The Paths More Taken: Matching DOM Trees to Search Logs for Accurate Webpage Clustering

Deepayan Chakrabarti
Yahoo! Research
Santa Clara, CA
deepay@yahoo-inc.com

Rupesh R. Mehta
Yahoo! Labs,
Bangalore, India
rupeshm@yahoo-inc.com

ABSTRACT

An unsupervised clustering of the webpages on a website is a primary requirement for most wrapper induction and automated data extraction methods. Since page content can vary drastically across pages of one cluster (e.g., all product pages on `amazon.com`), traditional clustering methods typically use some distance function between the DOM trees representing a pair of webpages. However, without knowing which portions of the DOM tree are "important," such distance functions might discriminate between similar pages based on trivial features (e.g., differing number of reviews on two product pages), or club together distinct types of pages based on superficial features present in the DOM trees of both (e.g., matching footer/copyright), leading to poor clustering performance.

We propose using search logs to automatically find paths in the DOM trees that mark out important portions of pages, e.g., the product title in a product page. Such paths are identified via a *global* analysis of the entire website, whereby search data for popular pages can be used to infer good paths even for other pages that receive little or no search traffic. The webpages on the website are then clustered using these "key" paths. Our algorithm only requires information on search queries, and the webpages clicked in response to them; there is no need for human input, and it does not need to be told which portion of a webpage the user found interesting. The resulting clusterings achieve an adjusted RAND score of over 0.9 on half of the websites (a score of 1 indicating a perfect clustering), and 59% better scores on average than competing algorithms. Besides leading to refined clusterings, these key paths can be useful in the wrapper induction process itself, as shown by the high degree of match between the key paths and the manually identified paths used in existing wrappers for these sites (90% average precision).

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous

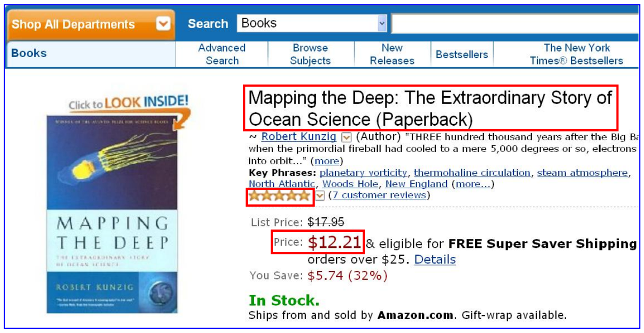
General Terms

Algorithms

Keywords

Wrapper induction, Clustering, Search logs

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.



Mapping the Deep: The Extraordinary Story of Ocean Science (Paperback)

~ Robert Kunzig (Author) "THREE hundred thousand years after the Big Ba
when the primordial fireball had cooled to a mere 5,000 degrees or so, electrons
into orbit..." (more)

Key Phrases: planetary vorticity, thermohaline circulation, steam atmosphere, North Atlantic, Woods Hole, New England (more...)

★★★★★ (7 customer reviews)

List Price: ~~$17.95~~

Price: $12.21 & eligible for **FREE Super Saver Shipping** orders over $25. Details

You Save: $5.74 (32%)

In Stock.

Ships from and sold by **Amazon.com**. Gift-wrap available.

Figure 1: A webpage and its key portions.

1. INTRODUCTION

The Web provides unstructured data while the databases underlying most information systems prefer structured data. This basic mismatch has fueled a lot of work on data *extraction* from the Web, so that a service like `pricegrabber.com` can extract structured data from webpages, store it in a database, and then allow powerful queries into it, such as searches by manufacturer, or store, or limited price ranges, etc. The data gathering is typically handled by a *wrapper* that, given a webpage, can extract informative fields from it by following certain pre-defined paths in its DOM tree. However, building wrappers is a labor-intensive task requiring manual labeling of informative sections on a few example webpages, after which an induction process automatically infers paths in the DOM tree that point to those sections [15]. Since a website will typically have several types of pages, multiple wrappers need to be learnt, and most wrapper induction work assumes that some oracle can readily provide example pages of each type.

In practice, however, such example pages must themselves be obtained in a pre-processing step. Scalability concerns rule out manual exploration of websites to find webpages of each type. The typical approach is to cluster the webpages, and then declare each cluster to be a distinct type. Any errors in this clustering stage, however, can have a very significant effect on the quality of wrapper induction. If pages of the same type are split into multiple clusters, manual labeling and wrapper induction will have to be performed for each, leading to scalability and cost issues. Worse still, if a cluster is heterogeneous, then it might be impossible to learn one wrapper for all pages in the cluster, leading to loss of data or costly reprocessing. Thus, the quality of the clustering plays a crucial role in the final quality and comprehensiveness of the extracted data.

Pages of the same type on a website could have very different content (e.g., all "product" pages on `amazon.com`), implying that clustering based on content is unlikely to succeed. Instead, we need webpages with similar *structure*, possibly generated by the same server-side script. As each webpage can be represented by a DOM tree, the obvious choice is to cluster using some tree-matching technique that can account for minor variations between trees. However, pre-existing methods such as Path-shingles [5] and pq-Grams [1] consider all nodes and paths in the DOM tree as equally important. This can lead to pages being matched due to superficial features (e.g., both contain ads/copyright notices) or incorrectly being split apart (e.g., differing number of reviews on two product pages). We remedy this by leveraging search logs to identify "key" paths in the DOM trees, and using these to match DOM trees and cluster the webpages.

The key paths on a webpage are those whose contents provide the basic information desired by most readers of that webpage (Figure 1). Consequently, these are also the most important paths for data extraction and wrapper induction. For accurate clustering, we need to find at least one key path for each webpage type in a website: if we cluster all pages possessing a *product title* path (as in Figure 1), these will automatically be good example pages for learning wrappers to extract *price* and *rating* as well. The search logs aid us by pinpointing terms in webpages that users care the most about (the query terms), and our algorithms match query terms to path contents, over all queries and webpages, to identify a global set of key paths for the entire website.

Note that this work is not simply about finding the informative sections of a page, but rather about paths that are markers for pages of a certain type. Key paths typically come from informative sections, but not all paths in the informative sections are key paths.

Our contributions. In this paper, we present a completely unsupervised approach to find key paths, that can subsequently be used to cluster pages on the website. Our contributions are as follows.

(1) Our algorithm performs a global analysis of the entire site, using searches on popular pages to infer the key paths applicable even to pages with little or no search activity.

(2) We study the problem of dependence between key paths, for instance when they co-occur too frequently or their content is too similar. Such dependence can lead to incorrectly split clusters. We propose a weighted independent set formulation to pick the best independent key paths in such a setting. This technique is applicable to large-scale feature selection in general, when the features have associated rankings or goodness scores.

(3) We demonstrate the accuracy of our algorithm empirically on 10 real-world websites, where the automatically generated clustering matches the ground truth with adjusted RAND scores of 0.9 or higher on 5 sites (a score of 1 indicates perfect match), and outperforming the best competing baseline by 59% on average.

