Hierarchical Topic Segmentation of Websites

Ravi Kumar
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
ravikumar @ yahoo-inc.com

Kunal Punera*
Dept. of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712
kunal @ ece.utexas.edu

Andrew Tomkins
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
atomkins @ yahoo-inc.com

ABSTRACT

In this paper, we consider the problem of identifying and segmenting topically cohesive regions in the URL tree of a large website. Each page of the website is assumed to have a topic label or a distribution on topic labels generated using a standard classifier. We develop a set of cost measures characterizing the benefit accrued by introducing a segmentation of the site based on the topic labels. We propose a general framework to use these measures for describing the quality of a segmentation; we also provide an efficient algorithm to find the best segmentation in this framework. Extensive experiments on human-labeled data confirm the soundness of our framework and suggest that a judicious choice of cost measures allows the algorithm to perform surprisingly accurate topical segmentations.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Measurements

Keywords

Website Hierarchy, Website Segmentation, Tree Partitioning, Classification, Facility Location, Gain Ratio, KL-distance

1. INTRODUCTION

As the major established search engines vie for supremacy, and new entrants explore a range of technologies to attract users, we see researchers and practitioners alike seeking novel analytical approaches to improve the search experience. One promising family of approaches that is generating significant interest is analysis at the level of websites, rather than individual webpages. There are a variety of techniques for exploiting site-level information. These include detecting multiple possibly-duplicated pages from the same site [2, 4], determining entry points [8], identifying spam and porn sites [18], detecting site-level mirrors [4], extracting site-wide templates [16] and structures [25], and visualizing content at the site level [15].

In this paper we consider *site segmentation*, a particular form of site-level analysis in which a website must be segmented into one or more largely uniform regions. The segmentation may be performed based on the topics discussed in each region, or based on the look and feel, or based on the authorship, or other factors. We focus specifically on *topical segmentation*, i.e., segmenting a site into pieces that are largely uniform in the topics they discuss. Such a topical segmentation offers many potential advantages:

(i) Various algorithms that are currently applied to websites could more naturally be applied to topically-focused segments.

(ii) Websearch already incorporates special treatment for pages that are known to possess a given topic—for instance, many engines provide a link to the topic in a large directory such as the Yahoo! Directory, Wikipedia, or the Open Directory Project. These approaches can naturally be extended when several pages from a search result list lie within a topically-focused segment.

(iii) The resultant segments provide a simple and concise site-level summary to help users who wish to understand the overall content and focus of a particular website.

(iv) A host such as an ISP may contain many individual websites, and a topical segmentation is a useful input to help tease out the appropriate granularity of a site.

(v) Website classification is a problem that has been addressed using primarily manual methods since the early days of the web, in part because sites typically do not contain a single uniform class. Segmentation is an important starting point for this larger problem.

Site segmentation may be viewed from two distinct perspectives. First, it may be viewed as a constrained clustering problem in which the allowable segments represent constraints on the possible clusters that the algorithm may return. At the same time, site segmentation may be viewed as an extended form of site-level classification in which the algorithm may choose to classify either the entire site, or various sub-sites. The measure we propose for the quality of a segmentation is much simpler than standard measures from machine learning. As a result, while the problem may be viewed as a constrained version of the NP-hard clustering

*This work was done while the author was at Yahoo! Research.

problem, or an extended version of classification that incorporates a search for the appropriate objects to classify, the simple measure of segmentation quality, combined with the class of allowable segmentations, will allow us to provide an algorithm to return the *optimal* segmentation in polynomial time. To achieve this bound, we employ a dynamic programming algorithm that is quite different from traditional algorithms for either clustering or classification.

One could consider many different classes of allowable segmentations of a website, for example based on the hierarchical structure of the site, or based on clusters in the intra-site link graph, or based on regions of the site that display some commonality of presentation template, and so forth. We will focus specifically on segmentations that respect the hierarchical structure of a website, for two reasons. First, we believe that of the many possible approaches to segmenting websites, hierarchy is the most natural starting point. Site authors often think in terms of a site being made up of several sub-sites, each of which may contain sub-structure of its own; and the layout of pages on a website often follows a "folder" structure inducing a natural hierarchy. And second, in many applications an individual segment must be returned to the user in some succinct manner. Rather than simply returning a long list of URLs located at various positions within the site, it is desirable to return instead a pointer to a particular sub-site.

In general, the hierarchical structure of a website may be derived from the tree induced by the URL structure of the site, or mined from the intra-site links or the page content of the site. Our algorithm makes use of whatever hierarchical information is available about a site to constrain the possible segmentations. We show that 85-90% of sites exhibit a non-trivial form of hierarchy based on the URL tree that can exploited by our algorithm for segmentation. The remaining fraction of sites might have a latent hierarchical structure that could be mined by further analysis of intra-site links or content, but that is beyond the scope of this paper.

Thus, our paper is on *hierarchical topic segmentation* (HTS): the segmentation of websites into topically-cohesive regions that respect the hierarchical structure of the site.

Formulation. Consider a tree whose leaves have been assigned a class label or a distribution on class labels, perhaps by a standard page-level classifier. A distribution is induced on an internal node of the tree by averaging the distributions of all leaves subtended by that internal node. These distributions, along with a hierarchical arrangement of all the pages in the site, are provided to the HTS algorithm. The algorithm must return a set of segmentation points that optimally partition the site. The objective function for the segmentation is a combination of two competing costs: the cost of choosing the segmentation points (the nodes) themselves and the cost of assigning the leaves to the closest chosen nodes. Intuitively, the *node selection* cost models the requirements for a node to serve as a segmentation point, while the *cohesiveness* cost models how the selection of a node as a segmentation point improves the representation of the content within the subtree rooted at it. For example, in a particular instance of the problem, the node selection cost can capture the requirement that the segments be distinct from one another and the cohesiveness cost can capture the requirement that the segments be pure. The underlying tree structure enables us to obtain an efficient polynomial-time algorithm to solve the HTS problem.

To complete the overview of HTS, we provide a brief discussion of the difference between segmentation and classification. The general website classification problem tries to assign topics to websites by employing features that are broad and varied. A few example features for this broader problem include the topic of each page, the internal hyperlinks on the site, the commonly link-to entry points to the site, with their anchor-text, the general external link structure, the directory structure of the site, the link and content templates present on the site, the description, title, and h1-6 tags on key pages on the site, and so forth. The final classes in a website classification problem may be distinct from the classes employed at the page level. HTS, on the other hand, specifically focuses on aggregating the topic labels on webpages into subtrees according to the hierarchy of a site, in order to convey information such as, "This entire sub-site is about Sports." Thus, HTS attacks the problem of determining whether and how to split the site, but is only the beginning of a broader research problem of classifying websites using rich features. The broader problem is of great interest in both binary cases (is the site spam? is it porn?) and multi-class cases (to what topics should I assign this site?). We believe that a clean and elegant solution to the HTS problem is essential to fully address the more general site classification problem.

