QA-Pagelet: Data Preparation Techniques for Large Scale Data Analysis of the Deep Web

James Caverlee and Ling Liu
College of Computing, Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA 30332
{caverlee, lingliu}@cc.gatech.edu

Abstract

This paper presents the QA-Pagelet as a fundamental data preparation technique for large scale data analysis of the Deep Web. To support QA-Pagelet extraction, we present the Thor framework for sampling, locating, and partioning the QA-Pagelets from the Deep Web. Two unique features of the Thor framework are (1) the novel page clustering for grouping pages from a Deep Web source into distinct clusters of control-flow dependent pages; and (2) the novel subtree filtering algorithm that exploits the structural and content similarity at subtree level to identify the QA-Pagelets within highly ranked page clusters. We evaluate the effectiveness of the Thor framework through experiments using both simulation and real datasets. We show that Thor performs well over millions of Deep Web pages and over a wide range of sources, including eCommerce sites, general and specialized search engines, corporate websites, medical and legal resources, and several others. Our experiments also show that the proposed page clustering algorithm achieves low-entropy clusters, and the subtree filtering algorithm identifies QA-Pagelets with excellent precision and recall.

Index Terms: Deep Web, Data Preparation, Data Extraction, Pagelets, Clustering

I. Introduction

One of the most promising new avenues of large scale data analysis is the large and growing collection of Web-accessible databases known as the Deep Web. Unlike the traditional or "surface" Web – where Web pages are accessible by traversing hyperlinks from one page to the next – Deep Web data is accessible by interacting with a Web-based query interface. This Deep Web provides access to huge and growing data repositories, offering a tremendous opportunity for a new generation of large scale data analysis – from supporting advanced search operators, to data integration, to data mining applications, and so on.

* This work is partially supported by the National Science Foundation under a CNS Grant, an ITR grant, and a DoE SciDAC grant, an IBM SUR grant, an IBM faculty award, and an HP equipment grant. The authors wish to thank the reviewers and the special issue guest editors for their helpful comments.

For the Deep Web to be successfully leveraged as an information platform, we assert that a fundamental data preparation approach is the identification and extraction of the high-quality query-related content regions in Deep Web pages (we call them the *Query-Answer Pagelets* or *QA-Pagelets*, for short). Deep Web pages tend to consist of one or more primary content regions that are directly related to the user query, and other less relevant data regions like advertisements, navigational elements, and standard boilerplate that obscure the high-quality query-related data that is of primary interest to the user. Data preparation of the query-related information-rich content provides the basis for the rest of any Deep Web data analysis platform by supplying the basic building blocks for supporting search, composition, and reuse of Deep Web data across various applications. Deep Web data preparation is especially challenging since the algorithms should be designed independently of the presentation features or specific content of the web pages, such as the specific ways in which the query-related information is laid out or the specific locations where the navigational links and advertisement information are placed in the web pages.

To support large scale data analysis over the Deep Web, we have begun a research effort called **Thor** for extracting, indexing, searching, and supporting advanced operators like composition and comparison over Deep Web sources. Thor is designed as a suite of algorithms to support the entire scope of a Deep Web information platform. In this paper, we present the fundamental data preparation component – the Thor data extraction and preparation subsystem – that supports a robust approach to the sampling, locating, and partitioning of the QA-Pagelets in four phases.

- **Collecting Deep Web Pages:** The first phase probes the Deep Web data sources to sample a set of pages rich in content.
- **Clustering Deep Web Pages:** The second phase uses a novel page clustering algorithm to segment Deep Web pages according to their control-flow dependence similarities, aiming at separating pages that contain query matches from pages that contain no matches.
- **Identifying QA-Pagelets:** In the third phase, pages from top-ranked page clusters are examined at a subtree level to locate and rank the query-related content regions of the pages (the *QA-Pagelets*).
- **Partitioning QA-Pagelets:** The final phase uses a set of of partitioning algorithms to separate and locate itemized objects in each QA-Pagelet.

In our experiments section, we show that Thor performs well over millions of Deep Web pages and over a wide range of sources, including eCommerce sites, general and specialized search engines, corporate websites, medical and legal resources, and several others. Our experiments show three important and interesting results. First, our page cluster algorithm achieves high quality of clusters with very low entropy across a wide range of sources. 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.

II. Related Work

The notion of a Deep (or Hidden or Invisible) Web has been a subject of interest for a number of years, with a growing body of literature. A number of researchers have discussed the problems inherent in accessing, searching, and categorizing the contents of Deep Web databases. The problem of automatically discovering and interacting with Deep Web search forms is a problem examined in [19]. Other researchers [11] have proposed methods to categorize Deep Web sites into searchable hierarchies. More recently, there have been efforts to improve source selection for metasearching Deep Web sources [16] and to match Deep Web query interfaces [25], [23], [22].

As a first step towards supporting Deep Web data analysis, several hand-tuned directories have been built (e.g. `www.completeplanet.com` and `www.invisibleweb.com`), and several domain-specific information integration portal services have emerged, such as the NCBI bioportal. These integration services offer uniform access to heterogeneous Deep Web collections using *wrappers* – programs that encode semantic knowledge about specific content structures of web sites, and use such knowledge to aid information extraction from those sites. The current generation of wrappers are mainly semi-automatic wrapper generation systems (e.g. [1], [15]) that encode programmers' understanding of specific content structure of the set of pages to guide the data extraction process. As a result, most data analysis services based on wrappers have serious limitations on their breadth and depth coverage.

Most closely related to Thor are a number of data extraction approaches that emphasize identifying and extracting certain portions of web data. For example, 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 to recognize data in target pages. For data extraction across heterogeneous collections of Deep Web databases, this approach is infeasible. Similarly, Arasu and Garcia-Molina [2] 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 Thor automatically partitions a set of diverse pages into control-flow dependent groupings. Thor builds on previous work in a related publication [8].

Bar-Yossef and Rajagopalan [4] 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.

III. Preliminaries

In this section, we define the fundamental representations used by Thor. We begin by discussing the modeling of web pages as tag trees, including the definition of several related components. We build on these representations to develop our notion of a QA-Pagelet.

A. Modeling Web Pages as Tag Trees

Using a variation of the well-known Document Object Model, we model a web page as a tag tree consisting of tags and text. By tag, we mean all of the characters between an opening bracket "$<$" and a closing bracket "$>$", where each tag has a tag name (e.g. BR, TD) and a set of attributes. The text is the sequence of characters between consecutive tags.

To convert a web page into a tag tree requires that the page be well-formed. The requirements of a well-formed page include, but are not limited to, the following: all start tags, including standalone tags, must have a matching end tag; all attribute values must be in quotes; tags must strictly nest. Pages that do not satisfy these criteria are automatically transformed into well-formed pages using Tidy [`http://tidy.sourceforge.net/`]. A well-formed web page can be modeled as a tag tree $\mathcal{T}$ consisting 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

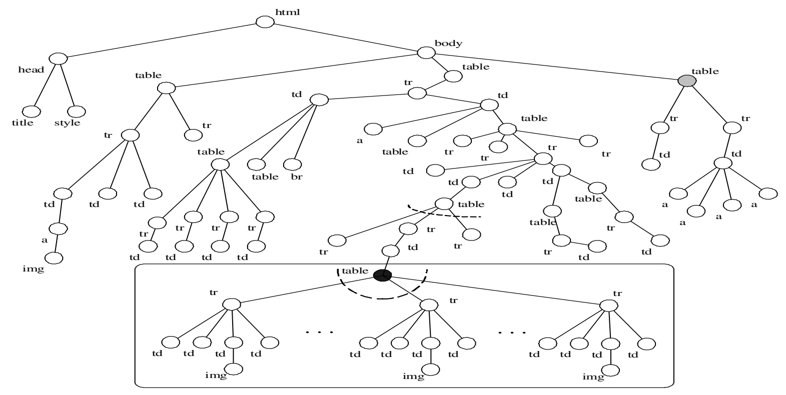

Fig. 1. Sample Tag Tree from IBM.com

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.

