Probe, Cluster, and Discover: Focused Extraction of QA-Pagelets from the Deep Web

James Caverlee, Ling Liu, and David Buttler
College of Computing, Georgia Institute of Technology
{caverlee, lingliu, buttler}@cc.gatech.edu

Abstract

In this paper, we introduce the concept of a QA-Pagelet to refer to the content region in a dynamic page that contains query matches. We present THOR, a scalable and efficient mining system for discovering and extracting QA-Pagelets from the Deep Web. A unique feature of THOR is its two-phase extraction framework. In the first phase, pages from a deep web site are grouped into distinct clusters of structurally-similar pages. In the second phase, pages from each page cluster are examined through a subtree filtering algorithm that exploits the structural and content similarity at subtree level to identify the QA-Pagelets.

1. Introduction

The Deep Web (*or Hidden Web*) comprises all information that resides in autonomous databases behind portals and information providers' web front-ends. Web pages in the Deep Web are dynamically-generated in response to a query through a web site's search form and often contain rich content. A recent study has estimated the size of this "deep web" to be more than 500 billion pages, whereas the size of the "crawlable" web is only 1% of the Deep Web (i.e., less than 5 billion pages) [4]. Even those web sites with some static links that are "crawlable" by a search engine often have much more information available only through a query interface. Unlocking this vast deep web content presents a major research challenge.

In analogy to search engines over the "crawlable" web, which fully automate the crawling of static web pages, we argue that one way to achieve robustness and scalability required for Internet-scale information integration service is to employ a fully automated approach to extracting, indexing, and searching the query-related information-rich regions from dynamic web pages. We envision that a search engine over the Deep Web will require three unique features that are not present in traditional search engines: (1) an efficient means of discovering and categorizing deep web data sources[1], (2) an effective method for indexing dynamic web pages in terms of content category and the data returned by a query over the search interface, and (3) a retrieval engine that supports searching by sites (e.g., list all bioinformatic web sites supporting BLAST queries) and supports searching by fine-grained content (e.g., list seller and price information of all digital cameras from Sony).

A basic building block for developing such a deep web search engine is a fully automated approach to extracting the query-related information-rich content from dynamic web pages, including scalable and robust methods for analyzing dynamic web pages of a given web site, discovering and locating the query-related information-rich content regions, and extracting itemized objects within each region. Fully automated query-related content extraction is possible because dynamic web pages typically consist of a handful of presentation region types. Three common examples include:

1. *The query-answer regions*, which present the primary content directly related to a query posed to the search interface of the content provider. Some web sites support multiple primary content regions.
2. *The advertisement region*, which presents the information about other products offered by the content provider or about related products offered by other companies.
3. *The navigational region*, which presents a collection of navigational links, often to other web sites provided by the same content provider (such as Amazon's music site, Amazon's DVD site, and so on).

With these observations in mind, we introduce the concept of a *Query-Answer Pagelet* (or *QA-Pagelet* for short) to refer to the query-related content region in a dynamic page that contains query matches. With the QA-Pagelet as our fundamental basis, we present THOR – a fully automated approach for discovering and extracting the query-related content from dynamically-generated pages. A unique character-

[1] Other researchers [16] have proposed methods to categorize deep web sites into searchable hierarchies.

istic of THOR is its novel two-phase algorithm for extracting the QA-Pagelets from dynamic pages. In the first phase, pages are clustered according to their structural similarities, aiming at separating pages that contain query matches from pages that contain no matches (like exception pages, error pages, and so on). A ranked list of page clusters is produced by applying a tag-tree signature based TFIDF similarity function with a K-Means clustering algorithm. In the second phase, pages from top-ranked page clusters are examined at a subtree level to identify the smallest subtrees that contain the QA-Pagelets.

We evaluated the first implementation of the THOR system using more than 5000 pages over 50 diverse deep web sources and a synthetic data set of over 5 million pages. Our experiments show three important and interesting results. First, our page cluster algorithm achieves high quality of clusters with very low entropy. Second, our algorithms for identifying the query-related content regions by filtering subtrees across pages within a page cluster achieve excellent recall (96%, meaning significant query-related content regions are left out only in rare cases), and precision (97%, meaning nearly all regions identified and extracted are correct) over all the pages we have tested. Most significantly, THOR's algorithms are robust against changes in presentation and content of deep web pages, and scale well with the growing number of deep web sources.

2. Focused QA-Pagelet Extraction: Overview

We model web pages as tag trees using a variation of the well-known Document Object Model [30]. Each tag tree $\mathcal{T}$ consists of *tag nodes* and *content nodes*. A tag node consists of all the characters from a particular start tag to its corresponding end tag, and is labeled by the name of the start tag. A content node consists of all the characters between a start tag and its corresponding end tag or between an end tag and the next start tag. We label a content node by its content. All content nodes are leaves of the tag tree.

In Figure 1 we show an abridged sample tag tree for a dynamically-generated page at IBM.com. Given a tag tree, there is a path from the root node to every other node in the tree. The path expression from the root of the tree to a node identifies the subtree rooted at that node. Using an XPath-style notation, we may refer to the subtree rooted at the gray-shaded table node in Figure 1 as `html/body/table[3]`.

In the context of the Deep Web, the content-rich regions in a dynamically-generated web page are the ones that contain direct answers to a user's query. We refer to these regions as the *QA-Pagelets*. The term *pagelet* was first introduced in [2] to describe a region of a web page that is distinct in terms of its subject matter or its content. In THOR, we use the term QA-Pagelet to refer to a subtree in the tag tree of a page that: (1) is dynamically-generated in response to a query and (2) is a page fragment that serves as the primary query-answer content on the page. The first condition excludes all static portions of a page that are common across many deep web pages, such as navigation bars, standard explanatory text, boilerplate, etc. But not all dynamically-generated content regions in a page are meant to be direct answers to a query. One example is a personalized advertisement that may be targeted to a particular query. The second condition is necessary to exclude from consideration those regions – like advertisements – that are dynamically-generated but are of secondary importance. In Figure 1, the subtree corresponding to the QA-Pagelet is shown in the dashed box. The root of the QA-Pagelet is the black-shaded table node.

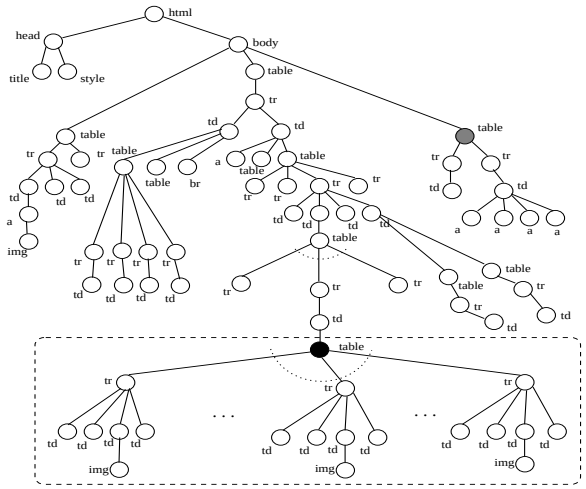

Figure 1: Sample Tag Tree from IBM.com

In THOR, the problem of focused extraction of QA-Pagelets from deep web pages is addressed in three stages, illustrated in Figure 2. The first stage collects the sample answer pages generated in response to queries over a deep web site. The second stage identifies and locates the QA-Pagelets in dynamically-generated web pages. The third stage partitions a QA-Pagelet into itemized QA-Objects, which are in turn fed into the deep web search or information integration system.

