A Graph-Theoretic Approach to Webpage Segmentation

Deepayan Chakrabarti Ravi Kumar Kunal Punera

Yahoo! Research
701 First Ave.
Sunnyvale, CA 94089.
{deepay,ravikumar,kpunera}@yahoo-inc.com

ABSTRACT

We consider the problem of segmenting a webpage into visually and semantically cohesive pieces. Our approach is based on formulating an appropriate optimization problem on weighted graphs, where the weights capture if two nodes in the DOM tree should be placed together or apart in the segmentation; we present a learning framework to learn these weights from manually labeled data in a principled manner. Our work is a significant departure from previous heuristic and rule-based solutions to the segmentation problem. The results of our empirical analysis bring out interesting aspects of our framework, including variants of the optimization problem and the role of learning.

Categories and Subject Descriptors

H.3.m [**Information Systems**]: Information Storage and Retrieval

General Terms

Algorithms, Experimentation

Keywords

Webpage sectioning, webpage segmentation, energy minimization, graph cuts, correlation clustering

1. INTRODUCTION

As web usage is increasing, content creation is becoming more mature, and its presentation even more sophisticated. Presentation involves placing different pieces of information on a webpage — each serving a different purpose to the end-user — in a manner that appears coherent to users who browse the webpage. These pieces have carefully-placed visual and other clues that cause most users to subconsciously segment the browser-rendered page into semantic regions, each with a different purpose, functionality, and content.

For our purposes, a segment is a fragment of HTML, which when rendered, produces a visually continuous and cohesive region on the browser window and has a unified theme in its content and purpose. Under this broad notion, the following can be termed webpage segments: left navigation menu bar, site-specific banner that might be at the top, navigation footer, links and abstracts of related webpages, banner and other forms of advertisements scattered throughout a page, copyright notices, terms of service, and contact information, and last but not the least, the actual content itself. Furthermore, depending on how the creator of the webpage chose to organize it, the content might itself be represented by multiple segments.

There are several applications of webpage segmentation. Segments demarcate informative and non-informative content on a webpage; they can also discriminate between different types of information. This is very useful in web ranking and web data mining applications. Consider a multi-word query whose terms match across different segments in a page; this information can clearly be useful in adjusting the relevance of the page to the query. Similarly, as we empirically show later in this paper, segments appropriately labeled as, say, informative and non-informative, can improve the precision of web mining tasks like duplicate detection. Identification of segments also plays a vital role in displaying webpages on screen-space constrained devices such as smart phones, PDAs etc.

Recognizing the importance of webpage segmentation, several papers have proposed solutions to this problem [2, 8, 10, 7, 20]. Most of these techniques involve simple rule-based heuristics. The heuristics typically utilize many features present on a webpage, including geometry-related features, and apply the rules in a greedy fashion to produce the segments. While a heuristic approach might work well on small sets of pages and for the specific tasks for which it was designed, it has several problems. First, it is hard to automatically adapt the heuristics to keep abreast of the inherently dynamic nature of presentation styles and content types on the web. Second, combining multiple heuristics that work well on different types of pages into a single all-in-one heuristic is a manually intensive trial and error effort. Third, since heuristics are inherently greedy, the solutions they produce tend to be local minima. These issues give rise to the following question: is there a more principled approach to webpage segmentation?

Our contributions. In this paper we consider the problem of automatically segmenting webpages in a principled manner. Our main contributions are the following.

(1) We formulate the segmentation problem in a combinatorial optimization framework. In particular, we cast it as a minimization problem on a suitably defined weighted graph, whose nodes are the DOM tree nodes and the edge-weights express the cost of placing the end points in same/different segments.

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

(2) We take this abstract formulation and produce two concrete instantiations, one based on correlation clustering and another based on energy-minimizing cuts in graphs, which is our centerpiece. Both these problems have good approximation algorithms that are practical. We show how to adapt these two problems to the specific constraints of webpage segmentation.

(3) The quality of segmentations depends heavily on the edge weights. We give a principled approach for learning these weights from manually labeled data.

(4) Through empirical analysis we show that the energy-minimizing formulation performs substantially better than the correlation clustering formulation. We also show that learning edge-weights from labeled data also produces appreciable improvements to accuracy.