(4) The key paths found by our algorithm also match many of the paths found manually via a separate wrapper induction system (90% average precision), again over the 10 websites. Thus, the key paths can not only provide cues for human labelers in the ensuing wrapper induction process, but also makes the clusters easily interpretable.

Organization. After a discussion of related work in Section 2, we describe our proposed method in Section 3. The accuracy of our key paths, and the accuracy of the corresponding clustering, are demonstrated in Section 4. Finally, we conclude in Section 5.

2. RELATED WORK

There has been a lot of work on wrapper induction as well as on matching webpages (and trees in general) based on their structure. We split up our discussion along these two lines.

Wrapper Induction. Given a few example pages and the tuples of information that can be extracted from them, wrapper induction is the problem of learning a *wrapper* that can then be applied to other pages of similar structure to extract the same tuples. There has been a lot of work on this problem [15, 8, 14, 17, 12, 7]. All of these require a few example pages to be specified, from which learning can occur. Our goal is to automatically create a clustering such that each cluster corresponds to a wrapper, and pages from each cluster can serve as examples for wrapper induction. Thus, our work would be useful in all such approaches.

Fully automated means of wrapper induction have also been developed [6], but these can extract data from pages where information is repeated in a similar structure, such as lists. However, in more general cases (say, a product page with one title and one price), one must still revert to the traditional semi-supervised techniques and hence need a clustering of the webpages.

Webpage Clustering. Methods for structural clustering of webpages have used similarity metrics based on the Fourier transform [11], distance functions over DOM trees [18, 22], URL patterns [9], and similarity of HTML tag or path [9]. Tree-matching techniques have also been studied outside the context of clustering DOM trees [20, 21, 16]. A few methods combine the processes of webpage clustering and wrapper induction [22, 8]. A survey of algorithms for document structure similarity and clustering is presented in [5].

While all of these are relevant to our work, we ideally need clustering algorithms that are extremely fast, and do not require pairwise distance measurements between all pairs of trees. We discuss a few such methods below.

Buttler [5] proposed a linear time technique to measure structural similarity between any pair of webpages by maintaining a shingle [4] of the structure of each webpage. This shingle consists of just a few paths in the webpage; hence we call it Path Shingles. He showed that this simple technique was not only the most efficient, but also accurate when compared to other similarity metrics, such as tree edit distance and Fourier-transform based distances. Due to its mix of efficiency and accuracy, we build on this basic scheme for our clustering algorithm.

Augsten et al. [1] proposed an approximate tree matching technique requiring $O(n \log n)$ time. Their method creates a sketch of a tree using "pq-Grams", which are defined as subtrees of a specific shape. The similarity between two trees is obtained by computing the pq-Gram distance, which is based on the number of common pq-Grams between two trees. pq-Grams provide a strong baseline against which to compare our work.

All of these methods treat all nodes and paths as equally important, and this can lead to heterogeneity or cluster splitting issues, depending on the site structure. As against all of these methods, we incorporate search query logs in our clustering method. This allows us to find paths and nodes in the DOM tree that are the most relevant to users, and we bias our clustering and distance functions towards these paths. As we show empirically, our technique achieves high accuracies in clustering the webpages in 10 websites, and yields significant improvements over both Path Shingles and pq-Grams. To the best of our knowledge, ours is the first technique to use web search logs to identify key paths and use them for structural clustering of web documents.

3. PROPOSED METHOD

Our goal is to cluster the webpages from a given website so that structurally similar pages are placed together in the same cluster. Note that pages in a cluster are allowed to be topically distinct. The structure of a webpage is given by its DOM tree; equivalently, the page can be represented as a set of paths from the root (`<html>`) to every node in the DOM tree. In addition, we have search logs which list the queries for which webpages from this website were returned as search results and clicked on by users.

Main Idea. There are three keys to our proposed method: (a) the terms included in search queries are the primary terms of interest to users, (b) a webpage clicked in response to a query often contains the query terms in its main content section, and hence, (c) the paths whose content includes such query terms frequently are good candidates for "key" paths. Formalizing this intuition is the first challenge.

The second challenge arises from the fact that paths can be *dependent*. An obvious example of this is when some path x_1 is a prefix of another path x_2: whenever x_2 exists on a page p, so must x_1, and the content $C(x_1, p)$ of x_1 in p must be a superset of $C(x_2, p)$. Thus, x_1 must always match at least as many query terms as x_2. Another example is when the same information is mentioned in multiple places in the webpage (e.g., the product title is mentioned in a header and in the "features" section). Even though the paths are not related, their contents are similar over many webpages. Two paths x_1 and x_2 can also be dependent if they occur on the same set of pages, even if their content is different (e.g., average rating only occurs on pages with a product title). Intuitively, one can use either of x_1 or x_2 as a key path to distinguish webpages of this "type" from other pages, and having both as key paths confers no extra advantages. In fact, it can *hurt* the clustering quality: for instance, if paths $x_1, \ldots, x_n$ mostly co-occur in one large cluster of pages but occasionally one or the other is missing on a few pages, then having all of them as key paths would lead to the break-up of the large cluster into one set of pages which contain all the paths, plus many small chunks, each of which lacks one particular x_i. In this case, it is better to pick just one path as a key path, creating just one large cluster. Thus, to achieve the best clustering, we must ensure that the key paths are *independent*.

Finally, once these key independent paths have been identified, the webpages must be quickly clustered according to the presence or absence of these paths in their DOM trees.

Thus, the three steps in our algorithm are:

(1) Find key paths for a website

(2) Remove dependence by picking the best set of independent key paths

(3) Cluster webpages using these independent key paths

The bulk of our work is in steps (1) and (2), so we present step (3) first in Section 3.1 and then focus on the key path identification problem in Sections 3.2 and 3.3. Finally, in Section 3.4, we will discuss some optimizations that improve the accuracy of the clustering.

Notation. As noted above, we use x_i to represent paths in the DOM tree, and also the nodes pointed to by those paths. Note that these need not be leaf nodes. We use $x_i = 1$ to denote the presence of x_i on some page, and $x_i = 0$ to denote its absence. If $x_i = 1$ on page p, then p contains an *instance* of x_i, and $C(x_i, p)$ represents the bag of terms in the content of x_i on p. When multiple instances of x_i exist on the same page, $C(x_i, p)$ contains the union of their contents. Associated with page p is the (possibly empty) set of queries $Q(p)$ for which p was returned as a search result and was clicked; depending on the context, $Q(p)$ also denotes the bag of query terms in the query set. The set of *matching* terms of x_i on p is given by $M(x_i, p) = C(x_i, p) \bigcap Q(p)$.