Stage 1: Sample Page Collection by Query Probing
In this preprocessing stage for focused extraction of QA-Pagelets, we collect sample answer pages by probing a deep web site with a set of carefully designed queries to generate a representative set of sample answer pages. Researchers have previously studied the problem of repeatedly querying an unknown database in an effort to generate a summary of the database internals [17, 7, 15, 22]. Our problem is slightly different. In addition to generating pages that are representative of the content of the underlying database, we need to sample the answer pages from a deep web site to allow us to generate a diverse set of pages, which capture all possible classes of structurally different answer pages. In the first

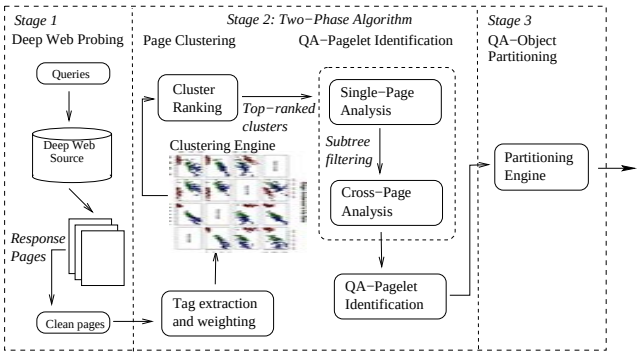

Figure 2: THOR System Architecture

prototype of THOR, we implement a technique that uses random words from a dictionary *and* a set of nonsense words unlikely to be indexed in any deep web database. Our sampling approach repeatedly queries a deep web site with single word queries taken from our two sets of candidate terms. At a minimum, this approach makes it possible to generate at least two classes of pages – normal answer pages and "no matches" pages. Our technique improves on the naive technique of simply using dictionary words, and is effective in collecting samples of diverse classes of answer pages.

Stage 2: Two-Phase QA-Pagelet Extraction
The heart of the THOR system is the Two-Phase QA-Pagelet Extraction stage, which consists of the *Page Clustering Phase* and the *QA-Pagelet Identification Phase*. This stage takes the collection of n sample pages generated in response to probing queries over a single deep web site. In the first phase, it generates page clusters by grouping structurally-similar pages together to separate pages with multiple matches in response to a query from pages with no matches at all. As a result, a ranked list of page clusters is generated. In the second phase, THOR examines pages from top-ranked page clusters at a subtree level to identify the smallest subtrees that contain the QA-Pagelets. The main techniques used include a subtree filtering algorithm that performs single-page analysis and cross-page analysis to group subtrees of similar tree shapes together and to prune those subtrees that are unlikely to contain QA-Pagelets. The output of the second stage is a set of QA-Pagelets from the probed pages. We discuss this two-phase approach in great detail in Section 3.

Stage 3: QA-Object Partitioning
In THOR's third and final stage, each QA-Pagelet is analyzed by a partitioning algorithm to discover the object boundaries and separate out any component objects within the QA-Pagelet. We call these sub-objects *QA-Objects*. A QA-Object is contained in a QA-Pagelet and is a close coupling of related information about a particular item or concept. In Figure 1, the QA-Pagelet contains ten QA-Objects corresponding to ten different query matches (note: only three are shown for illustration). To isolate the QA-Objects within a QA-Pagelet, the third stage first takes into consideration a list of recommended QA-Objects from THOR's second stage. One of the attractive features of the second stage is that the same techniques for extracting the QA-Pagelets can also be used to identify QA-Objects. Given a list of QA-Object candidates, the third stage examines each candidate's particular structure and then searches the rest of the QA-Pagelet for similar structures. Some of the parameters considered include the size, layout, and depth of the other potential QA-Objects. We then deduce the correct object separators and partition the QA-Objects.

Due to the space restriction, in the subsequent sections we focus on the two-phase extraction algorithms for identifying and locating QA-Pagelets in dynamically-generated web pages. Readers may refer to our technical report [8] for a more detailed discussion on the QA-Object partitioning techniques.

3. Two-Phase QA-Pagelet Extraction

A unique feature of the THOR system is its two-phase extraction framework for QA-Pagelet extraction. We identify two levels of relevance corresponding to structure and to content that lay the foundation for our system:

1. Structural relevance: Web pages dynamically-generated by a particular search form for a particular site tend to display certain traits not found in a random collection of pages from the Web. Many web sites tend to structure dynamically-generated pages in a similar fashion. There are clearly a handful of templates used to represent different types of answers to a search query over the deep web database source: be it an answer page with a list of matches, a single match page, or a "no matches" page.

2. Content relevance: For a particular class of dynamically-generated pages from a particular site – say, a set of n normal answer pages from Amazon, each generated in response to a different query – cross-page content information may yield clues as to which fragments of a page contain QA-Pagelets. Some portions of the page contain information that is similar across all pages in the cluster, while other portions are dynamically-generated in response to a particular user query.

These two levels of relevance naturally suggest a two-phase approach. The first phase partitions a set of n sample answer pages into k clusters, each corresponding to one type of answer page: be it multi-match pages, single-match pages, no-match pages, or exception pages. We refer to this phase as the *Page Clustering Phase*. The second phase partitions the set of subtrees in a single page cluster into a ranked list of common subtree sets, each corresponding to one type of content region in the set of structurally similar answer pages. We then use intra-cluster content metrics to filter out the common content and hone in on the QA-Pagelets. We

(a) Multi-matches (b) Single match (c) No matches

Figure 3: Three Page Types from AllMusic.com

call this second phase the *QA-Pagelet Identification Phase*. At the conclusion of the second phase, THOR recommends a ranked list of QA-Pagelets. This two-phase approach forms the core of the THOR system.

3.1. The Page Clustering Phase

THOR's QA-Pagelet extraction begins with a set of pages from a particular web site sampled at the query probing stage (recall Section 2). The goal of the page clustering is to find structurally-similar groupings such that pages that share similar content structure or are generated from the same page templates will be clustered together.[2] Figure 3 shows three page types for AllMusic.com: a multi-matches page consisting of a list of query matches; a single match page with detailed information on a particular artist (Elvis Presley, in this case); and a no matches page. Page clustering enhances the quality of focused extraction of QA-Pagelets by distinguishing between these page types: so multi-match pages may be analyzed separately from single match pages, and both of these rich response types may be analyzed separately from the no matches type.