Definition 1 (Tag Tree): A tag tree of a page p is defined as a directed tree $\mathcal{T} = (V, E)$ *where* $V = V_T \cup V_C$*,* V_T *is a finite set of tag nodes and* V_C *is a finite set of content nodes;* $E \subset (V \times V)$*, representing the directed edges.* $\mathcal{T}$ *satisfies the following conditions:* $\forall (u, v) \in E, (v, u) \notin E$*;* $\forall u \in V, (u, u) \notin E$*; and* $\forall u \in V_C$*,* $\nexists v \in V$ *such that* $(u, v) \in E$*.*

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. For a given node, the path expression from the root of the tree to the node can uniquely identify the node. In Figure 1, the path from the root node to the title node can be expressed as $html \Longrightarrow^* \ title$. By using an XPath-style notation, the example path could also be written as `html/head/title`. Whenever a node has siblings of the same tag name, we assign a number to uniquely identify the appearance order of the node in the tag tree, e.g. `html/body/table[3]` refers to the gray-shaded table node in Figure 1.

We denote the number of children of a node u as $fanout(u)$. For any node $u \in V$, $fanout(u)$ denotes the cardinality of the set of children of u. $fanout(u) = |children(u)|$ if $u \in V_T$ and $fanout(u) = 0$ if $u \in V_C$. We denote the content of a node u as $content(u)$. For any node $u \in V$, if $u \in V_C$, then $content(u)$ denotes the content of node u. Otherwise, if $u \in V_T$, then $content(u) = \emptyset$. Additionally, each node in a tag tree corresponds to the root of a subtree in the tag tree. For a tag tree $\mathcal{T} = (V, E)$, the total number of subtrees is $|V|$, where a subtree is defined as:

Definition 2 (Subtree): Let $\mathcal{T} = (V, E)$ be the tag tree for a page d, and $\mathcal{T}' = (V', E')$ is called a subtree of $\mathcal{T}$ anchored at node u, denoted as $\text{subtree}(u)$ *($u \in V'$), if and only if the following conditions hold: (1) $V' \subseteq V$, and $\forall v \in V, v \neq u$, if $u \Longrightarrow^* v$ then $v \in V'$; and (2) $E' \subseteq E$, and $\forall v \in V', v \neq u, v \notin V_C, \exists w \in V', w \neq v$, and $(v, w) \in E'$*

The content of a subtree rooted at node u, denoted $subtreeContent(u)$, is the union of the content of all nodes in the subtree. For any node $u \in V$, if $u \in V_C$, then $subtreeContent(u) = content(u)$. Otherwise, if $u \in V_T$, then $subtreeContent(u)$ is the union of the node content of all the leaf nodes reachable from node u, i.e. $subtreeContent(u) = \cup(content(v))$, $\forall v \in V_C$ such that $u \Longrightarrow^* v$.

B. Modeling the QA-Pagelet

Since Deep Web pages are generated in response to a query submitted through a web interface, we are interested in locating the portion of a Deep Web page that *answers* the query. Hence, we refer to the content-rich regions in a Deep Web page that contain direct answers to a user's query as *Query-Answer Pagelets* (or *QA-Pagelets* for short). The term *pagelet* was first introduced in [4] to describe a region of a web page that is distinct in terms of its subject matter or its content. Critical to our formulation of a QA-Pagelet is a minimal subtree:

Definition 3 (Minimal Subtree with Property P): Let $\mathcal{T} = (V, E)$ be the tag tree for a page p, and subtree$(u) = (V', E')$ be a subtree of $\mathcal{T}$ anchored at node u. We call subtree(u) a minimal subtree with property P, denoted as subtree(u, P), if and only if $\forall v \in V, v \neq u$, if $\text{subtree}(v)$ *has the property P, then $v \Longrightarrow^* u$ holds.*

In Thor, we use the term QA-Pagelet to refer to a dynamically-generated content region that contains matches to a query.

Definition 4 (QA-Pagelet): A QA-Pagelet is a minimal subtree that satisfies the following two conditions: (1) A QA-Pagelet is dynamically-generated in response to a query; and (2) it is a page fragment that serves as the primary query-answer content on the page.

Condition 1 of the definition 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. Condition 2 of the definition 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.

IV. QA-PAGELET ARCHITECTURE

We now present the overall framework for supporting large scale data analysis of the Deep Web, with an emphasis on the core data preparation modules, as illustrated in Figure 2. The **Thor** framework is intended to provide a wide variety of Deep Web analysis services, including data extraction, indexing, searching, Deep Web mining, and supporting advanced operators like composition and comparison over Deep Web sources. To support these services, we have developed a fundamental data preparation component – the Thor data extraction and preparation subsystem – that supports a robust approach to the sampling, locating, and partitioning of the QA-Pagelets. This data preparation is critical to the efficient and effective deployment of tools that operate over Deep Web data.

In the rest of this section, we present the intuition behind the Thor data preparation algorithms and the detailed QA-Pagelet extraction architecture. The QA-Pagelet algorithms are motivated by the observation that even though the Web as a whole is a mélange of content, structure, and organization, in the small, however, particular subsets tend to be very closely related both in terms of structure and content. We identify two levels of relevance that lay the foundation for our data extraction algorithm:

Control-Flow Relevance: Unlike pages on the surface Web, Deep Web pages are generated in response to a query submitted through a web interface. Although the underlying page generation process is hidden from the end user, we observe that many Deep Web sources tend to structure particular classes of pages in a similar fashion (often using templates), reflecting the underlying control flow process. For example, the control flow process that generates a "no matches" page differs from the control flow process that generates a list of matches. These differences in control flow are reflected in the layout and structure of the resulting pages. Hence, by analyzing the structure of the resulting pages, we can group pages – such as single match pages, list of matches pages, and "no matches" pages – that were generated by a common control flow process.

Topical Diversity: Given a group of pages generated by a common control flow process at a particular Deep Web source – say, a set of n normal answer pages from eBay, each generated in response to

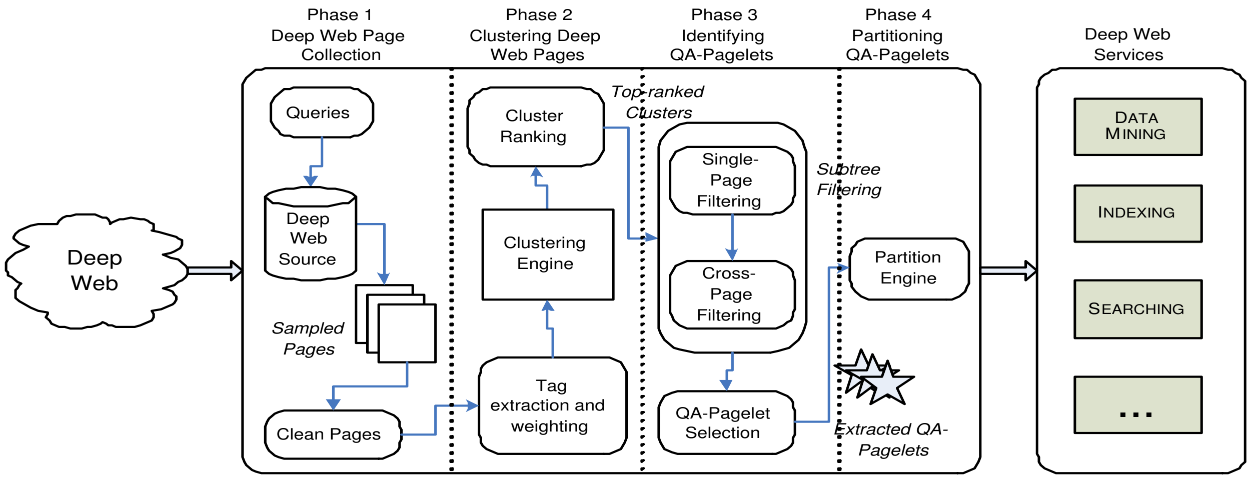

Fig. 2. Overall Architecture

a different query – the topical relevance across pages may yield clues as to which fragments of a page contain QA-Pagelets. Some portions of the page contain topically-related information that is common across all pages in the cluster, while other portions are dynamically-generated in response to a particular user query, and hence, topically-diverse. Each of the n pages from eBay may display different content due to queries with different search keywords, but each page also has a similar content layout and a similar page presentation structure. The navigation bar, advertisements, and other boilerplate often co-occur across all of the pages.