3.1 Webpage Clustering Using Key Paths

Every clustering technique requires some notion of similarity, either between two points, or between a point and a cluster. Our points are webpages, each of which can be represented as a set of paths. We do not count repetitions of a path on a page because pages of the same type can often have different number of repetitions for the same path, e.g., varying number of reviews/comments on a product page. A well-known similarity function on sets is Jaccard similarity:

$$J(S_1, S_2) = \frac{|S_1 \bigcap S_2|}{|S_1 \bigcup S_2|},$$

which can be estimated quickly using shingles: (a) for each set, hash all elements and pick the k elements with the minimum hash values, and then (b) compute the Jaccard similarity between the shingles of S_1 and S_2, which is an unbiased estimator of the Jaccard similarity between the two sets [4]. A basic shingle-based clustering algorithm places all sets with identical shingles in the same cluster; more complicated variants allow sets sharing at least $k' < k$ shingle-elements to be in the same cluster. The basic shingle-based clustering scheme has the advantage that any new page can be mapped to its cluster in $O(1)$ time (once its shingle has been computed), leading to faster runtime performance in a production environment; hence, we modify this basic scheme to accommodate key paths.

The key path identification algorithm (described in the following sections) returns a ranked list of key paths, and similarity computations should be biased towards these. We propose a two-step shingle construction process: given a webpage with a set of paths W, we first pick the top-ranked key paths present in W, and if there are fewer than k of these, we then pick the remaining paths in the shingle via the usual min-hashing scheme described above. If more than k key paths are present in W, the shingle consists of the top-ranked k of these. Now, all pages with the same shingle are mapped to the same cluster. This allows pages containing key paths to be clustered on the basis of these paths, while defaulting to the traditional path-shingling approach [5] for pages without any key paths (say, "Help" pages, and others pages that are never queried for).

Thus, given a ranked list of key paths, shingle-based clustering can be used to group pages quickly, and to cluster newly crawled pages as well. Next, we look at the problem of generating this ranked list.

3.2 Identifying Key Paths

Our proposed method first computes the goodness of each path independently of others, and then picks the best set of independent paths from among these. We discuss the first step in this section, and the second step in Section 3.3.

We want paths that: (a) contain most of the query terms, (b) contain few or no terms that are not present in queries, and (c) occur frequently on pages that are clicked. The first two conditions correspond to the *recall* and *precision* of a path with respect to the query terms, while the third ensures that the path is relevant to a significant fraction of the query workload. We formalize these conditions in terms of two properties of a given path x: its *expected* precision and *expected* recall.

Expected Precision of x. Among all possible instances of x over all pages of the website, pick one uniformly at random. Let this page be p. The precision of this instance is $|M(x,p)|/|C(x,p)|$ if $Q(p)$ is non-empty, or zero otherwise. We multiply this precision by the probability of having picked this instance of x, and average this over all instances of x to get the expected precision $EP(x)$:

$$\begin{aligned}\text{Prec}(x,p) &= \begin{cases} \frac{|M(x,p)|}{|C(x,p)|} & \text{if } x_p = 1 \text{ and } Q(p) \neq \{\phi\} \\ 0 & \text{otherwise}\end{cases} \\ EP(x) &= \sum_p P(x_p = 1) \cdot P(Q(p) \neq \{\phi\}) \cdot \text{Prec}(x,p) \quad (1)\end{aligned}$$

The first term in Equation 1 is the fraction of pages on which x is present, and when combined with the second term, it gives the fraction of *queried and clicked* pages where x is present. Depending on the application, these factors could be given different relative weights as well.

Expected recall of x. The recall of an instance of x in page p is given by $|M(x,p)|/|Q(p)|$. Weighing this by the probability of picking this instance, and averaging over all instances, gives us the expected recall $ER(x)$:

$$\begin{aligned}\text{Rec}(x,p) &= \begin{cases} \frac{|M(x,p)|}{|Q(p)|} & \text{if } x_p = 1 \text{ and } Q(p) \neq \{\phi\} \\ 0 & \text{otherwise}\end{cases} \\ ER(x) &= \sum_p P(x_p = 1) \cdot P(Q(p) \neq \{\phi\}) \cdot \text{Rec}(x,p) \quad (2)\end{aligned}$$

Combining the two. A good path is one that scores highly on both of these, and we observe empirically that precision is more important than recall. The usual method of combining precision and recall is via the F1 score, which is the harmonic mean of the two; weighted versions of the F1 measure can be used to handle different weights for precision and recall. However, the scores themselves can vary considerably across websites, due to page structure (pages with more text per DOM node might have lower precision and higher recall on average), query workload distribution over pages, and other such factors. Any particular combination of these scores might not work on all sites. Instead, we formulate this as a rank-aggregation problem [10], as follows. We rank all paths according to expected precision, yielding a ranking R_P. We create another ranking R_R according to expected recall. Now, we combine these rankings into one ranking, using Borda's Method [3]: each path is assigned a score that is the sum of the positions of that path in R_P and R_R, and then all paths are ranked according to this score to yield a final ranking R_F. The relative importance of precision to recall can be incorporated into the score computation by scaling the position of each path in R_R by some constant factor β. Intuitively, this is the same as saying that gaining one rank in the precision ranking R_P is worth losing $1/\beta$ ranks in the recall ranking R_R (in our experiments, $\beta = 0.3$). Apart from easy interpretability, this method is also immune to differences in the typical precision and recall values between websites.

To summarize, we compute expected precision and expected recall for all paths in a website, and then perform rank aggregation to obtain one final ranking R_F of paths. However, each highly-ranked path might have several other dependent paths, which might also have similarly high ranks. We need to pick one set of paths that are highly-ranked according to R_F, and are independent of each other as well. We discuss this next.

3.3 Picking Independent Key Paths

We are given a ranking R_F of paths, and need to pick a subset S that will be used in the clustering algorithm discussed in Section 3.1 (the ranking on S is induced by the ranking R_F). Recall that the clustering algorithm picks up to k paths from a webpage, with preference being given to paths in S. As noted before, paths may be dependent, and having all of them in S can lead to unnecessary splitting of clusters and poor clustering accuracy. We will first show how we measure dependence between paths, and then present the algorithm to pick the set S of independent paths.

Measuring Dependence. Let us consider a few examples of dependent paths that occur in practice, which will naturally lead to our algorithm for picking S. Suppose there are two paths x_1 and x_2 that are both highly ranked according to R_F. Let $P(x_1)$ and $P(x_2)$ be the set of webpages in which x_1 and x_2 are present. Three cases occur frequently.

(1) $P(x_1) \approx P(x_2)$: There are two common instances of this. The first example is when x_1 is the parent or close ancestor of x_2. The second example is when x_2 points to the product title header, and x_1 to the product price. In either case, keeping both x_1 and x_2 in S will mean that the shingle for each page will contain either both x_1 and x_2, or neither. Clearly, in the former case, two out of the k available slots in the shingle are being used to represent the same information, leading to suboptimal shingling. Hence, either x_1 or x_2 should be kept in S, and not both.