Summary of Contributions. We provide a rigorous formulation of HTS for websites that is general enough to capture many different hierarchical topic segmentation schemes. We show how to encode two natural requirements within our formulation: the segments themselves should be sufficiently 'distinct' from each other and the webpages in a segment should be reasonably 'pure' in topic. We also present a polynomial-time algorithm to solve the HTS problem optimally.

We conduct an extensive set of experiments to evaluate the performance of our algorithm with various natural cost measures on hand-labeled as well as semi-synthetic websites. We show that a judicious choice of the node selection cost and cohesiveness cost can vastly improve the performance of the algorithm.

Organization. Section 2 contains relevant work on hierarchical classification and segmentation. Section 3 presents the framework for the HTS problem. Section 4 contains algorithm for the HTS problem as well as definitions for the cohesiveness and node selection costs. Finally, the experimental results are presented and discussed in Section 5.

2. RELATED WORK

HTS operates upon a tree-structured object with a class distribution at each node. The goal is to segment the tree structure into connected components that are cohesive in terms of the class distributions. While HTS is not classification *per se*, we nonetheless turn to the machine learning community for related work on such objects. We have not been able to find references to previous direct study of such objects, but there are a number of related areas.

Site Level Classification. There has been some prior work on treating websites or groups of webpages as the basic unit of analysis [22, 30, 31]. Kriegel et al. [21, 11] present various website classification schemes based on features extracted from the individual webpages. Some of their schemes consider topics at individual webpages but they use these as features for the site level classifier. They have no notion of segmenting a website into sub-parts, and they learn models for websites as a whole. More recent work by Tian et al. [33] uses "hidden Markov trees" to model both the site directory trees as well as the DOM trees of the webpages. They then employ a two-phase system through a fine-to-coarse recursion to classify the site. Sun et al. [28] propose a technique to partition websites into "Web Units", which are collections of webpages. These fragments are created using heuristics based on intra-site linkages and the topical structure within the website is not considered.

Classification of Hierarchical Documents. There is a rich body of literature around classification of other tree-structured objects, including semi-structured documents in either HTML or, more commonly, XML.

Theobald et al. [32] discuss classification of XML documents, using features that derive from the tree structure of the XML document. However, these features are extracted from simple types of path relationships, and are then processed by a traditional classifier. They do not discuss the partitioning of the XML tree based on the classification.

Diligenti et al. [10] consider classification of semi-structured documents using a "hidden tree Markov model", in which each subtree is generated by a particular Markov model. Piwowarski et al. [23] and Denoyer and Gallinari [9] consider a similar model in which document trees are modeled using Bayesian networks.

Classification using Hierarchical Models. Other researchers have employed hierarchical models, not to the problem of classifying hierarchical objects, but in order to consider flat objects at different levels of granularity.

Hierarchical HMM, introduced by Fine, Singer, and Tishby [14], is a hierarchical generalization of the widely used hidden Markov model. Hierarchical HMMs can be useful for unsupervised learning and modeling of complex multi-scale structures that occur in language, handwriting, and speech; in particular, they were used to construct hierarchical models of natural English text.

Koller and Sahami [20] show a hierarchical generative model called tree-augmented naive Bayes. The goal is to capture local similarities in vocabulary within a document. Blei and Jordan [5] consider the problem of modeling annotated data and derive hierarchical probabilistic mixture models to describe such data.

Graph-based Approaches. Finally, there are many approaches to processing of objects as nodes of a graph, where edges denote relationship to other objects. Extensions to our model to include hyperlinks would make use of such techniques.

Chakrabarti et al. [6] present Hyperclass, a hyperlink-aware classifier operating upon documents in a graph that takes into account the classes of neighboring documents via a technique based on Markov Blankets. This approach is similar in spirit to our work, but their main goal was to classify documents, rather than partition the underlying graph. Agrawal et al. [1] consider partitioning a graph so as to maximize the dissimilarity between the two sides. They propose a maximum cut-version of the problem and argue that spectral methods work well on their instances.

Hierarchical Partitioning. Moving beyond the purview of machine learning, there are numerous approaches to partitioning hierarchical objects. Fagin et al. [12, 13] consider a

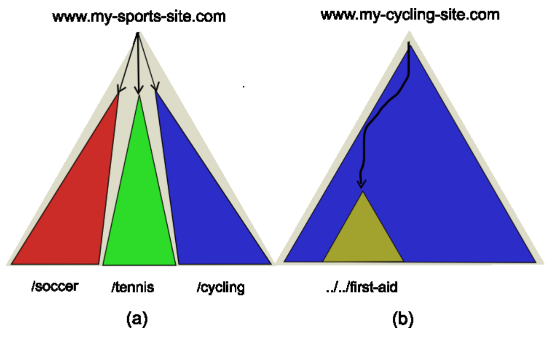

Figure 1: Two hypothetical sites with different topical organization along directories.

general notion of partitioning hierarchical structures based on a variety of different quality measures.

Our problem is closely related to the problem of facility location on trees. While the problem on general graphs is NP-hard [19], lot of work has been done to obtain fast exact algorithms for the facility location problem on trees with n nodes; the goal is to find k facilities. Tamir [29] obtained a dynamic programming algorithm that runs in time $O(kn^2)$; this was an improvement over the $O(k^2n^2)$ algorithm of Kariv and Hakimi [19] and $O(kn^3)$ algorithm of Hsu [17]. For some important special case distance functions, the best running time bound of $O(n \log n)$ is due to Shah and Farach-Colton [27].

3. FORMULATION

We begin this section with a brief discussion of HTS, and then present a general mathematical formulation.

3.1 Hierarchical Topic Segmentation

Consider the problem of describing the topical content of a website to a user. If the site is topically homogeneous we could provide the user with the URL of the site and a topic label representing the content. Our segmentation algorithms should do exactly this. However, most sites are not homogeneous, and in fact the organization of topics within directories can determine the best way to summarize site content for the user.

For instance, consider the two hypothetical websites shown in Figure 1. The site in panel (a) contains sub-sites on different topics, while the site in panel (b) contains a single topically coherent tree expect for a small directory deep in the site structure. In the first case we could describe the site using the top-level directories, such as `www.my-sports-site.com/tennis`, and for each such directory give its prevailing topic, such as Sports/Tennis. For panel (b) on the other hand, we could tell the user that the entire site (`www.my-cycling-site.com`) is about Sports/Cycling, except that a small piece at `www.my-cycling-site.com/.../first-aid/` is about Health/First-Aid. As this example shows, it is quite reasonable to describe a site using nested directories if this is the best explanation for the content.