3.1.1 Design Choices

In general the problem of page clustering can be defined as follows: Given a set of sample pages from a particular web site, which approach is most effective for clustering the pages into structurally-similar groups, and distinguishing the pages that contain QA-Pagelets from those answer pages that report no matches or exceptions? Formally, given a set of n pages, denoted by $\mathcal{P} = \{p_1, p_2, ..., p_n\}$, a page clustering algorithm can segment these n pages into a clustering $\mathcal{C}$ of k clusters: $\mathcal{C} = \{Cluster_1, \ldots, Cluster_i, \ldots, Cluster_k \mid \bigcup_{i=1}^{k} Cluster_i = \{p_1, ..., p_n\}$ and $Cluster_i \cap Cluster_j = \emptyset\}$

A page clustering approach in general consists of two basic building blocks: the similarity metric and the algorithm that performs partitioning over the set of n pages to produce k clusters ($1 \leq k \leq n$). The similarity metric involves both the conceptual definition of similarity or dissimilarity metric, and the formula that calculates and implements such similarity measure. It plays a critical role in implementing the specific objectives of clustering. The concrete clustering algorithm utilizes the concrete similarity metric to divide the set of n pages into k clusters such that pages in one cluster are more similar to one another and pages in different clusters are more dissimilar with each other. Given a set of pages, there are multiple ways to partition the set into k clusters. Most clustering approaches differ from one another in terms of the similarity metric they use and how well such a similarity metric can reflect the objectives of clustering.

Several popular page grouping approaches are possible, including URL-based [3], link-based [14, 5, 11, 19, 20], content-based [6, 31], and size-based. These techniques find groupings based on non-structural similarity characteristics of the pages, and hence are inappropriate for our task. For example, clustering based on URL-similarity – though effective at partitioning pages from *different* deep web databases by their point of origin – is not suitable here. A query of **Superman** and a query of a nonsense word like **xfghae** on eBay yield URLs of the form, "http://search.ebay.com/search/search.dll?query=superman", and "http://search.ebay.com/search/search.dll?query=xfghae". Even though the URLs are very similar, the two pages belong to two distinct classes – the normal listing of results for **Superman** versus a "no matches" page for the nonsense word. Similarly, the other popular approaches group pages along other non-structural features. For a more detailed discussion of these techniques, we refer readers to our technical report [8].

3.1.2 Our Approach

In contrast, THOR relies on a tag-tree based clustering approach to group pages with similar tag-tree representations into clusters. This approach is simple and yet very effective in its ability to efficiently differentiate dissimilar pages for focused extraction of QA-Pagelets from the Deep Web. First, the tag-tree representation naturally encodes much of the structural information useful for clustering different groups of answer pages provided by a deep web site in response to a query. Second, pages generated using different templates tend to have very different tag-tree structures.

Tag-Tree based Similarity Metric

In THOR a tag-tree based approach is used to define the structural similarity between pages generated in response to different queries. Such a similarity metric can be defined in two steps. First, we need to represent the pages in some specific format that is required for calculating distance between pages. One approach is to define a vector space model that represents each dynamically-generated page as a vector of tags and weights [26, 27]. Concretely, given a set of n

[2] See [18] or [13] for a general introduction to clustering.

pages and a total number of N distinct tags, a page can be described by:

$$p_i = \{\ (tag_1, w_{i1}), (tag_2, w_{i2}), \cdots, (tag_N, w_{iN})\ \}$$

We call such a tag-tree based vector representation of a page the tag-tree signature approach.

The design of the weight system is critical to the quality of the tag-tree signature based similarity metric. There are several ways to define the weights. A simple approach is to assign the weight to be the frequency of the tag's occurrence within the page. However, simply using the raw tag-frequency as the weight in page vectors may result in poor quality of clusters when there are pages that have very similar tag signatures but belong to different classes. For example, a "no results" page and a "single result" page may share the exact same tag signature except for a single <b> tag surrounding the single query result. To increase the distinguishing weight of this single critical tag, THOR weighs all tag-tree signatures using term-frequency inverse-document-frequency (TFIDF), a technique that weights all term vectors based on the characteristics of all the pages across the entire site-specific page space. Concretely, we use a variation of TFIDF, which defines the weight for tag *k* in page *i* as follows:

$$w_{ik} = \log\left(tf_{ik} + 1\right) \cdot \log\left(\frac{n+1}{n_k}\right)$$

where tf_{ik} denotes the frequency of tag k in page i; n is the total number of pages; and n_k denotes the number of pages that contain tag k. We then normalize each vector. TFIDF weights tags highly if they frequently occur in some pages, but infrequently occur across all pages (like the <b> example above). Since the set of possible tags is limited, we expect many tags to occur across all pages. Our version of TFIDF is intuitively appealing since it ensures that even a tag that may occur in all pages – say, <table> – will still have a non-zero impact on the tag signature if it occurs in varying degrees in different pages.

Once all the pages are represented as tag-tree signature vectors, we can compute the similarity between pages using a number of well-known similarity (or distance) metrics, including the simple vector product, the cosine similarity, or the Minkowski distance. In this paper, we choose the cosine similarity because of its appealing characteristics and widespread use in the information retrieval community.

Given a set of n pages, let N be the total number of distinct tags in the set of n pages, let w_{ik} denote the weight for tag k in page i. The cosine similarity between pages i and j is:

$$sim_{Cos}(p_i, p_j) = \left(\frac{\sum_{k=1}^{N} w_{ik} w_{jk}}{\sqrt{\sum_{k=1}^{N} w_{ik}^2} \cdot \sqrt{\sum_{k=1}^{N} w_{jk}^2}}\right)$$

Orthogonal page vectors in our normalized space will have a cosine similarity of 0.0, whereas identical page vectors will have a cosine similarity of 1.0. In the first prototype of THOR, we define our tag-tree based similarity metric by combining the tag-tree signature vector model of the pages, the TFIDF weight system, and a cosine similarity. Such a similarity metric ensures that tags like <html> and <body> that occur equally across many pages will not perversely force two otherwise dissimilar vectors to be considered similar.

Page Clustering Using Tag-tree Signatures

Given the tag-tree signatures of pages and the similarity (or distance) function, a number of clustering algorithms can be applied to partition the set of n pages into k clusters ($1 \leq k \leq n$). In the first prototype of THOR, we choose Simple K-Means since it is conceptually simple and computationally efficient. The algorithm starts by generating k random cluster centers. Each page is assigned to the cluster with the most similar (or least distant) center. The similarity is computed based on the closeness of the tag-tree signature of the page and each of the cluster centers. Then the algorithm refines the k cluster centers based on the centroid of each cluster. Pages are then reassigned to the cluster with the most similar center. The cycle of calculating centroids and assigning pages to clusters repeats until the cluster centroids stabilize. Let C denote both a cluster and the set of pages in the cluster. We define the centroid of cluster C as:

$$centroid_C = \left\{ \begin{array}{c} (tag_1, \frac{1}{|C|} \sum_{i \in C} w_{i1}) \\ (tag_2, \frac{1}{|C|} \sum_{i \in C} w_{i2}) \\ \cdots \\ (tag_N, \frac{1}{|C|} \sum_{i \in C} w_{iN}) \end{array} \right\}$$

where w_{ij} is the TFIDF weight of tag j in page i, and the formula $\frac{1}{|C|} \sum_{i \in C} w_{ij}$ denotes the average weight of the tag j in all pages of the cluster C.