(2) $P(x_1) \subset P(x_2)$, *similar content:* An example of this is when x_2 points to the product title header, while x_1 points to another mention of the product title, but in the "product features" section of the page. Since the features section does not exist on all pages, $P(x_1) \subset P(x_2)$. Now, if both x_1 and x_2 are placed in S, then the cluster of product pages would get split in two: the pages with the product title mentioned in features section, and the pages without. Clearly, x_2 does not provide extra information on top of x_1, and so such a split hurts the quality of the clustering.

(3) $P(x_1) \subset P(x_2)$, *different content:* An example of this is when x_2 points to the product title header, while x_1 points to the list of customer reviews, if any. Here, the decision to split or not is application-dependent. However, for the task of wrapper induction on product-centric websites, we find that such splits are unnecessary. One problematic case is if x_1 is ranked much higher than x_2 in R_F and gets picked in S: then, $P(x_1)$ and $P(x_2) \setminus P(x_1)$ must be in different clusters whether or not x_2 is placed in S. Our solution is to bias the precision and recall computations towards paths that occur more frequently (by assigning a high relative weight to the first term in Equations 1 and 2) so that x_2 is likely to be ranked higher than x_1 in R_F, and hence likelier to be picked in S. When this is not the case, it usually implies that x_1 is significantly more important than x_2, and $P(x_1)$ deserves to be in a cluster of its own.

These examples show that dependent paths can lead to incorrect splits of clusters. Moreover, the proper shingle size k for use in the clustering algorithm must then be adjusted for the degree of dependence in each website: if k is too small, the shingle will be filled up entirely by dependent paths, and if k is too large, there will be too many cluster splits. Finally, if dependent paths are not removed, then the set of key paths S becomes very large (by more than an order of magnitude). In a production environment where newly crawled pages from a large number of websites need to be clustered in a streaming fashion, having to maintain a large S in memory for each website can degrade runtime performance or introduce bottlenecks. Thus, S must contain only independent paths, and the previous discussion suggests two factors that are important in determining the degree of dependence.

Similarity of occurrence: Among all the pages where path x_1 is present, what is the fraction in which x_2 is present as well? This tells us the degree to which the set of instances of x_1 is *contained* within those of x_2. Considering this fraction, and its converse, gives us the similarity of occurrence $H(x_1, x_2)$:

$$H(x_1, x_2) = \frac{|x_1 = 1 \wedge x_2 = 1|}{\min(|x_1 = 1|, |x_2 = 1|)}.$$

High values of $H(.)$ imply that one path is mostly contained within the other, and so at most one of them can be present in S.

Similarity of content: The relevance of a path's content is given by the degree to which it matches query terms. Hence, we consider two paths to have similar content if they match similar query terms whenever they co-occur on the same page. We measure the degree of content similarity as just the average fraction of shared query terms:

$$\begin{aligned} F(x_1, x_2, p) &= \frac{|M(x_1, p) \bigcap M(x_2, p)|}{\min(|M(x_1, p)|, |M(x_2, p)|)} \\ F(x_1, x_2) &= E_{p \in P}\left[F(x_1, x_2, p)\right], \end{aligned}$$

where the expectation is taken over the set of pages P where both x_1 and x_2 are present. Higher values of $F(.)$ imply high degree of content similarity.

Thresholds can be used to determine the level of dependence that is suitable for the application. For the purposes of clustering for improving the subsequent wrapper induction process, we use only similarity in occurrence (similarity in content was empirically observed to be not useful, as observed in case (3) above). Thus, two paths x_1 and x_2 are dependent if $H(x_1, x_2) \geq \theta_{\text{OCC}}$ for some threshold θ_{OCC} (in our experiments, $\theta_{\text{OCC}} = 0.97$).

Picking Independent Paths. Given the relationships between paths, we need to pick the best set S of independent paths. We formalize this as a graph problem, as follows. The nodes of the graph are the set of top-k ranked paths in R_F (we empirically find that $k = 100$ is sufficient). Two nodes are connected by an edge if they are dependent. Since S must only contain pairwise-independent paths, no pair of nodes in S must have an edge connecting them. In addition, each node (path) contains some information relevant to queries on the website, so we want to cover as many of them as possible. Finally, since the set of nodes originally had a ranking R_F, we would like to stay true to it, and pick nodes preferentially from the top of the ranking. This same setting is also applicable to the broader problem of feature selection when we have some external data about the relative weights or rankings of the features, and would like to pick an independent set of features.

This problem can be formulated as the well-known *Maximum Weighted Independent Set* problem, as follows.

Problem 1 (Maximum Weighted Independent Set). *Given a graph $G = (V, W, E)$, where each node $i \in V$ has weight w_i, pick a set of nodes $S \subseteq V$ so as to:*

$$\begin{aligned} \textit{Maximize} \quad & \sum_{i \in S} w_i \\ \textit{subject to} \quad & (i, j) \notin E \quad \forall i, j \in S \end{aligned}$$

In our particular setting, we observe that the top-ranked paths are much more significant and useful than those lower down in the ranked list, so we propose an exponential weighting scheme for the paths. In effect, $w_i = \alpha^{R_F(i)}$, where $R_F(i)$ is the position of path i in R_F (the top-ranked path having position 1), and $\alpha < 1$ is the exponential weighting factor. Note that when $\alpha = 1$, all weights are equal and there is no extra reward for picking the top-ranked paths.

While the Maximum Weighted Independent Set problem is NP-Hard in general, for certain weight settings a greedy algorithm can get provably optimal solutions. We first present the algorithm and then the proof of optimality.

```
Algorithm Greedy_MWIS
  input: the weighted graph $G = (V, W, E)$
  Feasible set $F = V$; Result set $S = \{\phi\}$
while $F \neq \{\phi\}$
  new ← $\arg\max_{i \in F} w_i$            // Pick top-ranked feasible node
  $S \leftarrow S \cup \{\text{new}\}$                          // Update result set
  $F = F \setminus (\{\text{new}\} \cup \{i \mid (\text{new}, i) \in E\})$        // and feasible set
end while
```

The greedy algorithm essentially picks the highest-weighted node that is still feasible at every stage, and continues until no nodes can be added. When this happens, every node that is not in S is connected to some node in S, and so every vertex in V is "covered." Next, we see that this is optimal for some values of the exponential weighting factor α.