In general, we wish to make optimal use of the user's attention and convey as much information about the site as possible using the fewest possible directories, i.e., internal nodes. Hence, each directory we call out to the user should provide significant additional information about the site.

This informal description of the problem is in terms of explaining the contents of a website to a user. The other application areas listed in Section 1 leverage the same framework, but make use of the final description in other ways. Generally, the goal is to return a concise segmentation of a website into topically coherent regions.

Here we note that while in this paper we restrict ourselves to segmentations that follow the directory (URL) tree, our approach can be applied to any hierarchical structure within a website. Indeed, websites with trivial URL based hierarchical structure, for instance dynamic pages with URLs of the form `http://mysite.com/show.php?productid=42`, are increasing in number, especially in the e-commerce domain. However, besides being the first step to study the segmentation problem, our restriction to URLs captures the vast majority $(85-90\%)$ of websites, and allows us to study how to make use of this key element of site structure. Our approach could be applied to the remaining websites by first mining their latent hierarchical structure by a deeper analysis of links, content, or URL [26], but that is beyond the scope of this paper.

3.2 Formal Definition

The natural approach to modeling a directory structure is by a rooted tree whose leaves are individual pages.[1] We assume that there is a page-level classifier that assigns class labels or a distribution over class labels to each page of the directory structure. This induces a distribution on the internal nodes of the tree as well, by uniformly combining the distribution of all descendant pages. Our notion of cohesiveness of a subtree will be based upon the agreement between each leaf with the distribution at its parent. We require a few definitions to make this notion formal.

Let T be a rooted tree with n leaves; let $\text{leaf}(T)$ denote the leaves of T and let $\text{root}(T)$ denote its root. Let Δ be the maximum degree of a node in T. Let L be the set of class labels. We assume that each leaf x in the tree T has a distribution p_x over L, generated by some page-level classifier. We will write $p_x(i)$ to denote the probability that leaf x has class label i. For an internal node u with leaves $\text{leaf}(u)$ in the subtree rooted at it we define the distribution of labels at u as follows:

$$p_u(i) = \frac{1}{|\text{leaf}(u)|} \sum_{x \in \text{leaf}(u)} p_x(i).$$

A subset S of the nodes of T is said to be a *segmentation* of T if, for each leaf x of T, there is at least one node $y \in S$, such that x is a leaf in the subtree rooted at y. For example, S is always a segmentation if $\text{root}(T) \in S$. Given a parameter k, the goal now is to find a segmentation of size at most k whose components are cohesive. For a leaf $x \in \text{leaf}(T)$ let $S_x \in S$ be the first element of S on the ordered path from x to $\text{root}(T)$. We will say that x *belongs* to S_x, and we will define a *cohesiveness* cost $d(x, S_x)$ that captures the cost of assigning x to S_x. Further, we will define a *node selection* cost $c(y, S)$ that gives the cost of adding y to S. The overall cost of a particular segmentation S is then

$$\beta \sum_{y \in S} c(y, S) + (1-\beta) \sum_{x \in \text{leaf}(T)} d(x, S_x), \qquad (1)$$

[1]If internal nodes also correspond to pages, we simply model them using the standard "index.html" convention.

where β is a constant controlling the relative importance of the node selection cost and the cohesiveness cost. Our algorithms then find the lowest-cost segmentation, given functions $c(\cdot)$ and $d(\cdot)$ representing the problem instance.

The formulation in (1) is reminiscent of the uncapacitated facility location (UFL) problem in combinatorial optimization. In UFL, we are given a graph (V, E), a parameter k, and each vertex v has a cost $c(v)$ and the goal is to choose $S \subseteq V$ with $|S| = k$ such that $\sum_{v \in S} c(v) + \sum_{u \in V} \min_{v \in S} d(u, v)$ is minimized. Here, d is the graph metric defined by E. In this most general version, UFL is NP-hard. In our case, G is only a tree and the distance function is more general and is not necessarily a metric.

4. SEGMENTATION ALGORITHM

Our algorithmic approach is based on a general dynamic program that optimizes the objective function of (1). This dynamic program works for any cohesiveness cost $d(\cdot)$ and node selection cost $c(\cdot)$. It runs in time $O(k^2nd)$. After describing the dynamic program, we then present a set of candidate cohesiveness costs and node selection costs. We compare these different approaches empirically in Section 5.

4.1 A Generic Algorithm

The idea behind the dynamic program is the following. Given a subtree whose root has δ children, the optimal way of adding at most k nodes from the subtree to the segmentation must follow one of two patterns. In the first pattern, we add the root of the subtree and then recurse on the children with a budget of $k-1$. In the second pattern, we do not include the root of the subtree, and instead recurse on the children with a budget of k. A naive way of implementing the recursion would result in segmenting k (or $k-1$) into δ pieces in all possible ways. This is expensive, if $\delta \gg 2$. To circumvent this, we show a simple transformation that will convert the tree to binary, without changing the optimum.

Construct a new tree from the original tree T in the following way, starting from $\text{root}(T)$. Suppose y is an internal node of T with children $y_1, \ldots, y_\delta$ and $\delta > 2$. Then, this node is replaced by a binary tree of depth at most $\lg \delta$ with leaves $y_1, \ldots, y_\delta$. The cost $c(\cdot)$ of $y, y_1, \ldots, y_\delta$ are the same as before and the cost of the newly created internal nodes are set to ∞; this is so that they never get selected in any solution. The construction is recursed on each of $y_1, \ldots, y_\delta$. It is easy to see that the optimum solution of (1) on the new tree is the same as on T. Furthermore, the size of the new tree at most doubles and the depth of the tree increases by a factor of $\lg \Delta$, where Δ is the maximum degree of a node in T. This construction has been used previously; see for instance [12, 29].

From now on, we will assume that the tree is binary. Let S denote the current solution set. Let $C(x, S, k)$ be the cost of the best subtree rooted at node x using a budget of k, given that S is the current solution. Recall that S_x, if it exists, is the first node along the ordered path from x to the root of the tree T in the current solution S. If S_x exists, then all $x' \in \text{leaf}(T_x)$ (leaves in the subtree under x) can always be covered by S_x, each with cost $d(x', S_x)$.