To evaluate the effectiveness of our tag signature based approach in terms of time complexity and clustering quality, we compare our approach with several alternatives in our experiments section. To understand the time complexity of the tag-tree signature based page clustering algorithm, we compare it with a more sophisticated algorithm based on tree-edit distance, where the similarity between two tag trees is computed using a tree-edit distance measure [23]. Although this technique is quite powerful at discerning subtle differences between tag trees, the tree-edit distance is a few orders-of-magnitude slower than the simple tag signature approach used in the first prototype of THOR. To assess the effectiveness of our approach with respect to clustering quality, we compare our approach against a URL-based, a page size-based, and a content-based clustering approach. For the content-based approach we consider the raw content signature and the TFIDF-weighted content signature of each page. The content signature uses content terms in place of tags. Porter's stemming algorithm [24] is applied to gener-

ate content terms.

3.1.3 Ranking the Page Clusters

Once a set of pages has been clustered, we expect that some of the clusters will contain QA-Pagelets, while others (like error and exception pages) will not. Rather than pass along all possible clusters to the QA-Pagelet Identification phase, we rank the clusters according to their likelihood to contain QA-Pagelets and forward only the top-ranked clusters. Several ranking criteria could be used based on the tag-tree characteristics defined in Section 2. We briefly discuss three criteria that are considered in the first prototype of THOR for ranking page clusters:

Average Distinct Terms: Since content-rich pages are generated in response to different probe queries that are carefully designed to sample a given deep web database, we expect to find a higher number of unique words on content-rich pages than on non-content rich pages. The average number of distinct terms for a $Cluster_i$ is the average of distinct term counts of each page in the cluster, namely $\frac{1}{|Cluster_i|}\sum_{p \in Cluster_i} distinctTermsCount(p)$.

Average Fanout: Clusters that have pages with higher average fanout are more likely to contain QA-Pagelets. The average fanout for a $Cluster_i$ can be computed by the average of the largest fanout of a node in each page of the cluster. Namely, $\frac{1}{|Cluster_i|}\sum_{p \in Cluster_i} \max_{u \in p.V}\{fanout(u)\}$, where p.V denotes the set of nodes in page p.

Average Page Size: Larger pages may tend to be more likely to contain QA-Pagelets. We define the average page size for a $Cluster_i$ as $\frac{1}{|Cluster_i|}\sum_{p \in Cluster_c} Size(p)$, where $Size(p)$ denotes the size of page p in bytes.

Each ranking criterion works well for some pages and poor for some other pages. Our initial experiments show that a simple linear combination of the three ranking criteria works quite well.

3.1.4 Evaluating the Page Clusters

In THOR, we use internal similarity and entropy to measure the quality of clusters and to evaluate our clustering approach.

Internal Similarity

Given a set of n pages, the quality of clustering these n pages into k clusters can be measured by the internal similarity of the clustering, which is defined in terms of the internal similarities of each of the k clusters. We measure the internal similarity of a single cluster by a summation of the similarities of each page j to its cluster centroid ($1 \leq j \leq n$):

$$Similarity(Cluster_i) = \sum_{p_j \in Cluster_i} sim_{Cos}(p_j, centroid_i)$$

It is shown [29, 32] that measuring the similarity between each page and its cluster centroid is equivalent to merely finding the length of the cluster centroid. This calculation is appealing since it is computationally inexpensive. The similarity of the entire clustering $\mathcal{C}$ can be computed by a weighted sum of the similarities of all component clusters:

$$Similarity(\mathcal{C}) = \sum_{i=1}^{k} \frac{n_i}{n} Similarity(Cluster_i)$$

We say a clustering has higher quality if its internal similarity is higher. In addition to evaluating the cluster quality, the internal similarity can also be used to as an internal clustering guidance metric to produce the best clustering since it is simple to calculate and requires no outside knowledge of the actual class assignments.

Recall our K-Means clustering algorithm discussed in Section 3.1.2. The quality of this K-Means algorithm can be affected by how the k initial cluster centers are selected. One way to address this problem is to run the K-Means algorithm repeatedly for M iterations. On each iteration we start with k randomly selected cluster centers, and calculate the internal similarity of the clustering produced by this iteration. Finally, we choose the best clustering for the given set of n pages on the iteration that yields the clusters with highest internal similarities.

Entropy

Entropy is a well-known measure of the randomness or disorder in a system [28]. For a particular clustering, entropy measures the quality of assignments of pages to clusters [12]. In the best case, a clustering of a collection of n pages belonging to c classes into k clusters would result in $k = c$ clusters, where each cluster contains only pages from a particular class. In the worst case, the pages from each class would be equally divided among the k clusters, resulting in valueless clusters.

Let $p(z)$ = Prob(page p belongs to class j | page p is in cluster i). For a single cluster, we measure entropy as:

$$Entropy(Cluster_i) = \frac{-1}{\log(c)} \sum_{j=1}^{c} (p(z) \log p(z))$$

where we may approximate $p(z)$ by $\frac{n_i^j}{n_i}$ where n_i = the number of pages in $Cluster_i$; and n_i^j = the number of pages in $Cluster_i$ that belong to $Class_j$.

For an entire clustering $\mathcal{C}$ of n pages, the total entropy is the weighted sum of the component cluster entropies:

$$Entropy(\mathcal{C}) = \sum_{i=1}^{k} \frac{n_i}{n} Entropy(Cluster_i)$$

Our goal is to choose the clustering that minimizes total entropy. Since entropy requires external knowledge about the correct assignment of documents to c classes, we may use entropy only for evaluation of THOR, not as an internal clustering guidance metric, as we do with similarity.

3.2. The QA-Pagelet Identification Phase

On completion of the Page Clustering Phase, only those top-ranked m page clusters that are likely to contain QA-Pagelets are passed to the second phase. The QA-Pagelet Identification Phase takes as input a single cluster of n_c pages, and outputs a list of QA-Pagelets. The main challenge of the second phase is how to effectively discover and locate QA-Pagelets from each of the m page clusters. Though the pages under consideration may contain QA-Pagelets, the QA-Pagelets are often buried by a variety of irrelevant information. As discussed in Section 2, each QA-Pagelet in a page is a subtree of the corresponding tag tree of the page. In THOR we perform the QA-Pagelet identification task in two steps: (1) filtering and ranking candidate subtrees, and (2) selecting the subtrees containing QA-Pagelets. The QA-Pagelets, once identified, are passed to the THOR Object Extraction module, which partitions each QA-Pagelet into a set of QA-Objects. We measure the success of the QA-Pagelet identification phase with precision and recall statistics.