These observations naturally suggest the approach we take in Thor's four phases, as illustrated in Figure 2. The first phase collects the sample answer pages generated in response to queries over a Deep Web source. The second phase clusters the sample answer pages into distinct groups of pages that are linked by their common control-flow dependence, each corresponding to one type of answer page: be it multi-match pages, single-match pages, no-match pages, or exception pages. The third phase identifies the QA-Pagelets in each highly-ranked page cluster by partitioning the set of subtrees in a single page cluster into a list of common subtree sets ranked by their topical diversity. Each common subtree set corresponds to one type of content region in the set of control-flow dependent answer pages. We then use intra-cluster content metrics to filter out the common content and hone in on the QA-Pagelets. At the conclusion of the third phase, Thor recommends a ranked list of QA-Pagelets. The fourth phase partitions the highly-ranked QA-Pagelets into itemized QA-Objects, which are in turn fed into the rest of the Thor Deep Web information platform.

Since the data preparation algorithms in phases 2 and 3 are the core Thor data preparation modules, we briefly summarize the first and fourth phase below before continuing with a detailed discussion of the core Thor modules in the following sections.

Collecting Deep Web Pages: In this first phase for QA-Pagelet extraction, we collect sample answer pages by probing a Deep Web source with a set of carefully designed queries to generate a representative set of sample answer pages. Efficiently and appropriately probing a data source is an area of active research with a number of open issues. In our context, we are interested in probing techniques that scale well to many sources *and* that support the generation of a variety of page types to capture all possible classes of possible control-flow answer pages. In the first 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 source 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. In the future, we anticipate expanding our probing approach to include domain-specific probing techniques to increase the coverage of sources. The sampled pages are then passed to the second and third phases to perform the page clustering and to extract the QA-Pagelets.

QA-Object Partitioning: The extracted QA-Pagelets are finally partitioned into itemized QA-Objects, which are in turn fed into the rest of the Thor Deep Web information platform. 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). These QA-Objects may then be incorporated into advanced services, for example for searching the Deep Web, integrating Deep Web data from multiple database sources, or reusing Deep Web data across various applications. A critical challenge of the QA-Object partitioning phase is to develop methods that are robust to the various types of QA-Objects. Some may simply be lists of hyperlinks that are neatly divided by the HTML list tag `<li>`. Others may be complicated nested table structures. To isolate the QA-Objects within a QA-Pagelet, the fourth phase first takes into consideration a list of recommended QA-Objects from Thor's second stage, examines each candidate's particular structure, and then searches the rest of the QA-Pagelet for similar structures using several different metrics that assess the size, layout, and depth of potential QA-Objects.

In the next two sections, we discuss the core Thor phases – the Deep Web Page Clustering Phase and the QA-Pagelet Identification Phase – in great detail.

V. Deep Web Page Clustering Phase

In this section we describe our design consideration by analyzing several known page clustering approaches with respect to the extraction of QA-Pagelets. We then present our page clustering algorithm, the ranking criteria, and the metrics used for evaluating the quality and effectiveness of our page clustering algorithm. The goal of the page clustering phase is to take the collection of n sample pages generated in response to probing queries over a single Deep Web source and automatically cluster the set of pages into groupings based upon common control-flow dependencies, and then give higher ranking to those page clusters that more likely to contain content-rich QA-Pagelets.

For example, the music site AllMusic.com generates at least three page types, where each corresponds to a distinct control-flow: a multi-matches page consisting of a list of query matches; a single match page with detailed information on a particular artist (e.g. Elvis Presley); and a no matches page. These structural and presentational differences reflect the differing underlying control flow processes that generate each page. By grouping pages by type, we may apply QA-Pagelet extraction techniques that are specialized for each group of pages, resulting in more accurate and efficient extraction.

A. The Page Clustering Problem

In our context, we define Deep Web page clustering as follows: Given a set of sample pages from a particular Deep Web source, which approach is most effective for clustering the pages into control-flow dependent 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: $\mathcal{C} = \{Cluster_1, \ldots, Cluster_i, \ldots, Cluster_k \mid \bigcup_{i=1}^{k} Cluster_i = \{p_1, ..., p_n\} \text{ and } Cluster_i \cap Cluster_j = \emptyset\}$.

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.

B. Alternative Approaches

Below we briefly discuss several popular page clustering approaches, emphasizing on their similarity metrics and discuss how well these concrete similarity metrics meet the specific objectives of page clustering in the context of QA-Pagelet extraction and the impact of the similarity metric on the effectiveness of the page clustering algorithm.

1) URL-based: Clustering based on URL-similarity is effective for partitioning pages from different Deep Web databases by their point of origin [5], but it is not suitable for focused extraction of QA-Pagelets for a number of reasons. First, different queries over a single Deep Web database often result in pages with similar URLs. Second, pages with similar URLs may be generated using totally different templates and present completely different types of data contents. For example 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`. The two queries generate pages belonging to two distinct classes – the normal listing of results for `Superman` versus a "no matches" page for the nonsense word. Furthermore, the class of page returned by a single query may change over time, even though the URL may remain exactly the same. In either case, the URLs alone provide insufficient information to distinguish between the two classes.

2) Link-based: Link-based approaches cluster pages based on their similarity of both in-links (citations) and out-links (references). Various link-based approaches have been proposed that are extremely effective for discovering groups of related pages (e.g. [6], [13], [14]). These techniques represent the Web as a graph where pages are nodes and hyperlinks are edges. Clusters are deduced by analyzing the graph (or link) structure. However, link-based approaches are not suitable for clustering dynamically generated Web pages. Most of the dynamic pages from the Deep Web tend to have few or no links pointing to them since the pages are generated dynamically. It is not a surprise that a collection of pages from the Deep Web have no common in-links. Additionally, some Deep Web sources like Google or AllTheWeb generate answer pages with out-links to external domains but lack sufficient in-links pointing to them. As a result, the link-based approaches are not effective for focused extraction of QA-Pagelets from the Deep Web since the underlying link-based graph structure is sparse and possibly populated with too many disconnected nodes.

3) Content-based: Content-based clustering techniques rely on discovering common characteristics of data content presented in the pages to glean similarities. There are many flavors of the content-based approach. The technique presented in [7] performs a syntactic comparison between pages to find clusters of similar pages. Alternatively, the technique presented in [24] uses a suffix-tree representation to find groups of similar documents. These techniques may be effective at finding pages that share similar content, but they do poorly in separating answer pages that contain query matches from those exception pages such as error reporting pages or "no matches" pages. This is because the pages to be clustered are generated in response to different queries and, by design, tend to differ greatly in content. Hence, we expect content-based methods to be poor differentiators of structurally-similar pages.

4) Size-based: A clustering technique based solely on the size of pages – thinking that "no matches" pages are significantly smaller than normal answer pages with multiple matches – will fail when page size is not a clear predictor of the structural similarity of pages. In many Deep Web sources an answer page with a single match returned in response to a query is nearly equal in size to a page with no matches, even though the two pages are clearly generated using two different templates and should belong to two different classes.

C. The Thor Page Clustering Algorithm

The common theme in each of the techniques discussed above is the reliance on non-structural similarity characteristics of the pages to find groupings. In contrast, Thor relies on a tag-tree based clustering approach to group pages with similar tag-tree representations into clusters, to reflect the underlying control flow process. 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 (corresponding to the underlying control flow) useful for clustering different groups of answer pages provided by a Deep Web source in response to a query. Second, pages generated using different templates tend to be produced by different control flow processes and have very different tag-tree structures.

We next introduce the three basic components of Thor's second phase: the tag-tree signature, the tag-tree similarity metric, and the page clustering algorithm using tag-tree signatures. The tag-tree similarity metric is introduced in the context of a vector-space model and relies on two conventional similarity functions. The page clustering algorithm relies on two well-known clustering algorithms – Simple K-Means and Bisecting K-Means.