Let x_1, x_2 denote the two children of x. The update rule for the dynamic program is given by

$$C(x,S,k) = \min \begin{cases} \min_{k'=1}^{k}(C(x_1,S,k') \\ \quad + \ C(x_2,S,k-k')) \\ c(x,S) + \min_{k'=1}^{k-1}(C(x_1,S\cup\{x\},k') \\ \quad + \ C(x_2,S\cup\{x\},k-k'-1)). \end{cases} \tag{2}$$

The top term corresponds to not choosing x to be in S and the bottom term corresponds to choosing x to be in S.

The base cases for the dynamic program are

- $C(x,S,k)$ where $x \in \text{leaf}(T)$. If we forbid including leaves in the solution and S_x doesn't exist, we set this cost to be ∞. If leaves of T are permitted to be part of the solution, the cost is given by

  $$C(x,S,k) = \begin{cases} \min\{c(x,S), d(x,S_x)\} & \text{if } S_x \text{ exists} \\ c(x,S) & \text{otherwise} \end{cases}$$

  (Note that if exactly k nodes are desired, then we can set $C(x,S,k)$ to ∞ whenever $k > 1$.)

- $C(x,S,0)$, where the cost is given by

  $$C(x,S,0) = \begin{cases} \sum_{x' \in \text{leaf}(T_x)} d(x',S_x) & \text{if } S_x \text{ exists} \\ \infty & \text{otherwise} \end{cases}$$

  This corresponds to assigning all nodes in the subtree T_x to the node S_x, if it exists, since we are out of budget in this case.

The dynamic program is invoked as $C(\text{root}(T), \emptyset, k)$. There are $knd \lg \Delta$ entries in the dynamic programming table and each update of an entry takes $O(k)$ time as given by (2). So, the total running time of the dynamic program is $O(k^2 \cdot n \cdot d \cdot \lg \Delta)$. Note that in the dynamic program, we compute $c(x,S)$ in terms of a partial solution set S that has been constructed so far and not in terms of the final solution set S as indicated in (1); however, since $c(x,S)$ depends only on the elements in S that are on the path from x to $\text{root}(T)$ and since we compute S top down, these are equivalent.

PROPOSITION 1. *The above algorithm solves the optimization problem in (1). The running time of the algorithm is* $O(k^2 \cdot n \cdot d \cdot \lg \Delta)$.

Notice that the node selection cost $c(\cdot)$ is helpful to incorporate heuristic choices and requirements. For instance, we might want to ensure that if two nodes, one of which is a parent of the other, are chosen in the solution, then they are guaranteed to have different distributions. This is accomplished by setting $c(\cdot)$ for the child node to be sufficiently high in such situations.

4.2 Cost Measures

We now suggest some different variants of the node selection cost c and the cohesiveness cost d that appear in (1). The different choices of these functions result in algorithms that make different trade-offs and are optimized for different conditions.

4.2.1 Cohesiveness Costs

We propose three cohesiveness costs that capture the purity of a topical segment. The first cost is based on information theory and the next two are geometric in nature.

KL-COST MEASURE. This cost measure is based on the Kullback–Leibler divergence in information theory. For every page x and the node S_x to which it belongs we define the cost of the assignment to be

$$d(x,S_x) = \text{KL}(p_x \parallel p_{S_x}) = \sum_{\ell \in L} p_x(\ell) \log\left(\frac{p_x(\ell)}{p_{S_x}(\ell)}\right).$$

The KL-divergence or the relative entropy of two distributions p_x and p_{S_x} over an alphabet L is the average number of extra bits needed to encode data drawn from p_x using a code derived from p_{S_x}. This corresponds to minimizing the wastage in description cost of leaves of the tree using the internal nodes that are selected. This properties make the KL-divergence an intuitive choice for the cohesiveness cost.

SQUARED EUCLIDEAN COST MEASURE. The distance between a leaf x (webpage) and an internal node S_x (directory) can be computed using the squared Euclidean distance between the corresponding class distributions. Therefore,

$$d(x,S_x) = \|p_x - p_{S_x}\|^2 = \sum_{\ell \in L} |p_x(\ell) - p_{S_x}(\ell)|^2.$$

The sum of squared Euclidean cost has previously been extensively used in many applications.

COSINE COST MEASURE. Drawing from information retrieval, the negative cosine dissimilarity measure may be employed as a cohesiveness cost, as follows:

$$d(x,S_x) = -\langle p_x, p_{S_x}\rangle = -\sum_{\ell \in L} p_x(\ell) p_{S_x}(\ell).$$

The cosine cost measure has previously been successfully used for clustering documents [3].

4.2.2 Node Selection Costs

Having presented three possible cohesiveness costs $d(\cdot)$, we turn now to the node selection cost $c(\cdot)$, representing the penalty for adding a new element into S. Our goal is to penalize a new node if it provides little information beyond its parent. We propose a cost measure to implement this condition, which we call the α-measure. This cost measure in the context of decision tree induction was introduced by Quinlan [24] and is referred to as *information gain ratio*. It is defined as follows.

Let T be a tree consisting of subtrees $T_1, \ldots, T_s$. Say we wish to encode the label of a particular leaf of T, and are allowed two possible encoding schemes. In the first scheme, we simply communicate the label using an optimal code based on the distribution of labels in T. In the second scheme, we first communicate whether or not the designated leaf lies in T_1, and then encode the label using a tailored code for either T_1 or $T \setminus T_1$ as appropriate. The second scheme corresponds to adding T_1 to the segmentation. Its overall cost cannot be better than the first, but if T_1 is completely distinct from $T \setminus T_1$ then (and only then) the cost of the second scheme will be equivalent to the first. Let $p_1 = |T_1|/|T|$ be the probability that a uniformly-chosen leaf of T lies in T_1. Then the cost of communicating whether a leaf lies within T_1 is $H(p_1)$. In the worst case, T_1 will look identical to $T \setminus T_1$ and the second scheme will actually be $H(p_1)$ bits more expensive than the first: the information about the subtree provides no leverage to the user. We may therefore characterize the value of subtree T_1 relative to its parent by asking where on the scale between $H(T)$ and $H(T) + H(p_1)$ the cost of

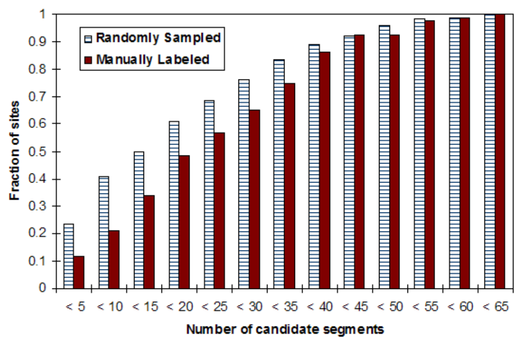

Figure 2: Cumulative distribution of number of candidate segments for all sites in our sample and for the sites we sampled for manual segmentation.

the second scheme lies. With this intuition in mind, we now provide the formal definition of the cost measure.