Lemma 1. Greedy_MWIS *is optimal when $\alpha \leq 0.5$.*

Proof. Suppose the Greedy_MWIS solution S and the optimal solution OPT differ. Let $v \in V$ be the node with the highest weight that belongs to S but not to OPT. Then,

$$\begin{aligned} \sum_{i \in S} w_i - \sum_{i \in OPT} w_i &\geq w_v - \sum_{i \in OPT \setminus S} w_i \\ &\geq w_v - \sum_{\{i \mid w_i < w_v\}} w_i \\ &\geq w_v - w_v \cdot \left(\sum_{1 \leq j < \infty} \alpha^j \right) \\ &\geq w_v - w_v \cdot \frac{\alpha}{1 - \alpha} \\ &\geq 0 \quad \forall \alpha \leq 0.5 \end{aligned}$$

Here, step (2) is true since nodes in OPT with higher weight than w_v must be present in S as well, and step (3) follows from the exponential weighting formula. □

We note here that (a) experiments run without an exponential weighting scheme (e.g., linear combination of precision and recall scores) yielded poor results, as the top-ranked paths are indeed the best in discriminating clusters, (b) results obtained for $\alpha > 0.5$ using a different greedy scheme [19] yielded similar solutions and clustering accuracy, and (c) as $\alpha \rightarrow 1$, the quality of results decreases, since the highest-ranked paths in R_F are no longer picked.

To summarize, we build a graph of the paths obtained from R_F, with edges marking dependence restrictions between paths, and then use GREEDY_MWIS to pick the maximum weight independent set S (for $\alpha < 0.5$). The ranking R_F induces a ranking on S as well, and this ranked list S is then used for shingle-based clustering, as described before in Section 3.1.

3.4 Optimizations for Better Clustering

There are two optimizations that enhance the quality of the clustering.

CURRENTLY-OPEN TAB. Consider a webpage with multiple tabs, e.g., "reviews", "screenshots", "specifications", etc., of which only one is currently open[1]. There is a special path for the currently open tab, call it x. Then, clustering based on whether x is present or not will clump all review, screenshots, and specs pages together since x is present in all of them (only the content of x tells us if this page contains reviews, or screenshots). However, a page with the "screenshots" tab open should ideally not be in the same cluster as a page with similar structure except that the "reviews" tab is open. We detect such tabs by looking for paths that are present on many pages, but whose content is only of a few types (between 2 and 10). Then, during clustering, the set of pages containing such tabs are split further according to the content of the tab.

TEMPLATE REMOVAL. Detecting templated content on a webpage, such as menus, navigation bars, and ads, has been studied extensively [2]. Occasionally, these can match query terms and pollute the set of key paths. We use the site-template detection method used in [2] to remove such instances. This is orthogonal to our work, and a better template detection method can only improve the clustering.

4. EXPERIMENTS

In this section, we present an empirical evaluation of our proposed method (named KEYCLUS). First, in Section 4.1, we discuss the dataset, the ground truth, and the baseline methods that we compare against. Then, in Section 4.2, we compare the clustering achieved by KEYCLUS to the ground truth, and demonstrate that it achieves significantly higher accuracy than existing techniques. In Section 4.3, we show that the key independent paths found by KEYCLUS (call these paths INDEPPATHS) are also close to paths picked by a separate wrapper induction process, showing that the INDEPPATHS are indeed picking out important structure in the webpages. Finally, in Section 4.4, we discuss a few issues and limitations of our work.

4.1 Data and Baselines

A random sample of around 20,000 webpages was collected from each of 10 websites (listed in Table 1) in April, 2009. We conduct all our experiments on these webpages. There is large variance in the number of paths per website, from 4,050 for `www.insiderpages.com` to over 2×10^6 for `www.hotels.com`, with an average of around 480,000 per site. Recent search logs were collected for this website, yielding an average of around 4,800 distinct searches per site. Note that this is much smaller than the number of webpages per site, and not all webpages see search activity.

GROUND TRUTH COLLECTION. These pages were also used to build site-specific wrappers via a separate supervised-learning approach that is unrelated to our work. The induced wrappers use a few paths to extract important content, such as product title, prices, hotel locations, etc., from a subset of pages on the website that are judged to be useful for data extraction. We have access to these paths used in the wrappers (call them GT-PATHS) and also the subset of pages where they apply (call it WRAPPERPAGES). This becomes our first ground-truth set: we will show that our automatically generated INDEPPATHS match these GT-PATHS, and thus can be used not only in webpage clustering but also in wrapper induction.

We also build the ideal ground-truth clustering on these pages, via an iterative process that interleaves human labeling with automatic clustering. We start with an automatic clustering of all webpages — each wrapper w induces a cluster of pages such that w can extract data from those pages with high confidence; pages not falling within any wrapper-induced cluster are then clustered via an in-house clustering method. Then, human editors view 10 random pages and 10 of the most structurally-dissimilar pages in each cluster, and judge each cluster to be homogeneous or not. A cluster is said to be homogeneous iff the presented set of pages have similar look and feel, and represent the same content class (such as product, business, or movies). Homogeneous clusters can also be merged if an editor finds them to be similar based on look and feel, and content class.

The clusters judged to be heterogeneous are split further by changing the thresholds in the automatic clustering algorithm, and the process is repeated until less than 5% of the pages in a website remain unclustered. The clusters are subjected to a separate manual inspection to confirm their homogeneity, with particular attention being paid to the largest clusters, and those intersecting significantly with WRAPPERPAGES. The end result is a ground-truth clustering for all the 10 sites (call it GT-CLUSTERS). Clearly, this is an extremely time-consuming and expensive process, and a fully-automated clustering algorithm like KEYCLUS would be most valuable for these purposes.

BASELINE ALGORITHMS. There are three baseline algorithms that we compare KEYCLUS against.

Path-shingles: Introduced in [5], Path-shingles is equivalent to KEYCLUS with no key paths. In other words, this is performance that is possible without access to search logs.

pq-Grams: Instead of using paths to represent a DOM tree, pq-Grams use the neighborhood of nodes [1]. In particular, the pq-Gram of a node X includes X, the $p-1$ ancestors of X, and some q children of X (if any of these do not exist, null values are used). Just as with path-shingles, a DOM tree can be represented as a shingle of pq-Grams, which can then be used in the clustering step described in Section 3.1. The original paper suggested $p = 2$ and $q = 3$, but we experiment with multiple settings and report the best scores.

[1] See, for example, `http://reviews.cnet.com/pc-games/penumbra-black-plaguepc/4505-9696_7-32622390.html`.

m/k Path shingles: Recall that, for efficiency reasons, we use the basic shingle-based clustering technique: every distinct k-shingle is in a cluster of its own. We could also use a more complicated variant where two shingles can be placed in the same cluster if they share only $m < k$ elements. In our implementation, clusters are represented by the most common m-element subsets of the k-element shingles of all webpages in the website, and any webpage whose shingle contains these m elements is placed in the cluster. This allows the merging of clusters that would otherwise be separated by the basic clustering algorithm. We show results with $m = 6, k = 8$, but results obtained with several other combinations (e.g, 2/4, 3/4) were also similar, with no one setting dominating the others.

4.2 Accuracy of Key Paths and Clusters

We shall compare KeyClus to the baselines in terms of clustering accuracy, measured using the *adjusted RAND index*, described next.

