = \{\{(a, x)\} : WS(a, x)$ contains at least β distinct pages $p\}$;
for each $\{(a, x)\}$ in C **do**
  $SS(a, x) = \emptyset$;
  for each web site W' with a record in the seed database R **do**
    $U = \{r[a] : r \in R \wedge W(r) = W'\}$;
    $V = \{p[x] : p \in W\}$;
    $ssim_a$ = STRONGSIM(U, V, a);
    $SS(a, x) = SS(a, x) \cup \{(r, p) : r \in R \wedge W(r) = W' \wedge p \in W \wedge ssim_a(r[a], p[x]) \geq T_s\}$;
  end for
end for
$C_1 = C$; $k = 1$; $S_{max} = \emptyset$;
while $C_k \neq \emptyset$ **do**
  for each set $S \in C_k$ **do**
    $sup(S)$ = number of distinct pages p in $\cap_{(a,x) \in S} SS(a, x)$;
    Prune S from C_k if $sup(S) < \beta$;
  end for
  Set S_{max} to set in C_k with maximum support;
  $C_{k+1} = \emptyset$;
  For each pair of sets S, S' in C_k with $k - 1$ elements in common and k^{th} element with distinct attributes, add $S \cup S'$ to C_{k+1};
  $k = k + 1$;
end while
return S_{max};

Once we have identified the maximal set S of (attribute, position) pairs with support $\geq \beta$, we extract entity records by extracting attribute values at the specified positions in S from the pages of web site W. Example B.1 in Appendix B highlights several aspects of FINDATTRPOS when applied on the dataset in Figure 1.

4. EXPERIMENTAL EVALUATION

In this section, we present experimental results with real-life web datasets which demonstrate the effectiveness of our content matching-based extraction approach. Specifically, we show that our strong similarity metric and multi-attribute matching technique result in high-precision extractions while ensuring adequate web site coverage.

4.1 Experimental Setup

Dataset	# seed records	# attributes	# test sites	# pages
Restaurant	40000	5	17	984992
Bibliography	40000	3	7	1299329

Table 1: Dataset summary.

Datasets: We use two real-life datasets covering two verticals: *restaurant* and *bibliography*. Each dataset consists of a set of seed records and crawled pages from a set of test sites. We use the seed records to extract from the single-entity pages belonging to each of the test sites, and report the precision and coverage of the extractions.We classify attributes into core and non-core. Core attributes are present in every page belonging to the test dataset, while non-core attributes are optional.

The seed data for restaurants is obtained from `chefmoz.com`. Data from chefmoz is available as structured data in RDF format. Extractions are performed on 17 sites. The attributes extracted are: 1. restaurant name (core) 2. address (core), 3. phone, 4. payment, and 5. cuisine. The seed dataset consists of 40000 records randomly selected from chefmoz data.

For the bibliography dataset, we use data from DBLP as seed records. DBLP data is available in XML format. The seed dataset consists of 40000 records randomly selected from this dump. 7 sites are used as test sites. The following attributes are selected from the DBLP dataset: 1. title (core), 2. author (core), and 3. source.

Table 1 summarizes the details of the datasets. Appendix C provides additional details.

Metrics: We use precision and coverage as the primary metrics to evaluate the quality of the extractions. Since the datasets we use are very large, generating the complete ground truth editorially is a daunting task. Hence, we choose a random set of 1000 pages from each dataset, and generate the ground truth for this set. Precision metrics are reported on this random set.

We define the *coverage* for a dataset as the fraction of pages in the dataset from which we are able to extract core attributes.

Extraction Schemes: We use the FINDATTRPOS procedure described in Section 3 to compute the attribute positions (from which values are extracted) for each test site. We set the support parameter α for strong similarity computation to 0.1, and use $\beta = 10$ to prune attribute position configurations with inadequate support. We fix the weak similarity threshold T_w at 0.5, and vary T_s, the strong similarity threshold, between 0.5 and 0.9 in our experiments.

In order to compare the quality of extractions using weak and strong similarity, we consider a variant of FINDATTRPOS which we refer to as FINDATTRPOSW. FINDATTRPOSW is identical to FINDATTRPOS except that it uses weak similarity (instead of strong similarity) to determine the matching attribute values between seed records and web pages. Thus, in FINDATTRPOSW, for a set S of (attribute, position) pairs, $sup(S)$ is the number of distinct pages in $\cap_{(a,x)\in S} WS(a,x)$. We set the minimum support parameter β to 10.