Let x denote the current node we are considering adding to the solution S. Recall that S_x is its nearest parent that is already a part of the solution S. We assume S_x exists (we will discuss this restriction further below) and for simplicity, denote S_x by y. Then let x' be a hypothetical node such that $\text{leaf}(T_{x'}) = \{\text{leaf}(T_y) \setminus \text{leaf}(T_x)\}$, i.e., the leaves under the subtree rooted at y but not x. Let $n = |\text{leaf}(T_y)|, n_x = |\text{leaf}(T_x)|$, and $n_{x'} = |\text{leaf}(T_{x'})|$. Here, the split cost is $H_2(n_x/n)$, the binary entropy. Then, the α-measure is defined to be

$$\alpha(x, y) = \frac{(n_x/n)H(x) + (n_{x'}/n)H(x') + H_2(n_x/n) - H(y)}{H_2(n_x/n)}$$

It can be seen that α takes values between 0 and 1, with lower values indicating a good split. The cost of adding a node to the solution is then

$$c(x, S) = c(x, y) = \alpha(x, y) \cdot n_x.$$

One requirement of using α-measure in the dynamic program is that we always need to select the root of T, i.e., $\text{root}(T) \in S$, in order to compute the cost of adding additional internal nodes. The requirement is not entirely unreasonable since the root directory of most sites contain a large number of files that cannot be made part of the solution on their own right and need the root to cover them.

5. EXPERIMENTS

In this section we evaluate our algorithms on their ability to segment sites obtained from the World Wide Web. First, Section 5.1 describes the hand-labeled and semi-synthetic benchmark datasets we created, and Section 5.2 gives an overview of the experiments we run based on these benchmarks. Then in Section 5.3 and Section 5.4 we study the performance of our algorithm on both the benchmarks. In Section 5.5, we study the performance of the three cohesiveness cost measures with and without the node selection cost based on α-measure.

5.1 Website Segments: Obtaining Labeled Data

We used a page-level classifier available within Yahoo! that classifies pages into a taxonomy of 90 topics selected from the Yahoo! directory. From the site listings of these 90 topics we picked a random set of 2150 sites. For each of these sites we fetched all the URLs indexed by Yahoo!, (up to a maximum of 1000 per site) and applied the classifier in order to determine their assigned category labels. The category labels have an associated confidence measure that is ignored for the purposes of these experiments.

Hence, a site is represented by a set of URLs, each of which is labeled by one of 90 topics. In the URL tree corresponding to a particular website, a *candidate segment* is defined as a node of the tree that has at least 1% of the total pages on the site under its subtree. In Figure 2 we plot the cumulative distribution of sites with different numbers of *candidate segments*. As we can see almost 25% of all sites have fewer than 5 nodes that can be selected as segments, and more than 50% have less than 15. We should note that in order to avoid uninteresting solutions to the HTS problem, we only consider sites that have at least two candidate nodes and more than 300 pages.

From this dataset, we generated two benchmark datasets. We refer to the first set as the *hand-labeled* website segments; this set contains 100 sites manually segmented into topically-cohesive regions. We refer to the second set as *semi-synthetic* website segments; it contains 1750 synthetically-generated websites. Each such site is created by artificially grafting together uniform regions from varying numbers of other websites, thus representing a benchmark with an unambiguous, known segmentation. We now describe the creation of each benchmark dataset in more detail.

Hand-Labeled Website Segments. We randomly sampled and manually segmented 100 websites from the Yahoo! directory. While sampling the sites around 10% were deemed to have a trivial directory structure based on URLs and were skipped. The cumulative distribution of the number of candidate segments in sites we labeled is plotted in Figure 2. As seen, only 10% of sites sampled for labeling have fewer than 5 candidate segments as compared to nearly 25% of the set of randomly sampled sites. Hence, while skipping websites with no directory structure we biased our sample towards sites that have larger number of candidate segments. This serves our purpose of robust evaluation of our approach as segmenting sites with very few candidate segments is a trivial and uninteresting task. Of 100 websites that were segmented 74 were segmented into two or more parts while the rest were labeled completely homogeneous (only one segment at the root directory). Among the former set of websites, the average number of segments per site was around 7, with the maximum being 18.

The criteria employed for manually selecting segments were the following. We always assumed that one segment is anchored at the root-level directory of the site; this was done to ensure complete coverage of all webpages. Subsequently, any directory that contained pages on a topic different from the aggregated topics at the root directory of the site was selected as a segment. We avoided selecting segments that were smaller than 1% of the site's size and those that were immediately enclosed within another segment on the same topic. These criteria were chosen to model the requirements mentioned in Section 3.1.

Semi-Synthetic Website Segments. The hand-labeled data can be used to measure our algorithm's ability to detect website segments as identified by humans. However, in order to perform more controlled evaluation of the algorithm's behavior we created a dataset with semi-synthetic segments. For this purpose we used the 26 sites that were manually labeled as homogeneous. We created a new site T' from site T_a by grafting k subtrees, from another set of sites $T_1, T_2, \ldots, T_k$, to internal directories of T_a. Since T_a and T_i are all relatively homogeneous w.r.t. topics, the new site tree T' should have $k+1$ segments (including the root directory), one from each of its constituent sites. We can now test our algorithms by measuring how many of the $k+1$ segments they discover. Certain precautions were taken while creating these hybrid sites. We only grafted subtrees that had 20 to 100 leaves under them. This ensures that the grafted subtree is larger than 1% of the hybrid site's size and that the grafted content doesn't overwhelm the existing content of T_a. If that happens, our algorithms might create segments of subtrees from the original T_a as they will now be significantly different from the topic distribution at the root directory. We created 7 such datasets for $k = 1, \ldots, 7$, each with 250 hybrid sites.

5.2 Measuring Segmentation Performance

We detail our methodology and metrics. In Section 3.2 we hypothesized that the "correct" segmentation (including the number of segments) can be detected by finding the k that minimizes (1). We perform experiments on the hand-labeled as well as semi-synthetic datasets to verify whether this hypothesis holds. All experiments are run to select at most $k' = 30$ segments from the set of *candidate segments* (nodes with at least 1% of website's webpages under them). The root always selected as a segment. As the final number of segments in solutions can vary, the best way to report results is by computing the precision and recall w.r.t. to the manually labeled segments, i.e., the true solution. The *precision* of a solution is the fraction of segments in the solution that were identified by our human judges as appropriate segmentation points. Similarly, the fraction of hand-identified segments that are found by the algorithm represents the *recall* of its solution. The *f-measure*—the harmonic mean $2pr/(p+r)$ of the precision p and recall r—can be used to report the quality of a solution as a single number. For each combination of cohesiveness cost and node selection cost, by varying β we can obtain solutions with different precision and recall values. We expect that configurations with high β would be very conservative in the number of segments they find, since they bias the cost function in (1) towards not adding a node. Hence, these configurations should have high precision but low recall. We expect the opposite behavior for low β values, with these configurations achieving low precision and high recall scores.