Adjusted RAND index. Given two clusterings C_1 and C_2 of the same set of points, we count the number of pairs of points that the clusterings agree or disagree about. In particular, let a be the number of pairs which are placed in the same cluster in both C_1 and C_2, b the number of pairs placed together in C_1 but not in C_2, c the pairs separated in C_1 but together in C_2, and d the number of pairs separated in both. In essence, the two clusterings agree over $a + d$ pairs, and disagree about $b + c$ pairs. The adjusted rand index [13] is given by:

$$AR = \frac{2(a \cdot d - b \cdot c)}{(b + a) \cdot (b + d) + (c + a) \cdot (c + d)}$$

The value of AR is upper-bounded by 1, and the expected score for a random clustering is 0. Higher scores are better.

Accuracy of Clustering. KeyClus and the three baseline algorithms were run with 50 random seeds, for shingle size varying from $k = \{1 \ldots 8\}$. Then, for each algorithm and each website, the best average score was picked among all shingle-sizes. The average scores and standard deviations are shown in Table 1. Two observations are immediate.

(1) KeyClus outperforms all baseline algorithms over all sites. The RAND score is over 0.9 for 5 sites, and KeyClus delivers a 59% relative improvement on average upon the scores of the next best algorithm (Path Shingles). This clearly demonstrates the importance of search logs. The relatively poor performance of Path Shingles and pq-Grams shows that randomly picked paths or subgraphs from the DOM tree are unlikely to pick out distinguishing characteristics of pages. m/k Path Shingles performs even worse than plain Path Shingles (only the 6/8 case is shown here, but results are similar for all other cases we tried). This is because each cluster is now marked by the most common m-element subset among all the k-element path shingles, and these common subsets tend to contain some very common paths, such as generic high-level paths present on most pages, which are not useful in distinguishing clusters. Template removal did not significantly affect baseline accuracy.

(2) The standard deviation of KeyClus is far smaller than that of the baselines. This is because most of the shingle elements are key paths, which do not change over multiple runs. However, the baselines represent the same webpage with different paths/nodes in different iterations, leading to instability. This is crucial in a real-world system, where one clustering must be picked for subsequent use. With no way to differentiate between clusterings, it is possible that the clustering that is picked is of very poor quality. The low variability of KeyClus is advantageous here.

We note that, contrary to the trend of very high RAND scores, `www.epinions.com` and `shopping.yahoo.com` achieve relatively lower RAND scores, even though KeyClus still outperforms the baselines on them. A closer inspection revealed that this was due to an unavoidable error during the ground truth collection process. There are several sets of clusters whose pages look alike, and hence these clusters were merged into one by the human editors. However, the DOM trees of the pages in these clusters are completely different. For example, the results of a search list are contained in a `<form>` tag in one set of pages but in a `<ul>` tag in the second. KeyClus places these pages in two different clusters, and this is the correct behavior if the clusters are to be subsequently used for wrapper induction. However, it is impossible for human editors to distinguish these pages simply by look-and-feel.

Shingle size. Table 2 shows the best shingle sizes for KeyClus, Path Shingles, and pq-Grams (6/8 Path Shingles always uses a shingle-size of 8). We see that a shingle-size of only 1 or 2 suffices for KeyClus, which has a few important implications.

(1) With a shingle size of 1, most pages have a shingle consisting of just one key path; and no randomly picked paths need to be added to the shingle. The accuracy achieved with such a small shingle shows that our carefully picked key paths are enough to discriminate between pages of different types. The corresponding clusters are also stable across multiple iterations using different random seeds, leading to the low variances observed above. Also, we find that the bulk of the pages on all website are covered by such key-path-only clusters; only 18% of the pages on average are not covered by any key path.

(2) A shingle-size of 1 gives good results for all websites, with shingle-size of 2 performing better only on `www.epinions.com` and `www.target.com`. This allows us to just pick a shingle-size of 1 for any new website, without having to worry about parameter tuning.

(3) A single-element shingle also speeds up execution in a real-world system. Each cluster is defined by just one path, and there are only a few clusters per website, meaning that every time a new page enters the system, it needs to be checked for the existence of only a few paths. In fact, if some of these paths can be uniquely identified by special attributes (e.g., unique `id` or `class` tags), then we can even just search for these attributes in the HTML file without building the DOM tree, leading to significant time savings.

(4) The high accuracy with a shingle size of 1 is an effect of picking independent paths in KeyClus: rarely do two key paths occur in the same cluster. As against this, for the baseline algorithms, the optimal shingle size depends considerably on the website, again contributing to the difficulty of using the baseline algorithms in a real-world situation.

Cluster sizes. Since small clusters are typically not worth the labeling effort in wrapper induction, webpages falling into such small clusters are often unused. We find KeyClus and pq-Grams place less than 1.5% of the pages into such clusters on average, but Path Shingles places nearly 8% of the pages in such clusters, which can lead to significant loss of informative and extractable content from a website.

Website	KeyClus	Path Shingles	pq-Grams	6/8 Path Shingles
reviews.cnet.com	**0.95** (±0.011)	0.85 (±0.12)	0.47 (±0.22)	0.54 (±0.18)
search.barnesandnoble.com	**0.86** (±0.019)	0.51 (±0.16)	0.58 (±0.25)	0.34 (±0.12)
shopping.yahoo.com	**0.65** (±0.005)	0.61 (±0.1)	0.49 (±0.19)	0.32 (±0.14)
www.amazon.com	**0.79** (±0.076)	0.51 (±0.11)	0.55 (±0.15)	0.4 (±0.06)
www.bhphotovideo.com	**0.97** (±0.0033)	0.52 (±0.23)	0.38 (±0.23)	0.18 (±0.1)
www.cduniverse.com	**0.91** (±0.0033)	0.26 (±0.24)	0.36 (±0.27)	0.09 (±0.04)
www.epinions.com	**0.61** (±0.034)	0.53 (±0.089)	0.33 (±0.19)	0.48 (±0.06)
www.hotels.com	**0.74** (±0.0065)	0.64 (±0.15)	0.63 (±0.21)	0.36 (±0.13)
www.insiderpages.com	**0.99** (±0.0049)	0.83 (±0.2)	0.76 (±0.24)	0.72 (±0.14)
www.target.com	**0.94** (±0.056)	0.58 (±0.17)	0.47 (±0.25)	0.54 (±0.15)

Table 1: ***Comparison of Adjusted RAND index for*** **KeyClus** ***and three baseline algorithms.*** **KeyClus clearly outperforms the rest, showing the importance of using search logs.**

Website	Number of indep. paths	Number of tabs
reviews.cnet.com	3	5
search.barnesandnoble.com	2	0
shopping.yahoo.com	3	0
www.amazon.com	13	0
www.bhphotovideo.com	4	10
www.cduniverse.com	5	5
www.epinions.com	19	9
www.hotels.com	7	7
www.insiderpages.com	5	0
www.target.com	3	0

Table 3: ***Number of*** **IndepPaths** ***and tabs.***

In the case of applications requiring human involvement on a per-cluster basis, e.g., wrapper induction, the *number* of clusters matters as well, since the cost of human interaction is proportional to number of clusters. Table 2 also lists the number of clusters with at least 20 webpages obtained by each algorithm. The number of clusters in KeyClus is always close to that for GT-Clusters, without any parameter tuning. The baseline algorithms get similar results only for their best parameter settings, which can vary significantly across websites. For a new website, and without any human supervision, it is unclear which parameters should be picked and how well the algorithms will perform.