Platform: All the experiments were performed on a shared Hadoop 0.20 cluster. The execution times reported are based on the number of map/reduce tasks and the average time of the map/reduce tasks.

4.2 Experimental Results

Effectiveness of Strong Similarity. In order to gauge the impact of strong similarity, we compare the precision and coverage of extractions generated by FINDATTRPOS and FINDATTRPOSW. Figure 2 plots these for the 2 datasets as T_s for FINDATTRPOS and T_w for FINDATTRPOSW are increased from 0.5 to 0.9 (T_w for FINDATTRPOS is fixed at 0.5). In the plots, we use suffixes SS and WS to qualify the precision and coverage metrics of procedures FINDATTRPOS and FINDATTRPOSW, respectively. It can be seen that the extraction precision increases with the similarity threshold for both the techniques. There is a significant coverage drop for FINDATTRPOSW at high (> 0.7) weak similarity threshold values. Strong similarity-based matching, on the other hand, provides both high precision and high coverage at higher strong similarity threshold values (> 0.7). This is because strong similarity boosts the similarity scores between diverse representations of the value of an attribute for the same entity which otherwise have low weak similarity scores. As a result, at the higher threshold values, true matches are retained (leading to high coverage) and false matches are pruned (leading to high precision). In fact, it is interesting to observe that FINDATTRPOS consistently has high coverage over the entire range of T_s values between 0.5 and 0.9.

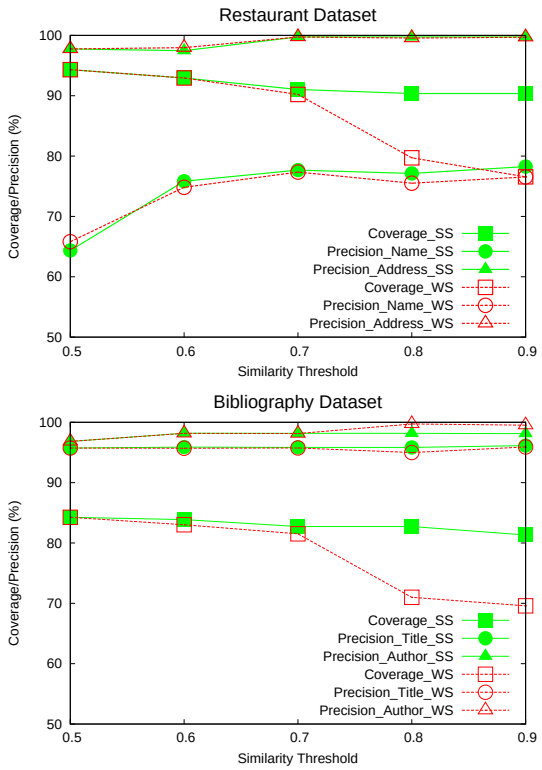

Figure 2: Precision/coverage of weak and strong similarity-based extraction of core attributes.

Figure 3 plots the strong and weak similarity scores for 200 address pairs between the seed database and the test sites. All of the address pairs have weak similarity scores exceeding 0.5, and are classified by hand into true and false matches. It is easy to see that the weak similarity scores of both true as well as false matches are distributed between 0.5 and 0.9. Thus, with weak similarity, there are true matches with low scores and false matches with high scores. This makes it difficult to find a threshold value that cleanly separates the true matches from the false ones. In contrast, for several true matches, the strong similarity scores are boosted close to 1 *even from very low weak similarity scores*. Thus, with a high enough strong similarity threshold (≈ 0.9), we can identify the true matches while filtering out the false ones.

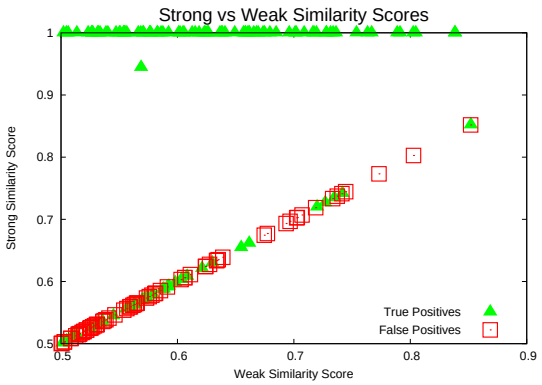

Figure 3: Scatter plot of strong similarity vs weak similarity scores. True matches are shown as green triangles and noisy matches as red squares.