5.3 Performance on Semi-Synthetic Benchmark

The precision–recall curves for the performance of different cohesiveness and node selection cost combinations over the semi-synthetic websites are plotted in Figure 3. The precision and recall values on the plot are averaged over all the 7 datasets with $k = 1, \ldots, 7$. These curves were computed by varying β from 0 to 1 in increments of 0.1. It can be seen that as we increase the value of β from 0 to 1, the curves move from the area of low precision, high recall to the area of high precision, low recall. On the plot we identify β values that provide good trade-off for the three combinations of cost measures. For KL+Alpha, $\beta = 0.8$ results in a precision/recall value of 0.94/0.94, while for Euclidean+Alpha and Cosine+Alpha the values at $\beta = 0.5$ are 0.94/0.91 and

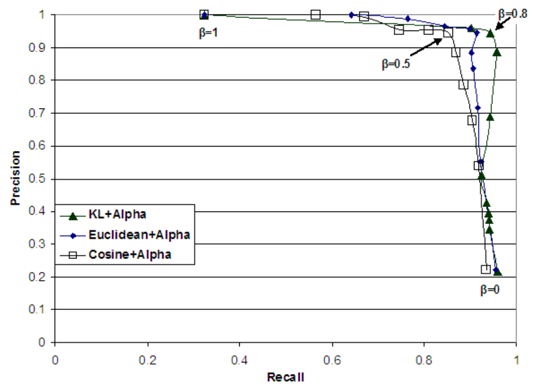

Figure 3: Precision–recall curves (with varying β) over the semi-synthetic benchmark. The values are averaged over all hybrid sites created (over different number of grafts settings).

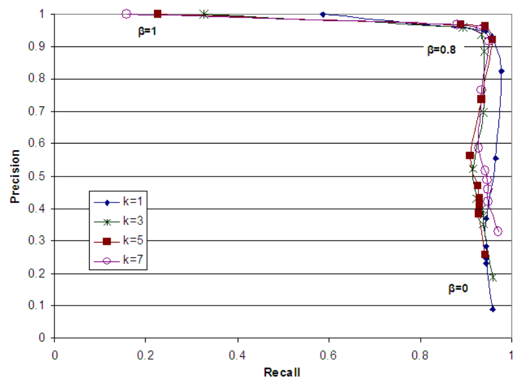

Figure 4: Precision–recall curves (with varying β) obtained by using KL+Alpha cost measures over the semi-synthetic benchmark. Different curves correspond to different number of grafts.

0.94/0.85 respectively. This shows that in the case semi-synthetic websites, all three cost combinations are able to find the correct number of segments and their locations.

Figure 4 plots the precision–recall curves of the KL+Alpha algorithm on semi-synthetic sites with varying number of grafts separately. Here we plot the data for $k = 1, 3, 5, 7$. We can see that the behavior of all the curves is similar, with best precision–recall trade-off at $\beta = 0.8$. The only difference between the four curves is at $\beta = 0, 1$. This is because when $\beta = 0$, the algorithm adds a large number of segments to the solution and hence the precision suffers for all curves. But number of true segments is different for each curve causing a different lowest precision value. Similarly, the lowest value of recall attained depends on the number of true segments and hence this value is lower for $k = 7$ than for $k = 1$. Finally, the best precision–recall values obtained are around 0.94/0.94 for all curves.

5.4 Performance on Hand-Labeled Benchmark

We consider the more difficult task of segmenting actual websites obtained from the World Wide Web. Figure 5 plots precision–recall curves for the performance of different cohesiveness and node selection cost combinations over the hand-

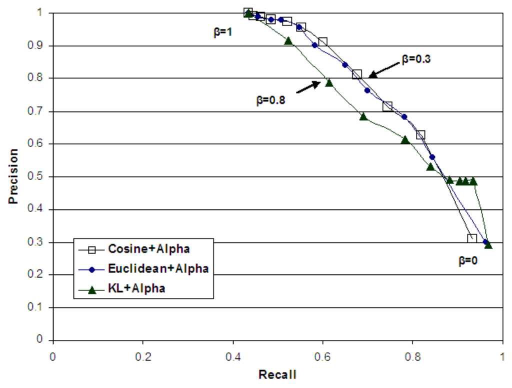

Figure 5: Precision–recall curve (with varying β) over the hand-labeled websites.

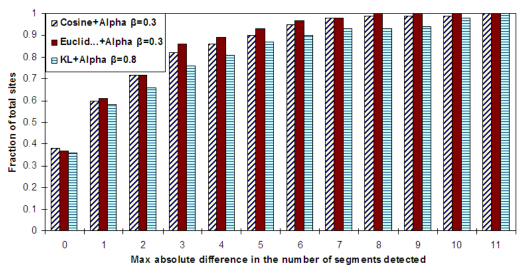

Figure 6: Cumulative distribution of the absolute error in the number of segments detected for the hand-labeled websites.

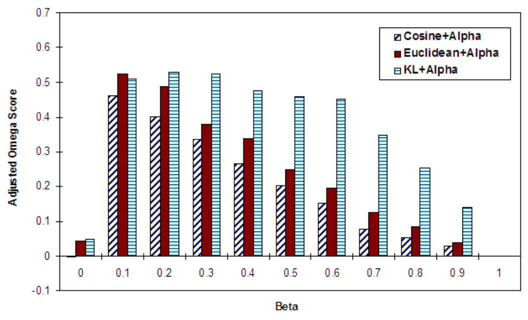

Figure 7: Adjusted Omega score obtained over the hand-labeled websites (with more than one true segment) for different values of β.