Tag-Tree Signature

To capture the structural similarity among pages generated by a common control flow process at a Deep Web source, we adopt a tag-tree based similarity metric. 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 [3]. Given a set of n 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 source-specific page space. Concretely, we use a variation of TFIDF, which defines the weight for tag k in page i as $w_{ik} = TF \cdot IDF = \log(tf_{ik} + 1) \cdot \log((n + 1) / n_k)$, 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). For web languages that allow user-defined tags like XML and XHTML , we may expect to find unique tags in different classes of pages, meaning the TF and IDF factors will each play a significant weighting role. Alternatively, since the set of possible HTML tags is limited, we might expect many HTML tags to occur across all pages, regardless of the page class. 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. And for those sets of HTML pages in which certain tags occur in some but not all pages, the IDF factor will have added distinguishing power. Empirically, we have found that many web sources do indeed use unique tags for some classes of pages but not all, enhancing the quality of the TFIDF weighting.

Interesting to note is that, although the tag-tree signature does not capture the subtle differences in content between two pages (say two otherwise perfectly similar pages, but with a pair of transposed subtrees), it is effective to distinguish those pages that contain query matches from those that do not. Our experiments in Section VIII show that the tag-tree based page clustering is more than sufficient to separate pages generated from different templates of a Deep Web source into different clusters. Pages with query matches will be distinguished from those with no matches.

Tag-Tree Similarity

Given a tag-tree signature vector for all pages, we can then compute the similarity between pages using a number of similarity (or distance) metrics. In this paper we consider two metrics: (1) the cosine similarity because of its appealing characteristics and widespread use in the information retrieval community; and (2) the well-known Euclidean distance.

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: $sim_{Cos}(p_i, p_j) = \frac{\sum_{k=1}^{N} w_{ik} w_{jk}}{\sqrt{\sum_{k=1}^{N} w_{ik}^2} \cdot \sqrt{\sum_{k=1}^{N} w_{jk}^2}}$ and the Euclidean distance function is: $distance(p_i, p_j) = (\sum_{k=1}^{n} |w_{ik} - w_{jk}|^2)^{\frac{1}{2}}$.

Orthogonal page vectors in our normalized space will have a cosine similarity of 0.0 (and a non-zero Euclidean distance), whereas identical page vectors will have a cosine similarity of 1.0 (and a Euclidean distance of 0.0). In the first prototype of Thor, we define our tag-tree based similarity metric

```
SimpleKMeans(Number of Clusters $k$, Input Vectors $\mathcal{D}$)
    Let $\mathcal{D} = \{d_1, ..., d_n\}$ denote the set of $n$ page vectors
    Let $N$ denote the total number of distinct tags in $\mathcal{D}$
    Let $d_j = < (tag_1, w_j1), \ldots, (tag_N, w_{jN}) >$ denote a
      page vector of $N$ elements, $w_{jl}$ is the TFIDF weight of
      the $tag_l$ in page $j$ $(l = 1, \ldots, N)$
    Let $\mathcal{C} = \{C_1, ..., C_k\}$ denote a clustering of $\mathcal{D}$ into $k$ clusters.
    Let $\mu_i$ denote the center of cluster $C_i$
    foreach cluster $C_i$
        Randomly pick a page vector, say $d_j$ from $\mathcal{D}$
        Initialize a cluster center $\mu_i = d_j$, where $d_j \in \mathcal{D}$
    repeat
        foreach input page vector $d_j \in \mathcal{D}$
            foreach cluster $C_i \in \mathcal{C}$ $i = 1, \ldots, k$
              compute $\delta_i = sim(d_j, mu_i)$
            if $\delta_h$ is the smallest among $\delta_1, \delta_2, \ldots, \delta_k$
              $mu_h$ is the nearest cluster center to $d_j$
            Assign $d_j$ to the cluster $C_h$
        // refine cluster centers using centroid of each cluster
        foreach cluster $C_i \in \mathcal{C}$
            foreach tag $l$ in $d_j$ $(l = 1, \ldots, N))$
              $cw_{ij} \leftarrow \frac{1}{|C_i|} \sum_{l=1}^{N} w_{jl}$
            $\mu_i \leftarrow < (tag_1, cw_{i1}), \ldots, (tag_N, cw_{iN}) >$
    until cluster centers no longer change
    return $\mathcal{C}$
```

Fig. 3. Tag Tree Signature Simple K-Means Page Clustering Algorithm

by combining the tag-tree signature vector model of the pages, the TFIDF weight system, and the cosine (or Euclidean distance) function. 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 this paper, we consider two popular clustering algorithms: (1) Simple K-Means; and (2) Bisecting K-Means.

Simple K-Means: Simple K-Means is appealing since it is conceptually very 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: $centroid_C = \left\{(tag_1, \frac{1}{|C|} \sum_{i \in C} w_{i1}), \cdots, (tag_N, \frac{1}{|C|} \sum_{i \in C} w_{iN})\right\}$, 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. A sketch of the Simple K-Means page clustering algorithm

```
BisectingKMeans(Number of Clusters $k$, Input Vectors $\mathcal{D}$, Iterations $I$)
    Define a clustering $\mathcal{C} = \{C_1\}$
    foreach input vector $d_j \in \mathcal{D}$
        Assign $d_j$ to $C_1$
    for $i = 1$ to $k - 1$
        Select a cluster $C_i \in \mathcal{C}$
        Let $\mathcal{D}_i$ denote the set of page vectors in $C_i$
        Define a set of candidate clusterings $Candidate = \{Candidate_1, ..., Candidate_I\}$
        for $j = 1$ to $I$
            $Candidate_j \leftarrow$ SimpleKMeans(2, $\mathcal{D}_{C_i}$)
        $\hat{C} \leftarrow$ BestClustering($Candidate$)
        $\mathcal{C} \leftarrow \{\mathcal{C} \cup \hat{C} \setminus C_i\}$
    return $\mathcal{C}$
```

Fig. 4. Tag Tree Signature Bisecting K-Means Page Clustering Algorithm

based on tag-tree signatures is provided in Figure 3.

Bisecting K-Means: Bisecting K-Means is a divisive variant of Simple K-Means. Rather than initially generating k cluster centers, this algorithm begins with a single cluster. It then divides the single cluster into two, where the two new clusters are found by the application of Simple K-Means. The algorithm repeatedly runs Simple K-Means for $k = 2$ for a specified number of iterations. The best cluster division is selected and the algorithm continues. Clusters continue to divide until there are exactly k clusters. This has the advantage over Simple K-Means of generating a hierarchy of clusters. A sketch of the Bisecting K-Means page clustering algorithm based on tag-tree signatures is provided in Figure 4. At each step we choose the largest cluster as the candidate cluster to be divided. We evaluate the quality of the cluster division using an internal similarity metric that is discussed below.

Improved Page Clustering

In Thor, we improve the quality of page clustering by relying on a number of clustering refinements. A sketch of the enhanced version of our page clustering algorithm is given in Figure 5. The algorithm may be tuned for either the Simple K-Means or Bisecting K-Means algorithm.

First, we attempt to optimize the choice of the number of clusters k, since the quality of both the Simple K-Means and Bisecting K-Means clustering algorithms can be greatly affected by this parameter choice. Selecting the optimal number of clusters is an open research problem with no approach guaranteed to select the optimal k. In general, most approaches select a reasonable value for k based on domain knowledge as a starting point, then update the value of k by learning from the clustering runs [12]. In our context, we run the K-Means algorithm repeatedly for k ranging from 1 to M. On each iteration we start with k randomly selected cluster centers and calculate a *quality* metric to evaluate the clustering produced by this iteration.

Given a set of n pages, the quality of clustering these n pages into k clusters can be measured by the

```
Enhanced K-Means Page Clustering Algorithm
    Let $P$ denote the set of $n$ pages sampled from Probing Phase
    Let $T$ be the set of $n$ page vectors corresponding to the $n$ pages
    Let $C_{best}$ denote the best clustering
    $C_{best} = \emptyset$
    create a TFIDF-tag signature $t$ for each page $p \in P$
    insert $t$ into $T$
    for $k = 1$ to $M$
        for $i = 1$ to $N$
            $C = SimpleKMeans(k, T)$ or $BisectingKMeans(k, T, I)$
            calculate $Similarity(C)$
            if $Similarity(C) \geq Similarity(C_{best})$, then $C_{best} \leftarrow C$
    rank the clusters in $C_{best}$
    pass the top-m clusters to the next phase ($m \leq k$)
```

Fig. 5. Thor Enhanced K-Means Clustering Algorithm

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 has been shown [21], [26] 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 quality of the entire clustering $\mathcal{C}$ can be computed by a weighted sum of the similarities of all component clusters: $Quality(\mathcal{C}) = \sum_{i=1}^{k} \frac{n_i}{n} Similarity(Cluster_i)$. We say a clustering has higher quality if its internal similarity is higher. We use the internal similarity 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. One of the advantages of the Thor approach is the selection of a k that is specific to the Deep Web source under analysis.