3.2.1 Filtering and Ranking Candidate Subtrees

The goal of filtering candidate subtrees is to discover subtrees from each page that contain dynamically-generated content. The main idea is to identify those subtrees that are likely to contain QA-Pagelets and remove those that are unlikely. Given a web page i and its tag tree $\mathcal{T}$, let $subtrees(i)$ denote the set of subtrees of the page i. Obviously not all the subtrees are likely to contain QA-Pagelets. Thus the task of filtering candidate subtrees is carried out by combining *single-page analysis* with *cross-page analysis*. The former prunes away subtrees that cannot possibly correspond to QA-Pagelets. The latter uses cross-page information to identify and rank those subtrees that have resemblance in multiple pages with respect to tree shape and structural characteristics.

Single-Page Analysis.
The task of single-page analysis takes one of the top-ranked page clusters resulting from the page clustering phase and outputs a set of candidate subtrees for each page in the given cluster of n_c pages. The goal of single-page analysis is to eliminate those subtrees that do not contribute to the identification of QA-Pagelets. It starts out with all the subtrees in the page and proceeds as follows: First, it removes all subtrees that contain no content, then it removes those subtrees that contain equivalent content but are not minimal. Furthermore if the subtree anchored at u is a candidate subtree, then for any descendant w of u, the fanout(w) is greater than one.

Cross-Page Analysis
The goal of cross-page analysis is to group the candidate subtrees that correspond to the same type of content region into one subtree cluster and produce a ranked list of common subtree sets. It takes the n_c sets of candidate subtrees, one set for each page in the given page cluster, and produces a ranked list of k common subtree sets. Each set contains at most one subtree per page and represents one type of content region in all pages of the given page cluster. Since QA-Pagelets in a page are generated in response to a particular query, the subtrees corresponding to the QA-Pagelets should contain content that varies from page to page. In contrast, the common subtree set that corresponds to the navigational bar in each of the n_c pages contains the same or very similar content across all pages. Based on these observations, we leverage the cross-page subtree content to eliminate or give low ranking scores to the subtrees with fairly static content. Cross-page analysis is carried out in two steps: finding common subtree sets by grouping subtrees with a common purpose and ranking the candidate subtrees according to their likelihood of being QA-Pagelets.

Step 1: Finding Common Subtree Sets
For the k page clusters generated in the Page Clustering Phase, only the top-m page clusters are passed to the second phase for QA-Pagelet identification. Consider the set of n_c pages in one of the m page clusters, a common subtree may be a subtree corresponding to the navigation bar, the advertisement region, or the QA-Pagelet.

The algorithm for finding common subtree sets starts by randomly choosing a page, say p_r, from the set of n_c pages in a given page cluster. We call p_r the prototype page. Let $CandidateSubtrees(p_r)$ be the set of candidate subtrees resulting from the single-page analysis. For any subtree anchored at node u of the tag tree of p_r, we find a subtree that is most similar to u in terms of subtree shape and subtree structure. In THOR we introduce four metrics that are content-neutral but structure-sensitive to approximate the *shape* of each subtree. The goal is to identify subtrees from across the set of pages that share many shape characteristics. The metrics we consider are: (1) the path (P_j) to the root of the subtree, (2) the fanout (F_j) of the subtree's root, (3) the depth (D_j) of the subtree's root, and (4) the total number of nodes (N_j) in the subtree. Each subtree, say $subtree_j$, is modeled by a quadruple : $< P_j, F_j, D_j, N_j >$.

For a collection of n_c pages, we will identify k sets of common subtrees ($1 \leq k < |CandidateSubtrees(p_r)|$). Each common subtree set is composed of a single subtree from each page, and a subtree labeled j from page l is denoted as $subtree_j^l$):

$$CommonSubtreeSet_i = \{subtree_{i,1}^1, ...subtree_{i,n_c}^{n_c}\}$$

The criterion for assigning a subtree to one of the k common subtree sets is defined based on the four metrics using the following distance function:

$$distance(subtree_i, subtree_j) = w_1 \frac{EditDist(P_i, P_j)}{max(len(P_i), len(P_j))}$$
$$+w_2 \frac{|F_i - F_j|}{max(F_i, F_j)} + w_3 \frac{|D_i - D_j|}{max(D_i, D_j)} + w_4 \frac{|N_i - N_j|}{max(N_i, N_j)}$$

where $\sum_{i=1}^{4} w_i = 1$.

This distance function measures the distance between subtree i and subtree j. It is designed to minimize the distance between subtrees that share a similar shape (and hence, a similar function in the pages). Any two subtrees within one common subtree set are more similar to one another according to the four distance metrics. Similarly any two subtrees coming from two different common subtrees are less similar in terms of subtree shape and structural characteristics.

The first term measures the string edit-distance between the paths of the two subtrees. String edit-distance [21] captures the number of "edits" needed to transform one string into another. The edit-distance between the strings "cat" and "cake" would be two; there are two edits necessary, changing the "t" to a "k" and adding an "e". To compare two paths, we first simplify each tag name to a unique identifier of fixed length of q letters. This ensures that comparing longer tags with shorter tags will not perversely affect the distance metric. We then normalize the edit-distance by the maximum length of the two paths to normalize the distance to range between 0.0 and 1.0. For example, with $q = 1$ we convert `html` to `h`, `head` to `e`, and so on. The paths *html/head* and *html/head/title* would first be simplified to *he* and *het*. The edit-distance between the paths is 1, which would then be scaled to $1/3$.

The second term $\frac{|F_i - F_j|}{max(F_i, F_j)}$ of the distance function will be 0.0 for the two subtrees with the same fanout. Conversely, the term will be 1.0 when comparing a subtree with no children to a subtree with 10 children. A similar relationship holds for the third and fourth terms as well.

Intuitively, our distance function is designed to quickly assess the shape of each subtree. We expect subtrees with a similar shape to serve a similar purpose across the set of pages in one page cluster. Note that each unweighted term ranges in value from 0.0 (when the two subtrees share the exact same feature) to 1.0, so the overall weighted distance between any two subtrees also ranges from 0.0 to 1.0. Initially we weight each component equally (i.e. $w_1 = w_2 = w_3 = w_4 = 0.25$). Our experiments (see Section 4) show that this distance metric provides an effective mechanism for identifying common subtrees.

Step 2: Ranking the common subtree sets

This step ranks the k common subtree sets by the likelihood that each contains QA-Pagelets. In THOR we determine the probability that a common subtree set corresponds to a QA-Pagelet by calculating its internal similarity and giving the highest ranking score to the common subtree set that has the lowest internal similarity. We first represent each subtree under consideration by a weighted term vector and then provide the similarity function to compute the internal similarity for each common subtree set.