Nature of IndepPaths. Finally, we discuss the nature of IndepPaths picked up by KeyClus. Table 3 lists, for each website, the number of independent paths, and tabs found, where each tab is a unique (tab path, tab content) pair. Recall that each tab corresponds to a cluster formed by the combination of a path and its content (see Section 3.4). Even with such a small size of independent paths for each website, we still obtain significant improvement in RAND score compared to baseline algorithms. The small size of the independent path set also means that the memory footprint required for clustering is very small, potentially improving runtime performance.

Table 4 shows the top-most independent path[2] picked by KeyClus for several sites, and what it refers to in the page. We observe that the paths typically represents the key information of the page such as product title, business name, or hotel address. Several of these paths end in `<h1>` or `<h4>`, i.e., they pick out headers. Note that KeyClus prefers paths representing mandatory information such as product title, over paths representing optional information such as reviews or product description. The selection of such paths gives KeyClus an edge over baseline algorithms.

4.3 Matching IndepPaths to GT-Paths

The previous section presented indirect evidence of the goodness of the IndepPaths by measuring clustering accuracy. In this section, we will directly verify their goodness by comparing them to paths that are actually used in wrappers, namely GT-Paths. If IndepPaths match GT-Paths, it means that IndepPaths point to information that is considered important by human editors (since GT-Paths are generated using human input), and that the clusters generated by KeyClus are interpretable.

Detecting a match. An exact match between paths in IndepPaths and GT-Paths is obviously the clearest indicator of a match, but this is rare due to two reasons: (a) wrapper induction typically picks paths/nodes that are high up in the DOM Tree, while our paths are geared more towards precision and hence lower down in the tree, and (b) two paths can be very close and have very similar content and yet not be identical, e.g., one path leading to a node and another to its only child, or paths leading to two very close siblings. Intuitively, if two path differ only in the tail, they typically belong to the same section within the page, and we would like to count this as a match. We handle this by allowing matches on *long common prefixes*, as follows.

A path x_1 in IndepPaths is considered a match to a path x_2 in GT-Paths if the longest common prefix between the two is at least 90% of the length of x_2. Thus, x_1 can "fork off" from x_2 only at some close ancestor of x_2. Note that this is a fairly harsh condition: if x_2 points to a node at depth 9 or less, then x_1 can only match by matching x_2 exactly or by being a descendant of x_2.

Results. Table 5 presents the precision of IndepPaths when compared to GT-Paths; recall is unimportant here because our goal in producing IndepPaths was not to extract all content, but only enough to distinguish between different types of pages. From the table, we can observe that although the precision is high overall, it is poor on a few sites. The reason for this is that wrappers were induced only on certain types of pages judged to be more important, and not on others. For example, it is more important to extract information from a page about one product, than to extract a listing of products from a "category list" page, especially if the product page itself also notes the category of the product. However, for the purposes of clustering, it is important to pick relevant paths from the "category list" pages as well, in order to distinguish these pages from other

[2] This set of webpages was crawled in April'09, so it is possible that page structure might have changed.

Website	KeyClus		Path Shingles		pq-Grams		GT-Clusters
	Shingle size	#Clusters	Shingle size	#Clusters	Shingle size and (p,q)	#Clusters	#Clusters
reviews.cnet.com	1	49	1	54	7 (4,3)	80	37
search.barnesandnoble.com	1	7	2	15	1 (3,1)	7	8
shopping.yahoo.com	1	16	1	20	3 (4,5)	32	21
www.amazon.com	1	45	2	32	3 (4,2)	63	29
www.bhphotovideo.com	1	17	1	32	2 (1,5)	24	10
www.cduniverse.com	1	14	1	13	1 (3,5)	8	9
www.epinions.com	2	38	3	34	3 (1, 1)	18	19
www.hotels.com	1	20	2	17	4 (4, 2)	17	9
www.insiderpages.com	1	10	2	19	2 (3,1)	14	21
www.target.com	2	9	4	23	2 (3,1)	12	14

Table 2: ***Best shingle size and average number of clusters for each algorithm.***

Website	Top-most independent path	What it refers
reviews.cnet.com	html body[@class="siteId7 pageType4505"] div[@id="rb_bodyWrap"] div[@id="rb_shell"] div[@id="rb_content"] div[@id="contentMain"] div [@id="overviewHead"] h1	product title
shopping.yahoo.com	html body[@id="sl-search][@class="rsprite1"] div[@id="outerdoc"] div[@id="doc4"] [@class="search"] div[@class="doc-wrapper1 vsprite1"] div[@class="doc-wrapper4 vsprite1"] div[@id="bd"] div[@id="list"] div[@id="list_l"] div[@id="list_cont"][@class= "list-cont-rlp"] div[@id="list_lhs"] div[@id="list_lhs_w"] div[@class="narrow"] div[@class="narrow_w"] h4	search results header
www.amazon.com	html body[@class="dp"] div[@class="singlecolumnminwidth"] form[@id="handleBuy"] div[@class="buying"]	product title
www.epinions.com	html body div[@class="xkb"] table tbody tr td table tbody tr td table tbody tr td table tbody tr td h1[@class="title"]	product title
www.hotels.com	html body div[@id="sizer"] div[@id="contentBody"][@class="clearFix"] div[@id= "propertyHeaderTripDetails"] [@class="curvedBorder"] div[@class="innerWrap clearFix "] div[@id="propertyHeaderHotelDetails"] [@class="clearFix"] div [@class ="hotelSnippet vcard"] div[@class="hotelAddress"] div[@class="adr"] span[@class="locality"]	hotel address
www.insiderpages.com	html body div[@id="width"] div[@id="min_width"] div[@class="content"] table[@id ="main_info"] tbody tr td table tbody tr td div[@class="business_card"] div[@class="business_info"] h1	business name
www.target.com	html body div div[@id="mainContent"] div[@id="middleSlots"] div[@id="mainBody"] div[@id="middleUpperSlots"] div[@id="rightColumn"] div[@id="right-2"]	product title

Table 4: ***Top-most independent path for a few sites, and what it refers to in the page***

page types. Thus, the comprehensiveness of IndepPaths hurts it in this comparison.

To make a fair comparison, we look at the subset of pages on which GT-Paths can extract information. These are the pages for which the wrappers were induced. We find the set S' of paths in IndepPaths that are present on this subset of pages. Now, S' and GT-Paths are applicable to the same subset of pages, and hence a comparison between them is fair. The results are presented in Table 5, and show the high precision achieved on almost all sites (90% precision on average). Thus, the key independent paths that were generated completely automatically by our algorithm are able to match paths picked by human editors to extract important content from webpages. Not only does this make our paths and clusters more interpretable, but it also helps editors in the wrapper-induction step following the clustering obtained by KeyClus. Given the significant cost and effort required in manual labeling, any such help is very useful.