Second, for a given choice of k, we search a larger portion of the solution space by repeatedly running the clustering algorithm. Each iteration of the K-Means algorithm is guaranteed to converge [10], though the convergence may be to a local maxima. Hence, for each choice of k, we additionally run the K-Means algorithm N times; for each iteration, we choose a random starting set of cluster centers. Finally, we choose the best clustering for the given set of n pages on the iteration that yields the clusters with highest quality.

D. Recommending Page Clusters

The final step of the second phase is to filter out clusters that are unlikely to contain QA-Pagelets (like error and exception pages), and to recommend clusters that are likely to contain QA-Pagelets. Clusters are ranked according to their likelihood to contain QA-Pagelets and only the top-ranked clusters are passed along to the third phase. Several ranking criteria could be used based on the tag-tree characteristics defined in Section III. We briefly discuss three criteria that are considered by Thor for ranking page clusters:

Average Distinct Terms: 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 may be 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 poorly for some other pages. Our initial experiments over a wide range of sources show that a simple linear combination of the three ranking criteria works quite well, though it may be appropriate to revise the page ranking criteria based on domain-specific knowledge.

E. Incremental Evaluation

In our discussion above, the page clustering takes a set of sample pages and clusters them in a batch process. The Thor algorithm can also be applied incrementally to handle online queries of a Deep Web source. After an initial collection of sample pages and page cluster generation, we will have constructed cluster centers and ranked the clusters. As new online queries are issued to the Deep Web source, the resulting pages can be compared to the previously computed cluster centers and assigned to the most similar cluster. As additional new pages are incrementally assigned, the page clustering can be re-run to re-calculate cluster centers. The page clusters (both in the batch and incremental case) may then be further processed according to the third and fourth phases to be described next.

VI. QA-Pagelet Identification Phase

In the third phase, Thor examines pages from top-ranked page clusters at a subtree level to identify the smallest subtrees that contain the QA-Pagelets. This phase takes as input a single cluster of n_c pages, and outputs a list of QA-Pagelets. The main challenge of the third 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 III, each QA-Pagelet in a page is a subtree of the corresponding tag tree of the page. Thor relies on three main steps to identify QA-Pagelets: (1) single-page subtree filtering, (2) cross-page subtree filtering, and (3) selecting the subtrees containing QA-Pagelets. The goal of the single-page and cross-page filtering steps is to identify subtrees from each page that are candidate QA-Pagelets and remove those subtrees that are unlikely to be QA-Pagelets. The QA-Pagelet selection algorithm then ranks the resulting subtrees and recommends the QA-Pagelet for each page.

A. Single-Page Subtree Filtering

Single-page filtering 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 filtering 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. Formally, let $candidateSubtrees(p)$ be the set of candidate subtrees of page p produced by the single-page analysis step. The candidate subtree set can be defined as follows: $CandidateSubtrees(p) = \{u \in V \mid subtreeContent(u) \neq \emptyset$, $\forall v \in V$, $v \neq u$ and $v \Longrightarrow^{*} u$, $subtreeContent(u) \neq subtreeContent(v)$, $\forall w \neq u \in V, u \Longrightarrow^{*} w, fanout(w) > 1\}$.

B. Cross-Page Subtree Filtering

Cross-page filtering groups the candidate subtrees that correspond to the same type of content region into one subtree cluster and produces 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 filtering 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 filtering. 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: the path (P_j) to the root of the subtree, the fanout (F_j) of the subtree's root, the depth (D_j) of the subtree's root, and 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 ($1 \leq k \leq |CandidateSubtrees(p_r)|$) of common subtrees. Each common subtree set is composed of a single subtree from each page, and a subtree labeled j from page l is denoted as $subtree_j^l$: $CommonSubtreeSet_i = \{subtree_{i,1}^1, ...subtree_{i,n_c}^{n_c}\}$

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 [3] 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 show that this distance metric provides an effective mechanism for identifying common subtrees.

Step 2: Filtering the common subtree sets

This step ranks the k common subtree sets by the likelihood that each contains QA-Pagelets and filters out low-ranked subtree sets. 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 V-C. 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 [18]. 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: $subtree_{ij} = \Big\{(term_1, w_{i1}), (term_2, w_{i2}), \cdots, (term_{N_j}, w_{iN_j})\Big\}$.

The weight w_{iq} for subtree j is defined using the TFIDF weight function $w_{iq} = \log(tf_{iq} + 1) \cdot \log\left(\frac{n_j+1}{n_{qj}}\right)$, 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 as $IntraSubtreeSetSim_i = \sum_{j=1}^{n} \sum_{l \neq j}^{n} sim(subtree_j, subtree_l)$, where $1 \leq i \leq k$.

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.

C. QA-Pagelet Selection

After the single and cross-page filtering steps, 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 sources).

```
QA-Pagelet Selection Algorithm
    Choose $u \in DCS(p)$ s.t. $count_{DCS}(u)$ is maximized.
    repeat
        $\lambda(u) = \{v | v \in DCS(p)$ and $hops(u, v) = 1\}$.
        Select $u' \in \lambda(u)$, s.t. $count_{DCS}(u')$ is maximized.
    until $count_{DCS}(u) - count_{DCS}(u') < \delta$
    return $u'$
```

Fig. 6. QA-Pagelet Selection Algorithm

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 subtrees – say, the subtree corresponding to the entire page.

For each page p, we denote the remaining top-ranked subtrees (i.e. those subtrees that have passed both the single and cross-page filtering steps) as $DynamicContentSubtrees(p)$ or simply $DCS(p)$. Let $count_{DCS}(u)$ = the number of subtrees in $DCS(p)$ that are contained in subtree u, $u \in DCS(p)$. Let $hops_{DCS}(u, v)$ = the number of subtrees in $DCS(p)$ that are rooted along the path below u up to and including v, if v is reachable from u; 0, otherwise. The $hops(u, v)$ function is 1 if there are no other subtrees in $DCS(p)$ rooted along the path between u and v.

The QA-Pagelet selection algorithm in Figure 6 begins by finding the subtree that contains the most other dynamic content subtrees, since we expect the QA-Pagelet to also contain many subregions with dynamic content. Considering all the subtrees that are one "hop" deeper in the candidate subtree, it then chooses the subtree that contains the most dynamic content subtrees. This process continues until the stopping condition is met. By varying the parameter δ, we may control the relative depth of the selected QA-Pagelet. In our initial experiments, we find good results with $\delta = 2$. We anticipate refining this parameter in the future based on source-specific information to improve the accuracy of Thor.

At the end of third phase, we have a list of QA-Pagelets. The current selection algorithm supports the identification of a single subtree as the QA-Pagelet for a given page. Depending on the application scenario, it may be advantageous to relax this condition. For example, in cases in which the interesting dynamic content spans several subtrees, Thor could recommend the top-k candidate QA-Pagelets for further analysis. Similarly, 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 final phase to partition the QA-Objects.

VII. Experimental Setting

In this section, we discuss the data and metrics used to evaluate the quality of QA-Pagelet extraction. The software underlying the Thor architecture is written in Java 1.4. All experiments were run on a dual-processor Sun Ultra-Sparc 733MHz with 8GB RAM. Pages took on average 1.2 seconds to parse.

A. Data

To evaluate Thor, we rely on two datasets: (1) a collection of over 5000 pages from 50 diverse real-world web data sources; and (2) a synthetic data set of over 5 million pages designed to emulate the diversity and scope of real-world Deep Web sources.