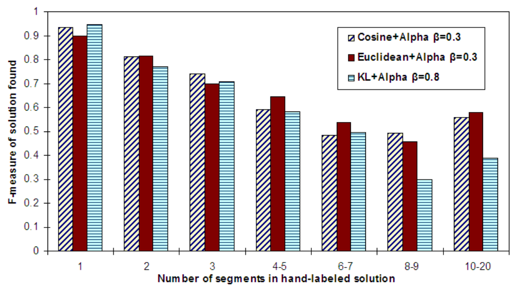

Figure 8: The averaged f-measure of segmentation found by the algorithm for websites with different number of segments in the labeled solution.

labeled dataset. These curves were computed by varying β from 0 to 1 as in Section 5.3. On the plot we identify β values that provide reasonable trade-off of precision and recall for the three combinations of cost measures. For the KL+Alpha combination, $\beta = 0.8$ results in a precision–recall value of 0.79/0.62, while for Euclidean+Alpha and Cosine+Alpha the values at $\beta = 0.3$ are 0.76/0.69 and 0.8/0.67 respectively. The curves in Figure 5 show that for a robust set of values of β our algorithm produces very good segmentations of websites.

Now that we have seen that the algorithm finds more or less the correct segments, lets take a closer look at how well the algorithm performs in estimating the "correct" number of segments. Figure 6 shows the cumulative distribution of error in the number of segments detected. Here, the magnitude of error is the absolute difference in the number of segments found by our algorithm and the manual labeling. As seen, our algorithm finds the correct number of segments in nearly 40% of the cases and for more than 70% of cases the number of segments found is within ± 2 of the number of manually labeled segments. Furthermore, the performance of all three cost combinations is very similar.

Comparing with performance on semi-synthetic websites. The results in Figure 5 are similar to those for semi-synthetic benchmarks (Figure 3) in that the curves move from the area of low precision, high recall to the area of high precision, low recall as we increase β. There are, however, a couple of differences; the "knee" of the curves is more prominent and precision–recall values are higher for the semi-synthetic dataset.

These differences can be explained by pointing out that the true segments are much more unambiguously defined in the case of the semi-synthetic websites than the hand-labeled ones. In other words, in the case of semi-synthetic websites the benefit of adding a graft as a separate segment is much higher than the benefit of adding a subtree of T_a (see Section 5.1) as a segment to the solution. This makes it easier to distinguish the true segments from T_a and hence we obtain very high precision and recall values. In the case of real websites, the benefit of adding nodes as segments are often very close to each other and in many cases the segment boundaries are fuzzy. Hence, even though we get fairly high values of precision and recall, there is a large range of β values over which the precision–recall trade-off is good.

A relaxed performance criterion. The precision–recall curves plotted above take into account only segmentation points (directories), and treat even small differences in segmentation boundaries as total errors. Two segmentations with slightly different boundaries are equally acceptable if these differences do not impact too many webpages. Here we evaluate our algorithms using a measure that considers the context of segments as well as the segmentation points: the *Omega* measure, which has been previously used for comparing overlapping clusterings [7]. The solutions found by our algorithms can be considered overlapping clusterings, with each segment in the solution acting as a cluster and each webpage belonging to all segments on its path to the root. In this context, the Omega measure computes the fraction of pairs of webpages that occur together under the same number of segments in both the segmentation being evaluated and the manually created segmentation. In Figure 7, we

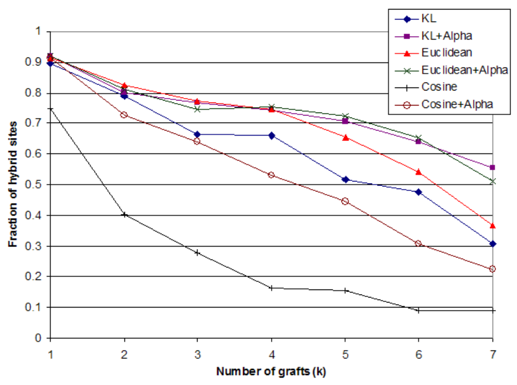

Figure 9: The fraction of runs in which all grafts in the hybrid tree were found vs number of grafts.

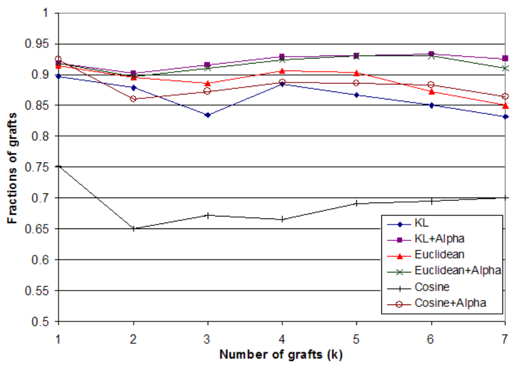

Figure 10: The recall of grafts in the hybrid tree vs number of grafts.

plot the Omega measure adjusted so that the expected performance value of a random segmentation is zero. Hence, a value of 0.5 can be interpreted as meaning that the segmentation under evaluation shows a 50% agreement with the manual segmentation, over and above any agreement that can be expected due to chance. The results in this plot are similar to the results in Figure 5, though with the higher recall (low β) region translating to slightly higher omega scores. The interesting point to note is that while the best performance of all cost measure combinations is similar, the KL+Alpha combination has the desirable property of giving good results over a much larger range of β than the other two cost measures.

PERFORMANCE VARIATION WITH NUMBER OF SEGMENTS. Here we want to evaluate our algorithm's performance on tasks of varying difficulty. In general, it is easier for the algorithm to find all the labeled segments in a site-tree if the number of segments is small. A partial reason is that the variance in the manual segmentation of site-trees increases as the number of prospective segments increases. Relatively cohesive sites with few directories of topically different content are easy for a human (and our algorithms) to segment. In Figure 8 we plot the f-measure of the segmentation found by our algorithm for websites in the hand-labeled set with different number of segments. From the plot we see that as the number of segments in the true solution increases, the f-measure drops to around 0.6 for both Euclidean and Cosine cohesiveness cost measures. The performance of the KL+Alpha combination, however, decreases significantly as the number of segments in the solution increases.

5.5 Exploring the Role of α-Measure

The α-measure acts as a regularization term in our objective function and is necessary to discover the correct number of segments. In this section we want to evaluate whether the α-measure also plays a role in the selection of good segments, or at least in avoiding bad candidates. For this experiment we use our algorithm to segment the website trees in the semi-synthetic dataset into a *specified* number of segments. The intuition behind these experiments is that since the number of segments is fixed, the α-measure will only be able to affect the specific segments selected for the solution and not how many are selected. This will give us a way to compare solutions obtained with and without the use of α-measure to determine its impact. The reason we use the semi-synthetic dataset is because unlike the hand-labeled dataset, here the number of true segments is well-defined.

As stated in Section 4.1, our algorithm can be modified to work when the number of segments is fixed a priori. We used this modified algorithm to segment each tree into $k' = k + 1$ segments (number of grafts plus the root). The β value used for KL+Alpha combination was 0.8, while for Euclidean+Alpha and Cosine+Alpha it was 0.3. To run our algorithm without the α-measure, β was set to 0. Results for each different value of k are summarized in Figures 9 and 10. Each point in the plot is averaged over 250 websites.