To determine which common subtrees contain dynamic content generated in response to a query, we first transform each subtree's content into a vector of terms and weights, similar to the method described for tag-tree signature based vector model in Section 3.1.2. Here, we are interested not in each subtree's tags, but only in its content. THOR uses the subtree's content vector for cross-page content analysis to reveal the QA-Pagelets.

We preprocess each subtree's content by stemming the prefixes and suffixes from each term [24]. Then we apply the TFIDF method and the cosine similarity metric to compute the internal similarity of each common subtree set. We have k common subtree sets, each containing n_j subtrees ($1 \leq j \leq k$). Let $subtree_{ij}$ denote the ith subtree in the jth common subtree set. Let N_j denote the total number of distinct terms in the jth common subtree sets. Each of the subtrees is represented by a term-subtree vector:

$$subtree_{ij} = \left\{ \begin{array}{c} (term_1, w_{i1}) \\ (term_2, w_{i2}) \\ \\ \cdots \\ (term_{N_j}, w_{iN_j}) \end{array} \right\}$$

The weight w_{iq} for subtree j is defined using the TFIDF weight function given below:

$$w_{iq} = \log\left(tf_{iq} + 1\right) \cdot \log\left(\frac{n_j + 1}{n_{qj}}\right)$$

where $1 \leq i \leq n_j$, tf_{iq} denotes the frequency of the term indexed by q ($1 \leq q \leq N_j$) in subtree i, n_j is the total number of subtrees in common subtree set j, and n_{qj} denotes the number of subtrees in common subtree set j that contain the term with index q. As discussed before, TFIDF weights terms highly if they frequently occur in some subtrees but infrequently occur across all subtrees. This allows THOR to easily distinguish and to identify the subtrees containing terms that vary from subtree to subtree and thus from page to page.

With the content term-subtree vector model, we can compute the internal similarity for each of the k common subtree sets, denoted by $IntraSubtreeSetSim_i$ ($1 \leq i \leq k$):

$$IntraSubtreeSetSim_i = \sum_{j=1}^{n} \sum_{l \neq j}^{n} sim(subtree_j, subtree_l)$$

Note that the cosine similarity of two identical subtrees is zero. For a particular cluster of similarly-structured domain-specific pages, we expect some subtrees to be relatively static; the QA-Pagelets should vary dramatically from page to page, since it is generated in response to a specific query and these queries differ from page to page. That is, the intra-subtree set similarity should be high for the static subtrees that are the same for all pages. In contrast, for QA-Pagelets, the intra-subtree set similarity should be low since the QA-Pagelets vary in content greatly. Hence, we rank the k subtree sets in ascending order of the intra-similarity of the subsets and then prune out all subtree sets with similarity greater

than 0.5. Our experiments show that the common subtree sets are clearly divided into static-content (high similarity) groups and dynamic-content (low similarity) groups, so that the choice of the exact threshold is not essential.

3.2.2 Selecting the Minimal Subtrees with QA-Pagelets

In this final step of THOR's second phase, the only subtrees left in consideration are those that contain dynamically-generated content. Among these subtrees, some may correspond to QA-Pagelets, some to the QA-Objects within a QA-Pagelet, and some to certain personalized content that is dynamically-generated (perhaps based on cookies maintained at the deep web sites).

We adopt a QA-Pagelet selection criterion that favors subtrees that (1) contain many other dynamically-generated content subtrees; and (2) are deep in the tag tree. The first guideline captures the notion that a QA-Pagelet will contain many sub-elements that are dynamically-generated (which we have termed QA-Objects). The second guideline is designed to discourage the selection of overly large (and broad) subtrees – say, the subtree corresponding to the entire page.

Readers may refer to our technical report [8] for a more detailed discussion on the QA-Pagelet selection.

At the end of our two-phase extraction algorithm, we have a list of QA-Pagelets. There are some circumstances where the subtree corresponding to the QA-Pagelet may contain data in addition to the QA-Pagelet. As a result, we annotate each QA-Pagelet with a list of the other dynamic content subtrees that it contains to guide the QA-Object partitioning stage. These QA-Pagelets are then sent through THOR's third and final stage to partition each QA-Pagelet and generate the QA-Objects.

4. Experiments

In this section we report three sets of experiments. The first set studies the effectiveness of the page clustering phase. In the second set, we investigate the effectiveness of the QA-Pagelet identification phase. The final set of experiments evaluates the overall effectiveness of the THOR system.

Using a breadth first crawl of the Web starting at a seed URL and Google, we identified over 3,000 unique search forms. We randomly selected 50 of the 3,000 search forms. We then submitted to each form 110 queries, each with one keyword selected from 100 random words in the standard Unix dictionary and 10 nonsense words. This results in a set of 5,500 pages in a local cache for analysis and testing. We labeled each page by hand with the appropriate class (e.g. "normal results", "no results", etc.), and identified the QA-Pagelets in each page if they exist. Based on the overall class distribution, we randomly generated three much larger synthetic data sets. If x% of the pages in the set of 5,500 sampled pages belong to class c, approximately x% of the synthetic pages will also belong to class c. To create a new synthetic page of a particular class, we randomly generated a tag and content signature based on the overall distribution of the tag and content signatures for the entire class. We applied this method repeatedly to create data sets of 55,000 pages (1,100 pages per site), 550,000 pages (11,000 pages per site), and 5,500,000 pages (110,000 pages per site).

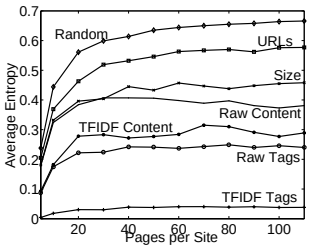

Figure 4: Entropy (a)

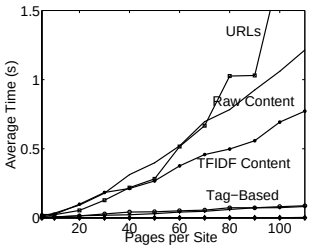

Figure 5: Time (a)

The software underlying the THOR architecture is written in Java 1.4. All experiments were run on a dual-processor Sun Ultra-Sparc III 733MHz with 8GB of RAM. All pages were pre-processed by the HTML Tidy utility [25]. Pages took on average 1.2 seconds to parse.

4.1. Effectiveness of Page Clustering

In the first set of experiments we examine THOR's page clustering phase and evaluate the effectiveness of both the similarity metrics and the clustering algorithm. To show the benefits of our TFIDF tag-tree signature based weighting scheme, we evaluated the entropy and time complexity of our approach against several alternatives mentioned in Section 3.1, including the raw tags, the TFIDF-weighted content, the raw content, the URL, and the size of each page. For the tag and content-based approaches, we generated the vector space representation described in Section 3.1.2. For the URL-based approach, we described each page by its URL and used a string edit distance metric to measure the similarity of two pages. For the size-based approach, we described each page by its size in bytes and measured the distance between two pages by the difference in bytes. As a baseline, we also considered an approach that randomly assigned pages to clusters.