Web data: Using a breadth first crawl of the Web starting at several randomly selected College of Computing personal homepages, we identified over 3,000 unique search forms. We randomly selected 50 of the 3,000 search forms. The web sites chosen represent a wide range of sources, including eCommerce sites, general and specialized search engines, corporate websites, medical and legal resources, and several others. 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. In Table I, we list more detailed information about the 50 web sites, including relevant information on the average page size, number of terms per page, and tags per page.

Synthetic data: With the web data we can evaluate the Thor algorithm, but we are also interested in assessing the scalability of the algorithm to ensure that the Thor approach is valid over increasingly large datasets. Since evaluating Thor's two phases requires the burdensome task of hand-labeling each page and QA-Pagelet, we rely on a synthetic data set designed to closely match the real world data to evaluate Thor's scalability. 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

TABLE I

WEB DATASET SUMMARY: 110 PAGES PER SITE

URL	Description	Avg Size (KB)	Avg Total Tags	Avg Unique Tags	Avg Total Content Tokens	Avg Unique Content Tokens
1. abcnews.go.com/	News	41.8	813.9	27.0	526.3	213.7
2. alberta.indymedia.org/	Independent news	22.3	614.5	23.0	503.5	326.2
3. c2.com/cgi/wiki/	Wiki community resource	3.0	82.4	8.7	48.8	46.3
4. dmoz.org/	Web directory	10.6	185.6	21.9	454.9	205.6
5. monogo.ru/	Russian portal	8.0	127.7	25.5	64.7	39.3
6. www.aidscience.org/	Scientific research	25.3	298.2	17.2	115.7	69.2
7. www.ameinfo.com/	Middle East news	24.8	464.9	28.3	482.9	240.8
8. www.ama-assn.org/	Physician resources	50.9	883.0	27.3	637.9	276.6
9. www.anime100.com/	Anime portal	46.4	772.3	20.5	2719.4	243.5
10. www.aolsearch.com/	Search engine	22.1	240.6	23.1	419.4	215.4
11. www.atomz.com/	Corporate Web site	6.7	119.8	20.6	215.5	107.7
12. www.autoweb.com/	Car resources	33.3	293.9	21.0	125.7	84.8
13. www.bn.com/	Bookseller	72.3	1098.7	22.3	430.0	171.2
14. www.biblegateway.com/	Bible search engine	30.1	308.1	20.6	242.3	127.3
15. www.bookpool.com/	Discount bookseller	9.5	210.7	20.2	179.4	81.4
16. www.booksite.com/	Publishing industry developer	12.6	283.3	17.5	229.4	109.8
17. www.bostonherald.com/	News	32.9	484.0	21.8	509.3	258.4
18. www.careerbuilder.com/	Job search	29.1	387.0	23.0	170.7	98.9
19. www.cisco.com/	Corporate Web Site	33.5	591.4	21.0	430.0	128.6
20. www.clinicaltrials.gov/	NIH clinical research resource	8.5	156.8	22.1	154.2	81.9
21. www.dell.com/	Corporate Web site	26.2	242.1	11.7	73.7	30.8
22. www.ebay.com/	Online marketplace	75.4	1461.5	27.6	516.9	241.4
23. www.enhydra.org/	ebusiness app server	11.9	197.6	22.5	91.4	65.4
24. www.esa.int/	Space news and portal	15.0	329.9	21.8	266.4	113.9
25. www.findlaw.com/	Attorneys and legal resources	32.9	655.9	25.8	1240.8	463.1
26. www.galaxy.com/	Search engine and directory	21.1	347.8	26.4	507.5	249.4
27. www.google.com/	Search engine	16.6	327.6	22.5	432.6	202.1
28. www.ibm.com/	Corporate Web site	19.1	343.7	23.4	347.0	144.6
29. www.ign.com/	Games and movies portal	59.9	1154.4	25.0	1197.1	217.0
30. www.itn.co.uk/	British news	8.0	126.5	17.3	163.0	92.6
31. www.latimes.com/	News	42.0	749.8	25.7	385.1	214.7
32. www.mamma.com/	Search engine	31.2	485.3	23.1	607.3	280.4
33. www.mp3.com/	Music search engine	25.8	365.6	23.4	208.3	120.3
34. www.msn.com/	Search engine and portal	29.6	530.6	24.0	587.0	287.5
35. www.omniseek.com/	Metasearch engine	14.6	330.8	25.9	518.4	280.8
36. www.phatoz.com/	Portal	7.7	178.3	23.4	210.1	122.5
37. www.powells.com/	Bookseller	57.4	1071.8	22.2	919.0	342.7
38. www.questfinder.com/	Search engine	15.9	322.7	19.9	433.2	205.8
39. www.reuters.com/	Business and financial news	37.1	567.1	20.0	339.0	179.4
40. www.scrubtheweb.com/	Search engine	20.8	530.8	19.8	832.2	313.4
41. www.searchking.com/	Search engine	12.1	246.3	29.0	295.4	174.4
42. www.searchport/	Search engine	7.6	169.5	25.0	189.5	109.5
43. www.techtv.com/	Technology news	21.3	259.8	21.4	314.9	168.7
44. www.terra.com.br/	Brazilian portal	29.4	462.4	28.9	724.0	347.7
45. www.thebookplace.com/	Bookseller	35.4	603.6	21.9	329.8	154.0
46. www.themoscowtimes.com/	Russian news	24.7	642.6	19.8	567.7	269.8
47. www.thesportsauthority.com/	Sporting goods shopping site	85.6	1101.3	22.3	430.6	171.4
48. www.usatoday.com/	News	19.1	319.5	22.3	537.1	229.0
49. www.wired.com/	Technology news	12.8	157.6	23.6	352.5	216.8
50. www.zope.org/	Zope community resource	5.7	117.3	18.3	81.6	62.7

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).

B. Metrics

We evaluate the Thor page clustering approach against several alternatives by measuring the time to cluster and the entropy of the clustering result. Entropy is a well-known measure of the randomness or disorder in a system [20]. For a particular clustering, entropy measures the quality of assignments of pages to clusters. 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 $\mid$ 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 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 described in Section V.

To evaluate the quality of QA-Pagelet identification, we rely on measures of precision and recall. Let $x =$ Number of QA-Pagelets Correctly Identified, $y =$ Number of Subtrees Identified as QA-Pagelets, and $z =$Total Number of QA-Pagelets in the Set of Pages. Then $precision = x/y$ and $recall = x/z$.

VIII. Experimental Results

In this section, we report a number of experimental results to evaluate the overall effectiveness of the Thor approach.

A. Clustering Deep Web Pages

We first examine the quality of Thor's Deep Web page clustering to evaluate the effectiveness of both the similarity metric 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 the Thor approach against several alternatives, 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 V. 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. Since a URL-based centroid cannot be easily defined, we assigned pages to clusters based on the average similarity of the page to each page in the candidate cluster. For the size-based approach, we described each page by its size in bytes, measured the distance between two pages by the difference in bytes, and calculated each cluster centroid as the average size in bytes of the cluster members. As a baseline, we also considered an approach that randomly assigned pages to clusters.

We selected n pages from each of the 50 collections and ran each set through our enhanced K-Means clustering algorithm using the cosine similarity function. We repeated this process 10 times

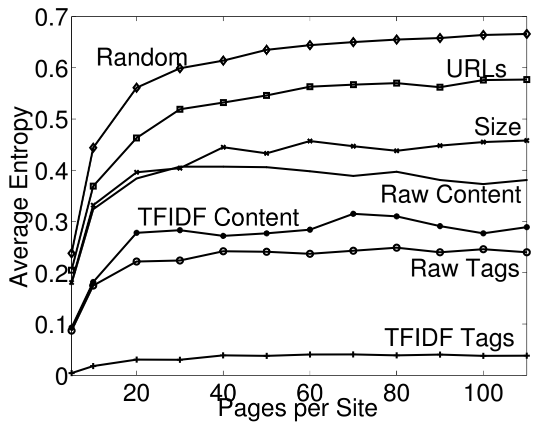

Fig. 7. Entropy (a)

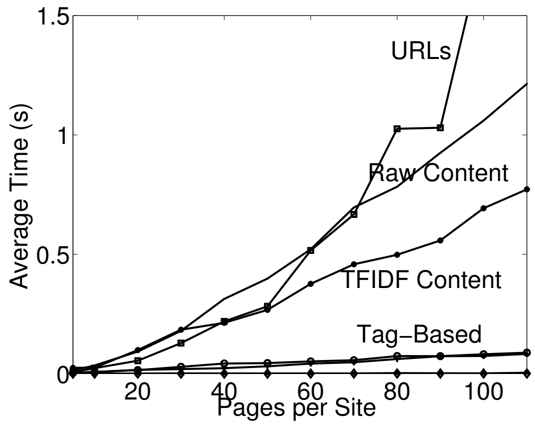

Fig. 8. Time (a)

to generate an average entropy and clustering time for collections ranging from 5 to 110 pages. In Figure 7, 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). Internally, the clustering algorithm ran 10 times, at each iteration assessing the overall similarity of the clustering. The clustering with the highest similarity was then recommended as the final clustering for evaluation.