We also looked deeper into the paths in IndepPaths that did not match GT-Paths even on the wrapper-extractable pages. There are two main reasons behind such mismatches. First, some independent paths are actually "tabs," such as "overview", "reviews," and "screenshots," as discussed before in Section 3.4. These are clearly important to get the clustering correct, but their content is not specific to the content of the webpage. So, they are not useful for data extraction from the page, and are absent from GT-Paths. The second reason is that several websites use multiple *variants* for laying out the same information. Common examples include extra `<div>`, `<table>`, and `<tbody>` tags that are inserted into pages of the same type; these do not affect the visual layout of the webpage but can fragment information about essentially the same path in our analysis. We are currently studying ways to deal with such flexible paths. However, we note here that even without such flexibility, we get very good matches between IndepPaths and GT-Paths over all 10 websites.

4.4 Discussion

While both the accuracy of KeyClus and the precision of IndepPaths were verified in the previous sections, there are a few issues that merit discussion. The first is that of different paths being used to point to the same type of content on different pages, e.g., `<form>` and `<ul>` tags being used to list items in `shopping.yahoo.com`. This becomes more problematic if one of the paths occurs relatively infrequently, or occurs on pages that receive fewer queries, making it likelier that this path will be ranked too low in R_F to be selected into IndepPaths. One approach would be to compute the coordinates at which DOM nodes are rendered on screen, and match paths if their coordinates are similar.

A second problem is the proper detection of the "tabs," discussed before in Section 3.4. Quite often, the content of the tab does not match queries, and so the tab is not

Website	All IndepPaths	IndepPaths present on WrapperPages
reviews.cnet.com	0.33	0.50
search.barnesandnoble.com	1	1
shopping.yahoo.com	1	1
www.amazon.com	0.84	0.80
www.bhphotovideo.com	0.75	1
www.cduniverse.com	0.60	1
www.epinions.com	0.67	0.67
www.hotels.com	0.17	1
www.insiderpages.com	0.25	1
www.target.com	0.67	1

Table 5: *Precision against* GT-Paths*:* The precision with respect to GT-Paths is shown for (a) IndepPaths, and (b) the subset of IndepPaths that occur on wrapper-extractable pages.

picked up as an important path in the website (e.g., tabs for showing products in “list view” versus “grid view”). However, not all paths whose content comes from a small set are tabs, e.g, “price range $0 – $99” versus “price range $100 – $199,” so matching the query workload is still important. Correct detection of tabs would improve webpage clustering, but we observe that detection of tabs is not as crucial for wrapper induction, since the most informative content on a page is often present irrespective of the currently open tab.

Another issue is that of parameter selection. The shingle size, the dependence threshold θ_{OCC}, and the number of top ranked paths in R_F that are considered in the independent-path selection are all parameters that affect the trade-off between number of clusters and heterogeneity of clusters. We have attempted to provide a setting that gives good results on the adjusted RAND score, which gives equal weight to incorrect cluster merges and splits. However, the parameter setting should be modified based on the application.

5. CONCLUSIONS

We presented a completely automated, unsupervised approach to cluster structurally similar webpages, with the primary goal of using the clusters in a subsequent wrapper induction step, but not restricted to it. Our algorithm leverages search logs to identify paths in DOM trees such that their content often matches the query terms, both in terms of precision and recall. However, some of these paths might be related to each other, either in their patterns of occurrence on webpages or in their content. We formalize this as a maximum weighted independent set problem, and propose an algorithm that is optimal under certain settings of weights. This yields a set of independent paths, which are then used in a shingle-based clustering of the webpages on a website. We demonstrate empirically on a set of 10 websites that the clusters found by our algorithm are in very close agreement with the ground truth clustering, achieving a RAND score of over 0.9 on 5 sites, and performing 59% better on average than competing baselines. In addition, we show that the paths themselves are very similar to the ones found by human labelers during wrapper induction (90% average precision), leading to increased interpretability for both the set of independent paths picked by our algorithm, and the associated clustering.

6. ACKNOWLEDGMENTS

We thank Ravi Kumar, Srinivasan Sengamedu, Aravindan Raghuveer, Charu Tiwari, Kunal Punera, and Sandeep Pandey for their help with formalizing the problem and with the experiments.

7. REFERENCES

[1] N. Augsten, M. Böhlen, and J. Gamper. Approximate matching of hierarchical data using pq-grams. In *VLDB*, 2005.

[2] Z. Bar-Yossed and S. Rajagopalan. Template detection via data mining and its applications. In *WWW*, 2002.

[3] J. C. Borda. Memoire sur les elections au scrutin. *Histoire de l'Academie Royale des Sciences*, 1781.

[4] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. 1997.

[5] D. Buttler. A short survey of document structure similarity algorithms. In *Internet Computing*, 2004.

[6] C.-H. Chang and S.-C. Lui. IEPAD: Information extraction based on pattern discovery. In *WWW*, 2001.

[7] W. Cohen and W. Fan. Learning page-independent heuristics for extracting data from web pages. In *WWW*, 2002.

[8] V. Crescenzi, G. Mecca, P. Merialdo, U. Roma, T. UniversitĹ, B. UniversitĹ, and R. Tre. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, pages 109–118, 2001.

[9] V. Crescenzi, P. Merialdo, and P. Missier. Clustering web pages based on their structure. *Data Knowl. Eng.*, 54(3):279–299, 2005.

[10] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *WWW*, 2001.

[11] S. F. Giuseppe, E. Masciari, L. Pontieri, and A. Pugliese. Detecting structural similarities between xml documents. In *In Proc. of the 5th Intl. Workshop on the Web and Databases*, pages 55–60, 2002.

[12] A. Hogue and D. Karger. Thresher: Automating the unwrapping of semantic content from the World Wide Web. In *WWW*, 2005.

[13] L. Hubert and P. Arabie. Comparing partitions. *J. Classification*, 2:193–218, 1985.

[14] U. Irmak and T. Suel. Interactive wrapper generation with minimal user effort. In *WWW*, 2006.

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

[16] S. Lu. A tree-to-tree distance and its application to cluster analysis. *PAMI*, 1(2), 1979.

[17] I. Muslea, S. Minton, and C. Knoblock. STALKER: Learning extraction rules for semistructured, web-based information sources. In *AAAI Workshop on AI and Information Integration*, 1998.

[18] D. C. Reis, P. B. Golgher, A. S. Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *WWW*, 2004.

[19] S. Sakai, M. Togasaki, and K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Math.*, 126(2-3):313–322, 2003.

[20] D. Shasha and K. Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, 1990.

[21] J. T.-L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching, 1992.

[22] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *KDD*, 2007.