Figure 9 plots the average fraction of sites (out of 250) for which all the graft points in the hybrid tree were detected by the algorithm as a function of the number of grafts (k). As we increase the number of grafts, the difficulty of identifying all the grafts increases and the fraction of sites perfectly segmented decreases. All cost measure combinations other than Cosine perform almost identically for $k = 1$, but as we increase k, their performance numbers diverge significantly. In all cases, techniques that use the α-measure perform better than their counterparts that don't. Moreover, as the difficulty of the task increases, the decrease in accuracy of techniques using α-measure is gentler.

Figure 10 plots the fraction of total grafts that were discovered (recall) by the algorithm for each value of k. The root segment of the hybrid tree, which is always detected, is not considered a graft and hence not counted in the fraction of grafts detected. As we can see, in spite of the fact that performances in Figure 9 fall drastically as k is increased, the values in Figure 10 stay relatively the same. This shows that even though the algorithms aren't able to segment the entire hybrid tree perfectly as the problem becomes harder, they do discover most of the segments. As in the earlier experiments, techniques that employ the α-measure perform better than their counterparts that do not. These two experiments show that the α-measure is useful not just in regulating the size of the solution but also in identifying the "correct" segments when the solution size is specified.

6. CONCLUSIONS

We considered the problem of identifying and segmenting topically cohesive regions in the URL tree of a large website. We developed a general framework to use various cost measures for describing the quality of a segmentation; we also provided an efficient algorithm to find the best segmentation in this framework. Our experiments on hand-labeled and semi-synthetic benchmarks confirm the soundness of our framework and suggest that a judicious choice of cost measures can improve significantly improve precision/recall. Interesting future work includes extending our framework and algorithms to deal with general graphs, especially those induced by hyperlinks.

7. REFERENCES

[1] R. Agrawal, S. Rajagopalan, R. Srikant, and Y. Xu. Mining newsgroups using networks arising from social behavior. In *12th WWW*, pages 529–535, 2003.

[2] D. J. Aumueller. A tool for gathering, analysing, exporting, and visualizing the structure of a website. Master's thesis, University of Leeds, Institute of Communications Studies, 2003.

[3] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra. Clustering on the unit hypersphere using von Mises–Fisher distributions. *JMLR*, 6:1345–1382, 2005.

[4] K. Bharat, A. Broder, J. Dean, and M. R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *JASIS*, 51(12):1114–1122, 2000.

[5] D. Blei and M. Jordan. Modeling annotated data. In *26th SIGIR*, pages 127–134, 2003.

[6] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext classification using hyperlinks. In *SIGMOD*, pages 307–318, 1998.

[7] L. M. Collins and C. W. Dent. Omega: A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. *Multivariate Behavioral Research*, 23(2):231–242, 1988.

[8] N. Craswell, D. Hawking, and S. Roberston. Effective site finding using link anchor information. In *24th SIGIR*, pages 250–257, 2001.

[9] L. Denoyer and P. Gallinari. Bayesian network model for semi-structured document classification. *Information Processing and Management*, 40(5):807–827, 2004.

[10] M. Diligenti, M. Gori, M. Maggini, and F. Scarselli. Classification of HTML documents by hidden tree-Markov models. In *6th ICDAR*, pages 849–853, 2001.

[11] M. Ester, H.-P. Kriegel, and M. Schubert. Web site mining: A new way to spot competitors, customers and suppliers in the world wide web. In *8th KDD*, pages 249–258, 2002.

[12] R. Fagin, R. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Multi-structural databases. In *24th PODS*, pages 184–195, 2005.

[13] R. Fagin, P. Kolaitis, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Efficient implementation of large-scale multi-structural databases. In *31st VLDB*, pages 958–969, 2005.

[14] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[15] D. Gibson. Surfing the web by site. In *13th WWW*, pages 496–497, 2004.

[16] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *14th WWW*, pages 830–839, 2005.

[17] W. L. Hsu. The distance-domination numbers of trees. *Operations Research Letters*, 1:96–100, 1982.

[18] S. D. Kamvar, M. T. Scholsser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *12th WWW*, pages 640–651, 2003.

[19] O. Kariv and S. L. Haikim. An algorithmic approach to network location problems, part II: p-medians. *SIAM J. on Applied Mathematics*, 37:539–560, 1979.

[20] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *14th ICML*, pages 170–178, 1997.

[21] H.-P. Kriegel and M. Schubert. Classification of websites as sets of feature vectors. In *IASTED Intl. Conf. on Databases and Applications*, pages 127–132, 2004.

[22] J. Pierre. Practical issues for automated categorization of web sites. In *ECDL 2000 Workshop on Semantic Web*, 2000.

[23] B. Piwowarski, L. Denoyer, and P. Gallinari. Un modèle pour la recherche d'information sur des documents structurés. In *6th Journées internationales d'Analyse statistique des Données Textuelles*, 2002.

[24] J. R. Quinlan. Induction of decision trees. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally in *Machine Learning* 1:81–106, 1986.

[25] F. Ricca and P. Tonella. Web site analysis: Structure and evolution. In *16th ICSM*, pages 76–86, 2000.

[26] U. Schonfeld, Z. Bar-Yossef, and I. Keidar. Do not crawl in the dust: Different urls with similar text. In *15th WWW*, 2006.

[27] R. Shah and M. Farach-Colton. Undiscretized dynamic programming: Faster algorithms for facility location and related problems on trees. In *13th SODA*, pages 108–115, 2002.

[28] A. Sun and E.-P. Lim. Web unit mining: finding and classifying subgraphs of web pages. In *12th CIKM*, pages 108–115, 2003.

[29] A. Tamir. An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59–64, 1996.

[30] L. Terveen, W. Hill, and B. Amento. Constructing, organizing, and visualizing collections of topically related web resources. *ACM Transactions on Computer-Human Interaction*, 6(1):67–94, 1999.

[31] M. Thelwall and D. Wilkinson. Finding similar academic web sites with links, bibliometric couplings and colinks. *Information Processing and Management*, 40(3):515–526, 2004.

[32] M. Theobald, R. Schenkel, and G. Weikum. Exploiting structure, annotation, and ontological knowledge for automatic classification of XML data. In *6th WebDB*, pages 1–6, 2003.

[33] Y. Tian, T. Huang, W. Gao, J. Cheng, and P. Kang. Two-phase web site classification based on hidden Markov tree models. In *IEEE/WIC International Conference on Web Intelligence*, pages 227–236, 2003.