The 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 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). The TFIDF approach also results in more consistently good clusters across all 50 Deep Web sources, with a standard deviation signicantly less than the other techniques. 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, we discovered that many Deep Web sites do use some tags in some page types but not others, resulting in the better performance of the TFIDF-weighted approach. Hence, 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.

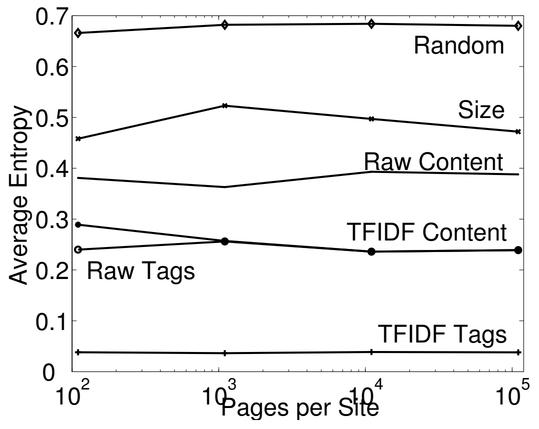

Fig. 9. Entropy (b)

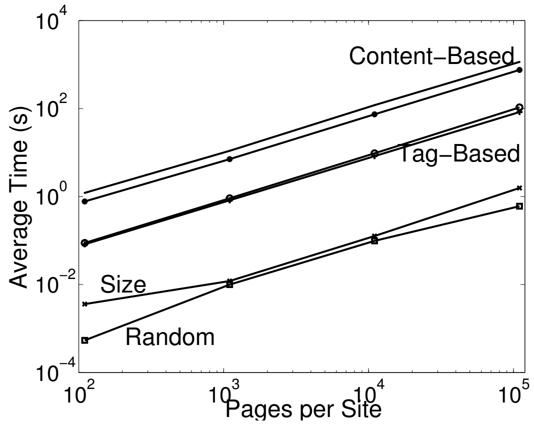

Fig. 10. Time (b)

In Figure 8, 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 [17]. 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 9, 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 10, 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.

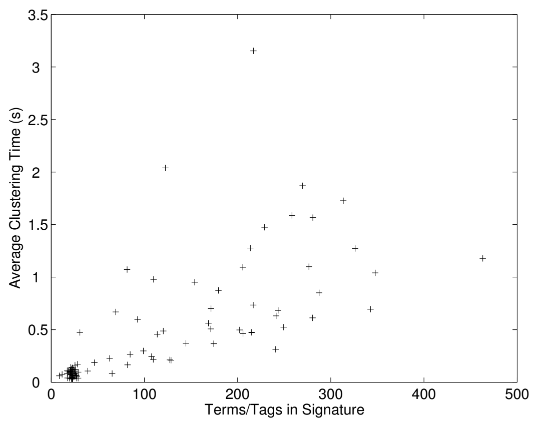

Fig. 11. Time versus Signature Size

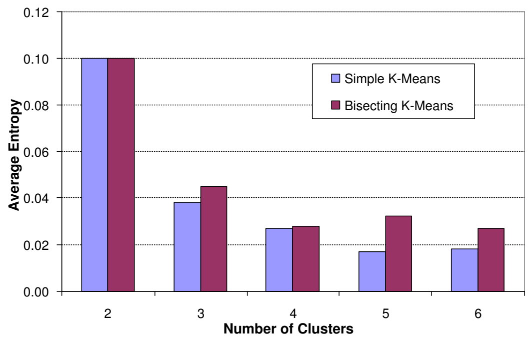

Fig. 12. Comparison of Clustering Algorithms

In Figure 11, we show the average clustering time as a function of the number of unique terms in the signature of the page for both the TFIDF-tag approach and the TFIDF-content approach. This experiment was conducted over each of the 50 collections. The dense group in the bottom-lefthand corner corresponds to the TFIDF-tag approach, which further validates the time advantage of the tag-based approach over the content-based approach. We also note the approximately linear growth of clustering time as a function of the total number of distinct tags or terms in the signature.

In Figure 12, we compare the effectiveness of the Simple K-Means clustering algorithm versus the Bisecting K-Means version over the original 5,500 page dataset using the TFIDF-tag approach and the cosine similarity function for several cluster number settings. As we would expect, both report better results for higher numbers of clusters. Interestingly, the Simple K-Means algorithm results in slightly lower entropy clusters. We attribute this advantage to the lack of a clear hierarchical nature to the underlying data. We would expect Bisecting K-Means to be preferred in cases in which such a hierarchical structure is more pronounced.

In Table II, we compare the effect of the underlying similarity metric on the performance of page clustering. We show the average entropy and standard deviation over the original 5,500 page dataset. In the presence of the TFIDF-tag-weighting, both metrics perform approximately the same (i.e. 0.038 versus 0.037). But in the un-normalized cases, the cosine significantly outperforms the Euclidean distance in cluster quality. We attribute this improvement to the cosine's implicit vector normalization; in contrast, the Euclidean distance may overstate the difference between vectors of different lengths.

To further illustrate the effectiveness of the TFIDF-weighted tag signature approach, we list in Table III the dominant tags in the centroid for the "Normal" page cluster and the "No Results" page

TABLE II

COSINE VERSUS EUCLIDEAN: ENTROPY

	Average	StDev
Cosine (TFIDF Tags)	0.038	0.074
Euclidean (TFIDF Tags)	0.037	0.073
Cosine (Raw Tags)	0.117	0.170
Euclidean (Raw Tags)	0.279	0.257

TABLE III

CENTROIDS FOR CLINICALTRIALS

	Normal	No Results
<i>	0.97	—
<hr>	0.11	0.08
<td>	0.05	0.03
<font>	0.05	0.03
<a>	0.04	0.03
<textarea>	—	0.42
<div>	—	0.61
<p>	—	0.66
vector length	1.00	1.00

TABLE IV

RAW CENTROIDS FOR CLINICALTRIALS

	Normal	No Results
<td>	0.63	0.49
<font>	0.57	0.60
<a>	0.32	0.46
<input>	0.23	0.05
<tr>	0.23	0.26
<th>	0.11	0.26
<b>	0.10	0.05
<table>	0.09	0.10
vector length	1.00	1.00

cluster for one of the Deep Web data sources (ClinicalTrials.com) we have tested. Our page clustering successfully segmented the two types of pages. On a closer examination of these tag signatures, we can see why the clustering was so successful. The tag signatures differ in both the presence of certain tags and in the relative contribution of each tag to the overall signature. The TFIDF-weighting scheme has emphasized the presence of the tag `<i>` in the "Normal" page cluster versus the highly-weighted `<div>`, and `<p>` tags in the "No Results" cluster.

In contrast, Table IV shows the dominant centroid terms for the same collection of pages, but using the *raw* normalized tag signatures for each page instead of the TFIDF weighted tag signatures. Interesting to note is that the distinguishing tag terms in the successful TFIDF case do not significantly contribute to the raw tag signature. And for the top three tags that are significant, the relative contribution of each is quite similar for both classes of pages (e.g. `<font>` is 0.57 in one case, 0.60 in the other). This illustrates how our TFIDF-weighting scheme is effective at focusing on the critical differentiating tags in each class of pages.