Restaurant		Bibliography	
Attribute	Precision	Attribute	Prec
Name	78.26	Title	96.14
Address	99.74	Author	98.12
Phone	100.00	Source	100.00
Payment	100.00		
Cuisine	100.00		

Table 2: Precision of extractions for all attributes.

Extraction of Non-Core Attributes. Next, we look at the precision metrics for all of the core and optional attributes for both the datasets (see Table 2). We only consider strong similarity for extraction, and set the parameter values $T_s = 0.9$ and $T_w = 0.5$ in our FINDATTRPOS extraction procedure since these settings yield the best results across the 2 datasets. As can be seen, the precision for most of the attributes is above 95% and the coverage of core attributes for both the datasets exceeds 80%.

The precision of the name attribute in the restaurant dataset is somewhat low because of the presence of long lists of "Nearby Restaurants" in the restaurant pages of `www.tripadvisor.com`; this results in false matches with the name attribute values in seed records.

Effectiveness of Multi-attribute Matching. In order to quantify the amount of filtering achieved due multi-attribute matching in procedure FINDATTRPOS, we track the number of attribute positions in the generated candidate sets C_k. Let $C_k(a)$ be the set of *distinct* positions (DOM tree paths) for an attribute a in C_k. The decay in the number of distinct positions $|C_k(a)|$ as a function of k indicates the efficiency of multi-attribute matching. Table 3 lists the number of distinct paths for the core attributes in both the datasets as a function of k. Observe that a majority of the attribute positions involved in spurious matches are pruned within two iterations. This indicates that considering pairs of attributes when matching values can substantially improve matching accuracy.

	Restaurant		Bibliography	
	Name	Address	Author	Title
$k = 1$	1002	387	254	694
$k = 2$	113	64	34	39

Table 3: Number of positions for core attributes in multi-attribute matching.

Seed size	2000	5000	10000	20000	40000
Coverage (%)	53.57	59.73	61.06	69.40	90.37

Table 4: Coverage of extractions vs seed set size for *restaurant* data.

Stage	Restaurant	Bibliography
Weak Similarity	23987	17488
Strong Similarity	163	15
Multi-attribute Matching	22	5
Extraction	65	26
Total	**24237**	**17534**

Table 5: Running time (in hours) of different steps for the two datasets.

Impact of seed database size. We next study the effect of seed database size on the coverage of core attribute extraction. Table 4 shows coverage of strong similarity-based extractions at $T_s = 0.9$ for the restaurant dataset as the number of seed records is increased from 2000 to 40000. As can be seen, coverage jumps from 53.57% to 90.37% due to higher content overlap between seed records and web pages at the larger seed set sizes.

Execution Times. Table 5 provides the execution times of different stages. It can be seen that weak similarity computation dominates the execution time despite the use of prefix filtering [7] in our implementation. A complete run involving all the stages can be done on a 1000 CPU cluster in 1 day for the restaurant dataset and in 0.73 days for the bibliography dataset. Note that the execution time for the restaurant dataset is more than that for the bibliography dataset. The reason is that pages in the restaurant dataset are structurally more complex than those in the bibliography dataset: the average number of nodes per page for the restaurant dataset is 2.5 times more than that for the bibliography dataset.

5. RELATED WORK

Web information extraction. In recent years, a number of research papers [16, 20, 9, 25, 14, 11, 13] have studied the problem of extracting structured data from web pages. Early proposals for extracting structured data from the web were based on wrapper induction [16, 20]. These require human editors to annotate pages from each site and thus have high overhead. In recent years, there has been a flurry of research activity on extraction techniques that incur little manual effort. [9, 25, 11] devise methods to detect repeated patterns of tags within a web page and use this to extract records from the page. In [5, 1], attribute models based on *Hidden Markov Models* (HMMs) are learnt from training data, and these are used to segment short text strings like addresses and bibliographic entries. Web pages have a more complex hierarchical structure, and Zhu et al. [14] propose *Hierarchical Conditional Random Fields* (HCRFs) to label attribute values in web pages. An HCRF is a graphical model that captures both hierarchical and sibling dependencies in the tree corresponding to a web page. *Markov Logic Networks* (MLNs) [21] go a step further and allow relationships between arbitrary tree nodes to be expressed as first-order formulas. MLNs are used in [13] to extract structured data from web forum sites. These models rely on structural features of pages (e.g., phone numbers follow address values) and content features of attributes (e.g., 5-digit numbers correspond to zip codes) to label attribute values.