Initially, we selected n pages from each of the 50 collections and ran each set through our enhanced K-Means clustering algorithm. We repeated this process 10 times to generate an average entropy and clustering time for collections ranging from 5 to 110 pages. In Figure 4, we show the average entropy across the 50 page collections for each collection size. Note that our entropy measure ranges from 0 (the best) to 1 (the worst).

Clearly, our TFIDF weighted tag-tree signature approach outperforms all the other techniques. It results in clusters with entropy on the order of 6-times lower (0.04 versus

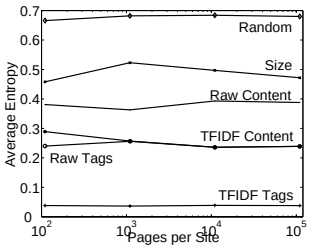

Figure 6: Entropy (b)

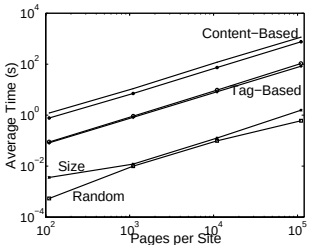

Figure 7: Time (b)

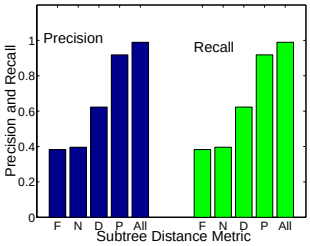

Figure 8: Distance Comparison

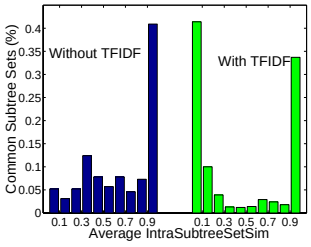

Figure 9: Subtree Similarity

0.24) than the raw tags technique, between 7 and 10-times lower than the content-based techniques (0.04 versus 0.28 and 0.38), and between 11 and 17-times lower than the size, URL, and random approaches (0.04 versus 0.44, 0.56, and 0.65). There are at least two reasons for such success. First, the objective of THOR's page clustering is simply to separate answer pages with query matches from those with no matches or errors. Thus the tag-tree signature approach is sufficient and effective. Second, the use of TFIDF to weight each tag-tree signature accentuates the distance between different classes, resulting in very successful clustering.

Note that the average entropy is fairly low when clustering few pages, then increases sharply before levelling off at around 40 pages. For small page samples, we would expect that very few different page classes would be represented; hence any clustering should result in low entropy. As the number of pages to be clustered increases so does the diversity of the page classes represented in the sample; so entropy should increase until the distribution of pages stabilizes.

In Figure 5, we show the average time to run one iteration of our page clustering algorithm for each of the sample sizes over the 50 collections and compare it with the alternative clustering approaches. The tag-based approaches take on average an order-of-magnitude less time to complete than the content-based approaches, and can scale better to much larger data sets. The tag-based approach dominates the content-based approach primarily due to the sharp difference in size between the tag and the content signature. On average, each page in our collection of 5,500 pages contains 22.3 distinct tags and 184.0 distinct content terms.

To further understand the time complexity of tag-tree based approach, we compared our tag-tree signature approach with a tree-edit distance metric based approach [23]. We found that for a single collection of 110 pages, tree-edit distance based clustering took between 1 and 5 hours, whereas our TFIDF-tag approach took less than 0.1 seconds. Given the excessive cost of the tree-edit-distance, we did not consider it in our other experiments.

To assess the efficiency and scalability of our algorithm, we next compared the alternative approaches against the TFIDF-weighted tag signature approach over the three synthetic data sets. In Figure 6, we report the average entropy over each of the 50 collections, where we consider from 110 to 110,000 pages per collection. The average entropy is nearly constant, even though the collections grow by 1,000 times. In Figure 7, we report the average time to run one iteration for each of the 50 collections, again considering the three synthetic data sets. The average clustering time grows linearly with the increase in collection size, as we would expect in a K-Means-based clustering algorithm. Overall, these results confirm that the TFIDF-weighted tag signature approach grows linearly as the size of the page collection grows by three orders of magnitude, and thus it can scale up smoothly.

Finally, we conducted extensive experiments with various cluster settings – ranging the number (k) of clusters from 2 to 5 and ranging the internal cluster iterations from 2 to 20. We found that varying the cluster number resulted in only minor changes to the overall performance of the system. If we set the number of clusters greater than the number of actual clusters, the clustering algorithm will merely generate more refined clusters. This is not a problem in our context, since QA-Pagelet identification is dependent only on the quality of each cluster; a sufficiently good cluster will yield reasonable results regardless of the grain of the cluster. We also found that running the clusterer 10 times provided a balance between the faster running times using fewer iterations and the increased cluster quality using more iterations.

4.2. Effectiveness of QA-Pagelet Identification

In the second set of experiments, we assess precision and recall of the QA-Pagelet identification phase. To consider this phase in isolation from the page clustering phase, we considered as input only those pages from each site that had been pre-labeled as containing QA-Pagelets. In the final set of experiments, we combine the two phases to report THOR's overall performance.

We measure precision and recall as:

$$Precision = \frac{\text{Number of QA-Pagelets Correctly Identified}}{\text{Number of Subtrees Identified as QA-Pagelets}}$$

$$Recall = \frac{\text{Number of QA-Pagelets Correctly Identified}}{\text{Total Number of QA-Pagelets in the Set of Pages}}$$

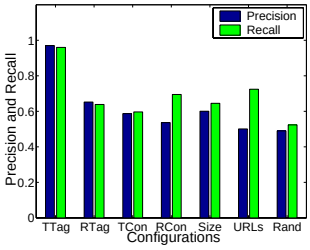

Figure 10: Overall P/R

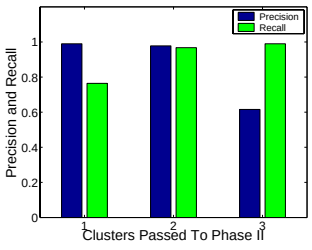

Figure 11: P/R Trade-off

To validate our choice of subtree distance function (see Section 3.2.1), we considered several variations: a distance based solely on each of the four subtree features – path (P), fanout (F), depth (D), and nodes (N) – and our distance based on a linear combination of all four (All). In Figure 8, we report the precision and recall of THOR's second phase with respect to the five distance metrics. As we might expect, simply judging subtree similarity by a single feature underperforms the combined metric. Our combined metric achieves precision and recall over 98%.

For our combined distance metric, on a careful inspection of the mis-labeled pages, we discovered that THOR was sometimes confused by pages with a region of dynamic non-query-related data. For example, some pages generate an advertisement region that varies somewhat across the space of pages. As a result, the intra-cluster content analysis may incorrectly identify the dynamic advertisement as a QA-Pagelet.