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. If we set the number of clusters greater than the number of actual clusters, the clustering algorithm will generate more refined clusters. This is not a problem in our context, since QA-Pagelet identification is dependent only on

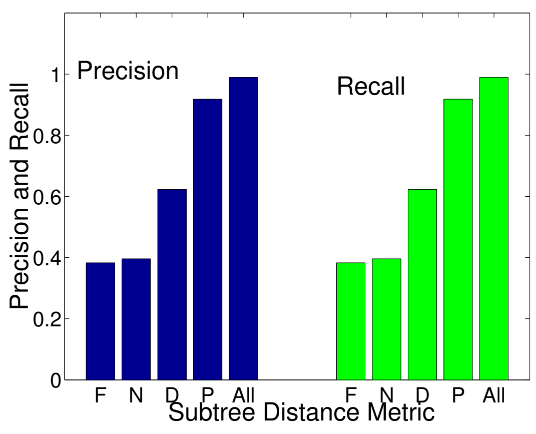

Fig. 13. Distance Comparison

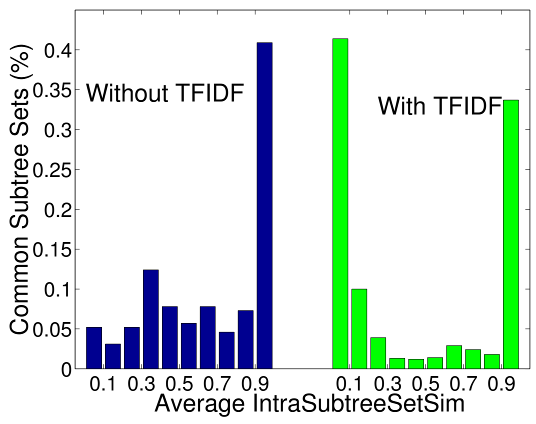

Fig. 14. Subtree Similarity

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.

B. Identifying QA-Pagelets

We next evaluate the quality of the Thor QA-Pagelet identification by measuring precision and recall statistics. To consider this phase in isolation from the page clustering phase, we considered as input only those pages from each Deep Web source that had been pre-labeled as containing QA-Pagelets.

We considered several variations of the subtree distance function in an effort to validate the Thor approach. These variations include 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 13, we report the average 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%. The combined distance metric performs better in nearly all cases (49 of 50) versus the next-best metric, achieving a standard deviation precision significantly less for the combined metric (2%) versus the other metrics (which range from 13% to 40%). 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 14 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 14 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.

C. *Evaluating Overall Performance*

We conclude by reporting the overall performance of the Thor QA-Pagelet extraction. In Figure 15, 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, performing better in nearly all cases (47 of 50) versus the next-best metric, and achieving a standard deviation precision significantly less for the combined metric (8%) versus the other metrics (which range from 28% to 35%). The quality of clusters generated in the first phase doubly impacts the overall performance. First, if a normal results page is mis-clustered 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 third phase will worsen QA-Pagelet identification by hampering the cross-page analysis. Hence, it is of critical importance to generate quality clusters.

Finally, in Figure 16 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

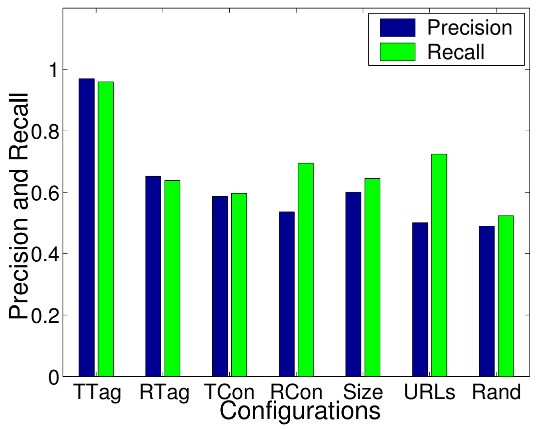

Fig. 15. Overall Precision and Recall

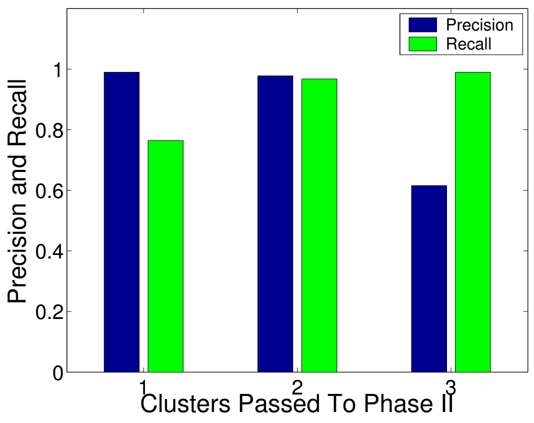

Fig. 16. Precision/Recall Trade-off

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. In general, we expect the optimal number to be domain specific.

IX. Conclusions

To support large scale data analysis over the Deep Web, we have introduced a fundamental data preparation approach for the robust sampling, identification, and extraction of the high-quality query-related content regions in Deep Web pages. We have introduced the concept of a QA-Pagelet and presented the Thor framework for efficiently extracting QA-Pagelets from the Deep Web in four phases. The first phase probes the Deep Web data sources to sample a set of pages rich in content. The second phase uses a novel page clustering algorithm to segment Deep Web pages 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 third phase, pages from top-ranked page clusters are examined at a subtree level to locate and rank the query-related content regions of the pages (the *QA-Pagelets*). The final phase uses a set of of partitioning algorithms to separate and locate itemized objects in each QA-Pagelet. We show that Thor performs well over millions of Deep Web pages and over a wide range of sources, including eCommerce sites, search engines, corporate websites, medical and legal resources, and several others.

Our experiments also show that the proposed page clustering algorithm achieves low-entropy clusters, and the subtree filtering algorithm identifies QA-Pagelets with excellent precision and recall.

X. Author Biographies

James Caverlee received the M.S. degree in Engineering-Economic Systems & Operations Research in 2000 and the M.S. degree in Computer Science in 2001, both from Stanford University. He is currently a Ph.D. student in the College of Computing at the Georgia Institute of Technology. His research interests are focused on the Deep Web, Web Services, and workflow systems.

Ling Liu is currently an associate professor at the College of Computing at Georgia Tech. There, she directs the research programs in Distributed Data Intensive Systems, examining research issues and technical challenges in building large scale distributed computing systems that can grow without limits. Dr. Liu and the DiSL research group have been working on various aspects of distributed data intensive systems, ranging from decentralized overlay networks, examplified by peer to peer computing and grid computing, to mobile computing systems and location based services, sensor network systems, and enterprise computing technology. She has published over 100 international journal and conference articles. Her research group has produced a number of software systems that are either open sources or directly accessible online, among which the most popular ones are WebCQ and XWRAPElite. She and her students have received a best paper award from IEEE ICDCS 2003 for their work on PeerCQ and a best paper award from International Conference of World Wide Web (2004) for their work on Caching Dynamic Web Content. Most of Dr. Liu's current research projects are sponsored by NSF, DoE, DARPA, IBM, and HP. She was a recipient of IBM Faculty Award (2003) and a participant of the Yamacraw program, State of Georgia.

References

[1] B. Adelberg. NoDoSE–a tool for semi-automatically extracting structured and semistructured data from text documents. In *SIGMOD '98*.

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

[3] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

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

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

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

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

[8] J. Caverlee, L. Liu, and D. Buttler. Probe, cluster, and discover: Focused extraction of QA-Pagelets from the Deep Web. In *ICDE '04*.

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

[10] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1/2):143–175, 2001.

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

[12] M. Halkidi, Y. Batistakis, and M. Vazirigiannis. Clustering validity checking methods: Part ii. *SIGMOD Record*, 31(3):19–27, 2002.

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

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

[15] L. Liu, C. Pu, and W. Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE '00*.

[16] Z. Liu, C. Luo, J. Cho, and W. Chu. A probabilistic approach to metasearching with adaptive probing. In *ICDE '04*.

[17] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *WebDB '02*.

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

[19] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB '01*.

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

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

[22] J. Wang, J.-R. Wen, F. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB '04*.

[23] W. Wu, C. T. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04*.

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

[25] Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *SIGMOD '04*.

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