The precision of machine learning models may be poor in web environments due to the heterogeneity in web page structure and

attribute content formats, and noise in web pages. Our extraction approach overcomes these problems by exploiting content redundancy across sites, and uses the actual extracted attribute values to find matching values within web pages. Thus, we circumvent the difficult problem of building models that can capture the diverse structural and content formats prevalent across web sites.

Finally, there is a body of work on iteratively growing seed sets of relation instances and patterns for relation extraction. This is done by finding occurrences of the seed data in the corpus, discovering patterns, and matching the patterns to augment the seed data (see, e.g., [6, 2]). The use *templatized page and content structure in a site* distinguishes our approach from these techniques.

Approximate string matching. At its core, our extraction approach depends on being able to approximately match the attribute values for an entity across multiple sites. Fortunately, a number of approximate string matching algorithms have been proposed for detecting duplicate records in databases, text searching in the presence of spelling errors, etc. Comprehensive surveys of approximate string matching techniques can be found in [10, 15].

Existing similarity functions for string matching take as input two strings, and return a similarity score that quantifies the match between them. A popular measure used to gauge the similarity between two strings is the string edit distance. The edit distance metric works well for typographical errors but it cannot capture word rearrangements, insertions, and deletions. To address this, numerous variants of the edit distance metric have been proposed in the literature like affine gap distance [24] that allows gap mismatches, block edit distance [18] that allows word moves, and a fuzzy match similarity function that allows words to be inserted/deleted with a cost equal to the IDF weight of the word [22]. However, most variants either do not handle word rearrangements well, or are too expensive from a computation perspective. For instance, finding the exact block edit distance between two strings is an NP-hard problem [18].

The WHIRL [8] system adopts a different approach based on Cosine similarity between IDF-weighted words which it borrows from the IR literature. Unfortunately, while Cosine similarity can handle word swaps and weighs words based on their importance, it is less resilient to word misspellings. To alleviate this problem, Gravano et al. [17] propose a similarity metric that computes the Cosine similarity between IDF-weighted q-grams (instead of words). This metric has a number of desirable properties – it is capable of handling both word reorderings as well as spelling errors, and is computationally efficient.

Our notion of weak similarity also employs q-grams and is a variant of the similarity function proposed in [17]. Further, our strong similarity metric adds a new dimension by also taking into account the template structure when matching strings. Unlike previous similarity functions, it takes as input two sets of string values, and refines similarity scores based on the matching pattern between value pairs from the two sets.

A bulk of the previous work has focused on using the above string similarity functions to match records with multiple attributes. [4, 23] train classifiers to combine the multiple attribute-level similarity scores into a single record-level similarity score, while [17, 22] simply extend the edit distance and Cosine similarity variants to work at the granularity of records as opposed to individual attributes. In contrast, in our web extraction scenario, we are interested in finding values within unstructured web pages that match attribute values within a record. Our problem setting is a lot more challenging due to the presence of noise in web pages; our proposed solutions filter out the noisy attribute value matches by exploiting the template structure of attribute content and web pages.

6. CONCLUDING REMARKS

In this paper, we proposed a novel approach that exploits overlapping content across web sites and the template structure of web pages to extract structured data from the web. We defined a new similarity metric for matching previously extracted attribute values with the content in a fresh page. Our new metric takes into account the matching pattern between attribute values from two sites to refine similarity scores for differently formatted attribute values belonging to the same entity. We also developed an Apriori-style algorithm for efficiently enumerating attribute positions with matching values in a sufficient number of pages. In our experiments with real-life web data sets, our techniques were able to extract records with $> 95\%$ precision and $> 80\%$ recall. An important direction for future work involves extending our methods to handle non-text numeric (e.g., price) and image (e.g., ratings) attributes.

7. REFERENCES

[1] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In SIGKDD, 2004.

[2] E. Agichtein and L. Gravano. Snowball: extracting relations from large plain-text collections. In ACM DL, 2000.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In SIGMOD, 1994.

[4] M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In SIGKDD, 2003.

[5] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In SIGMOD, 2001.

[6] S. Brin. Extracting Patterns and Relations from the World Wide Web. In WebDB, 1998.

[7] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In ICDE, 2006.

[8] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In SIGMOD, 1998.

[9] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In VLDB, 2001.

[10] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. IEEE TKDE, 2007.

[11] G. Miao et al. Extracting data records from the web using tag path clusterting. In WWW, 2009.

[12] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In WWW, 2005.

[13] J. Yang et al. Incorporating site-level knowledge to extract structured data from web forums. In WWW, 2009.