We now illustrate the role of our TFIDF-weighting scheme for computing the internal similarity of the common subtree sets. We show in the left-hand side of Figure 9 a histogram of the intra-similarity scores for the common subtree sets with no TFIDF-weighting. The number of common subtree sets with high dissimilarity is very low, whereas the number of common subtree sets with high similarity is very high. Extracting the QA-Pagelets in this case would be prohibitive. In contrast, in the right-hand side of Figure 9 we show a histogram of the intra-similarity scores for the common subtree sets *using* our TFIDF-weighting scheme. Note that there are many subtree sets at both the low end and the high end of the similarity scale. This shows that the common subtree sets clearly either have query-independent static content (i.e. high similarity) or query-dependent dynamic content (i.e. low similarity). The precise choice of threshold – 0.5 in the first prototype of THOR – is not very important.

4.3. Effectiveness of the Two-Phase Approach

In the final set of experiments, we report the overall performance of THOR's two-phase extraction algorithm. In Figure 10, we compare the overall impact on precision and recall of THOR's TFIDF-tag clustering approach ($TTag$) versus the raw tag ($RTag$), TFIDF content ($TCon$), raw content ($RCon$), size, URLs, and random approaches. In all cases we used the combined subtree distance metric discussed above. The overall THOR approach achieves very high precision (97%) and recall (96%) in contrast to the alternatives, mostly since the quality of clusters generated in THOR's first phase doubly impacts the overall performance of the system. First, if a normal results page is misclustered into a "no results" cluster, it won't advance to the QA-Pagelet identification phase, and hence its QA-Pagelets will be overlooked. Second, any "no results" pages that do advance to the second phase will worsen QA-Pagelet identification by hampering the cross-page analysis. Hence, it is of critical importance to generate quality clusters in the first phase.

Finally, in Figure 11 we show the trade-off in precision and recall as a function of the number of clusters passed from the page clustering phase to the QA-Pagelet identification phase, considering only the TFIDF-tag approach. In this experiment, the page clustering phase generates three clusters. If we pass along only one cluster to the second phase, the precision will be very high, since only pages that contain QA-Pagelets will be considered in the second phase. In contrast, recall will be much lower since many QA-Pagelets may occur in one of the clusters not passed on to the second phase. The reverse holds when we pass all three clusters on to the second phase. Precision falls, since many of the pages in consideration do not contain QA-Pagelets, but recall increases since every page in the original data set will be considered for QA-Pagelet extraction. In this case, there is a good compromise when passing two clusters.

5. Related Work

The WHIRL system [9] uses tag patterns and textual similarity of items stored in a deductive database to extract simple lists or lists of hyperlinks. The system relies on previously acquired information in its database in order to recognize data in target pages. For data extraction across heterogeneous collections of deep web databases, this approach is infeasible.

RoadRunner [10] automatically generates wrappers for extracting data from web pages. The RoadRunner algorithm compares pages generated by the same query form and constructs a regular expression based on the differences between the pages. Similarly, Arasu and Garcia-Molina [1] have developed an extraction algorithm that models page templates and uses equivalence classes for data segmentation. In both cases, there is an assumption that all pages have been generated by the same underlying template, whereas our THOR system automatically partitions a set of diverse pages into structurally-similar groupings. Secondly, the two techniques make no attempt to identify the primary query-related content on a page.

Bar-Yossef and Rajagopalan [2] call the functionally distinct portions of a page pagelets. They use this formulation to guide template discovery, which is ancillary to the data extraction problem. Their template discovery algorithm relies on close content similarity between pagelets, whereas we consider both structural and content attributes of a page and its component subtrees.

6. Conclusions

The continued growth of the Deep Web poses a major challenge for searching the Deep Web, integrating deep web data from multiple sources, and reusing deep web data across various applications. In this paper, we have presented a scalable and efficient mining system for discovering and extracting content-rich query-related information from the Deep Web. Our THOR system relies on a novel two-phase extraction approach to locate QA-Pagelets. In the first phase, pages from each web site are grouped into clusters of structurally-similar pages by combining a tag-tree signature based vector model of pages, a TFIDF-based cosine similarity function, and a simple k-means clustering algorithm, resulting in a ranked list of page clusters. In the second phase, pages from the top-ranked page clusters are examined through a subtree filtering algorithm. Our experiments show that THOR is robust against changes in presentation and content of deep web pages, and scales well with respect to the growing number of deep web sources.

7. Acknowledgments

The first author would like to thank Ashwin Ram for many helpful suggestions. The authors are partially supported by NSF through a CCR grant and an ITR grant, DoE under SciDAC, and DARPA under PECS.

References

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD '03*.

[2] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *WWW '02*.

[3] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Knowledge Discovery and Data Mining*, pages 407–416, 2000.

[4] M. Bergman. The deep web: Surfacing hidden value. *BrightPlanet*, 2000.

[5] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *SIGIR '98*.

[6] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *WWW '97*.

[7] J. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *SIGMOD '99*.

[8] J. Caverlee, L. Liu, and D. Buttler. Probe, cluster, and discover: Focused extraction of QA-Pagelets from the Deep Web. Technical report, GIT, 2003.

[9] W. Cohen. Recognizing structure in web pages using similarity queries. In *AAAI '99*.

[10] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB '01*.

[11] J. Dean and M. R. Henzinger. Finding related pages in the World Wide Web. In *WWW '99*.

[12] I. Dhillon, J. Fan, and Y. Guan. Efficient clustering of very large document collections. In R. Grossman, G. Kamath, and R. Naburu, editors, *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.

[13] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley and Sons, New York, 2001.

[14] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Hypertext '98*.

[15] D. Hawking and P. Thistlewaite. Methods for information server selection. *ACM TOIS*, 17(1):40–76, 1999.

[16] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB '02*.

[17] P. G. Ipeirotis, L. Gravano, and M. Sahami. QProber: A system for automatic classification of hidden-web databases. *ACM TOIS*, 21(1):1–41, 2003.

[18] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[19] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[20] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *WWW '99*, May 1999.

[21] V. I. Levenshtein. Binary codes capable of correcting deletions, insertion and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[22] W. Meng, C. T. Yu, and K.-L. Liu. Detection of heterogeneities in a multiple text database environment. In *CoopIS '99*.

[23] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *WebDB 2002*.

[24] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[25] D. Raggett. *Clean Up Your Web Pages with HTML TIDY*. http://www.w3.org/People/Raggett/tidy/, 1999.

[26] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. In *Readings in Information Retrieval*. Morgan Kauffman, San Francisco, CA, 1997.

[27] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *CACM*, 18(11):613–620, 1971.

[28] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[29] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000.

[30] WWW Consortium. *Document Object Model*. http://www.w3.org/DOM/.

[31] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *SIGIR '98*.

[32] Y. Zhao and G. Karypis. Criterion functions for document clustering: Experiments and analysis. Technical report, University of Minnesota, Department of Computer Science, Minneapolis, MN, 2002.