[14] J. Zhu et al. Simultaneous record detection and attribute labeling in web data extraction. In SIGKDD, 2006.

[15] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: Similarity measures and algorithms. In SIGMOD (Tutorial), 2006.

[16] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In IJCAI, 1997.

[17] L. Gravano et al. Text joins in an RDBMS for web data integration. In WWW, 2003.

[18] D. Lopresti and A. Tomkins. Block edit models for approximate string matching. Theoretical Computer Science, 181(1), 1997.

[19] C. Manning, P. Raghavan, and H. Schutze. Introduction to Information Retrieval. Cambridge University Press, 2008.

[20] I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semistructured information sources. Autonomous Agents and Multi-Agent Systems, 1(2), 2001.

[21] M. Richardson and P. Domingos. Markov logic networks. Machine Learning, 62(1), 2006.

[22] S. Chaudhuri et al. Robust and efficient fuzzy match for online data cleaning. In SIGMOD, 2003.

[23] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In SIGKDD, 2002.

[24] M. Waterman, T. Smith, and W. Beyer. Some biological sequence metrics. Advances in Math., 20(4), 1976.

[25] Y. Zhai and B. Liu. Web data extraction based on partial tree assignment. In WWW, 2005.

APPENDIX

A. WEAK SIMILARITY

Q-grams. The q-gram set of a string is the set of all q-length substrings of the string. We denote the q-gram set of a word w by $qg(w)$. To ensure that characters at the start and end of the word w appear in a sufficient number of q-grams, we pad w at the beginning and end with $q-1$ occurrences of a special padding character #. So for $q = 3$, qg("china") = {"##c", "#ch", "chi", "hin", "ina", "na#", "a##"}. Next, we derive the q-gram set for an attribute value by taking the union of the q-grams in the words comprising the value. More formally, for a value v, the q-gram set $qg(v) = \cup_{w \in v} qg(w)$. It is easy to see that q-grams can cope with spelling errors better than whole words. For instance, the Jaccard similarity between the strings "beijing bites" and "bejing bites" is only $\frac{1}{3}$ in Example 1.1 because "beijing" and "bejing" are treated as separate words. But with q-grams instead of words, the Jaccard similarity is more than $\frac{2}{3}$.

Q-gram weights. We associate a weight with each q-gram in $qg(v)$ based on the importance of the word that it originates from. Here, we adopt the popular *inverse document frequency* (IDF) weight from the IR literature [19] to capture the importance of each word w that appears in attribute a of a seed record. For a word, attribute pair (w, a), we define $IDF_a(w)$ as $\log \frac{N}{N_{a,w}}$, where $N = |R|$ is the number of records in the seed database R and $N_{a,w}$ is the number of records in R for which the attribute a contains w. Informally, the IDF score of a word decreases as its frequency increases, and so common words have low IDF scores. Note that the IDF value of a word can vary depending on the attribute. For example, the word "avenue" has a low IDF score if it is part of the address attribute, but a much higher score if it belongs to the name attribute. For clarity of presentation, we will drop the subscript for attribute a when it is clear from context.

We assign a weight $c_v(e)$ to each q-gram e in $qg(v)$ equal to the sum of the IDF weights of all the words in v that contain e. More formally, let $e_1, \ldots, e_l$ be the instances of q-gram e appearing in words $w_1, \ldots, w_l$ of v. Then $c_v(e) = \sum_{i=1}^{l} IDF(w_i)$. The intuition here is that q-grams should inherit importance scores from the words that they originate from. Thus, by associating the IDF scores of words as the weights of q-grams in them, we can ensure that q-grams belonging to important words are assigned higher weights. For example, the q-gram "ave" in the word "avenue" should have a lower weight compared to its weight in "davenport" which is less common. This is in contrast to previous approaches [17] that assign a single IDF weight to each q-gram based on its individual frequency irrespective of the frequencies of the words that contain it. So for instance, in [17], a frequent q-gram like "ave" will end up with a single low weight irrespective of whether it belongs to a rare word like "davenport" or a common word like "avenue".

Similarity score. We can conceptually map each value v into a vector in q-gram space, with the component in the dimension corresponding to q-gram e in $qg(v)$ set to $c_v(e)$. We then define the similarity between values u and v as the widely used Cosine similarity metric between their corresponding vectors in q-gram space.

DEFINITION A.1. *Given a pair of values u and v for attribute a, the similarity $sim_a(u, v)$ is given by*

$$sim_a(u,v) = \frac{\sum_{e \in (qg(u) \cap qg(v))} c_u(e) \cdot c_v(e)}{\sqrt{\sum_{e \in qg(u)} c_u(e)^2} \cdot \sqrt{\sum_{e \in qg(v)} c_v(e)^2}}$$

Observe that the above similarity function $sim_a(u, v)$ that returns a similarity score between 0 and 1. We refer to this as the *weak similarity* score between u and v. Further, we say that values u and v are *weakly similar* if the similarity score between the values is greater than or equal to a weak similarity threshold T_w. Else, they are considered to be dissimilar. In case an attribute is missing from a page, we use NULL to represent the value of the missing attribute. If either of the attribute values u or v is NULL, then the similarity score between the values is 0.

Typically, we need to compute the similarity between two values u and v of an attribute a, where one is an attribute value in a seed record and the other is text within a web page. It may happen that the IDF value for a word w within the value v from a page is not defined since it is not part of the attribute in the seed database. For such words, we set $IDF_a(w)$ to the IDF weight of the closest word w' appearing in the attribute a of seed records. Here, we use the Jaccard similarity between q-gram sets to measure closeness between w and w'; thus, w' is the word with the maximum $JC(qg(w), qg(w'))$ score. In the event that there are multiple words w' with the max $JC(qg(w), qg(w'))$ value, then we set $IDF_a(w)$ to the average of their IDF weights.

B. EXAMPLE FOR FINDATTRPOS

The following example illustrates the workings of Algorithm 3.

EXAMPLE B.1. *Consider the seed database R and pages p_1, p_2 in Figure 1. Let us number the positions for name and address values at the top of the pages as 1 and 2, respectively, and the positions for values under "Nearest Transit:" and "Related Restaurants:" as 3 and 4, respectively. Now, suppose that the (record, page) pairs with strongly similar values for the various (attribute, position) pairs are as follows:*

- $SS(name, 1) = \{(r_1, p_1), (r_2, p_2)\}$.
- $SS(name, 4) = \{(r_2, p_1), (r_1, p_2)\}$.
- $SS(address, 2) = \{(r_1, p_1), (r_2, p_2)\}$.
- $SS(address, 3) = \{(r_1, p_1)\}$.

The first bullet states that the name values in records r_1 and r_2 are strongly similar to the values in position 1 in pages p_1 and p_2, respectively. Similarly, it follows from the final bullet that the address value in record r_1 is strongly similar to the value in position 3 in page p_1. Let the minimum support parameter $\beta = 1$. Observe that the support of all four (attribute, position) pairs above is at least β, and so they will be added to C_1.

In the second iteration, the following four sets of (attribute, position) pairs will be added to C_2: $S_1 = \{(name, 1), (address, 2)\}$, $S_2 = \{(name, 1), (address, 3)\}$, $S_3 = \{(name, 4), (address, 2)\}$, and $S_4 = \{(name, 4), (address, 3)\}$. Of these only S_1 has support 2 (since name and address values in records r_1 and r_2 will match the values in positions 1 and 2 in pages p_1 and p_2, respectively). S_2 has support 1 (since name and address values in only record r_1 will match the values in positions 1 and 3 in page p_1), and both S_3 and S_4 have zero support. Thus, all but S_1 and S_2 will be pruned from C_2 due to lack of support, and the set S_1 will be returned by procedure FINDATTRPOS *since it has the maximum support of 2.* □

C. DATASET DETAILS

The following table lists the 17 test sites in the restaurant dataset along with the number of pages. The sites were a combination of head (sites with a few hundreds of thousands of pages) and tail

(sites with a few thousands of pages) sites.

Site	Number of Pages
cityguide.aol.com	81038
dinesite.com	1186
tupalo.com	4245
www.8coupons.com	1700
www.agoda.com	13485
www.blogsoop.com	7413
www.city-data.com	17152
www.insiderpages.com	32205
www.menupages.com	9207
www.opentable.com	17423
www.restaurantrow.com	46494
www.savorycities.com	2613
www.travelmuse.com	3323
www.tripadvisor.com	421275
www.urbanspoon.com	46293
www.yellowbot.com	106572
www.yelp.com	173368
Total	984992

For the bibliography dataset, we use the following 7 head and tail sites as test sites.

Site	Number of Pages
arxiv.org	119564
citeseerx.ist.psu.edu	51573
ieeexplore.ieee.org	321325
libra.msra.cn	2282
portal.acm.org	21897
www.citeulike.org	778130
www.sciencedirect.com	4558
Total	1299329