(5) We apply our segmentation algorithms as a pre-processing step to the duplicate webpage detection problem, and show that this results in a significant improvement to accuracy.

Organization. In Section 2 we present the related work. The framework is described in Section 3. Our algorithms based on correlation clustering and energy-minimizing cuts are presented in Section 4. In Section 5 we present the results of our experiments. Finally, Section 6 contains concluding remarks.

2. RELATED WORK

Prior work related to this paper falls into two categories. Due to paucity of space we describe each work very briefly. Readers are referred to original publications for more details.

There is plenty of past work on automated segmentation of webpages [2, 10, 7, 13, 20]. Baluja [2] gives a learning based algorithm to split a webpage into 9 parts suitable for viewing on a small-screen device. For the same application, Chen et al. [10] construct a two level hierarchical view of webpages, while Yin et al. [20] give an approach for ranking subparts of pages so as to only display relevant ones. Kai et al. [7] give an algorithm that uses rule based heuristics to segment the visual layout of a webpage. On the flip side, Kao et al. [13] give an algorithm for webpage segmentation that relies on content based features. Other notable works that use DOM node properties to find webpage segments are by Bar-Yossef and Rajagopalan [3], who use it to find templates and by Chakrabarti et al. [9], who use it for enhanced topic distillation. Finally, Chakrabarti et al. [8] give an algorithm based on isotonic regression whose by-product is a segmentation of a webpage into informative and non-informative parts.

Our work is also closely related to approaches for clustering data embedded as nodes of a graph. In particular, we make use of the correlation clustering algorithm of Ailon et al. [1] and the energy minimization framework of Kolmogorov and Zabih [15]. These approaches and our modification to them are described in detail in Section 4.

3. FORMULATION

The DOM nodes comprising an HTML page can be viewed in three independent contexts, each contributing its own piece of information. First, each node in the DOM tree represents a subtree that consists of a set of leaf nodes with various stylistic and semantic properties such as headers, colors, font styles, links, etc. Second, each node occupies visual real-estate when rendered on a browser. Node layout provides information on both its semantics (e.g., nodes rendered on the bottom of the page are likely to be less important than those in the center) as well as its relationships to other nodes (e.g., segment boundaries are more likely to occur where neighboring nodes are farther away from each other). Finally, since the same visual layout can be obtained from syntactically different DOM trees, the particular DOM tree structure used by the content creator implies semantic relationships among nodes (e.g., nodes separated by a large tree distance are more likely to be unrelated, and thus to lie across segment boundaries). The three sources of information together dictate if a subset of DOM nodes should be in the same segment or in separate segments, and any reasonable algorithm for segmentation should judiciously use all the pieces of information.

Our proposed approach is to combine this information into a single objective function such that the minimization of this function would result in a good segmentation. The objective function will encode the cost of a particular way of segmenting the DOM nodes. Taking a combinatorial approach to segmentation offers several advantages. First, it allows us to experiment with minor variants of the objective function, and thereby different families of algorithms. Second, it lets us fine-tune the combination of the views: depending on the website or application context, the relative importance of any view can be varied. For instance, feature-based segmentation may be more meaningful on webpages from `wikipedia.org`, which have strong structural cues such as headers and links to page subsections. On the other hand, visual relationships among nodes may be more useful on free-form pages, such as user homepages. Any such domain-specific knowledge can be easily incorporated into the objective function, leaving the algorithm for its minimization essentially unchanged.

For this framework to be algorithmically feasible in web-scale applications, where the time constraints on processing webpage can be very strict and one has to deal with large DOM trees with many thousands of nodes, the objective function and the combinatorial formulation have to be carefully chosen. A particularly important question that needs to be addressed is how the latent constraints that exist among different subsets of nodes in the DOM tree can be captured succinctly. On one hand, considering each DOM node individually is insufficient to express any form of inter-node relationships. On the other hand, allowing the objective function to encode relationships among arbitrary subsets of DOM nodes is unwieldy and can lead to combinatorial explosion. Hence, as a trade-off between expressibility and tractability, we propose using objective functions involving up to two-way (i.e., pairwise) interactions. In other words, the objective function deals with only quadratically many terms, one for each pair of nodes in the DOM tree, and lends itself to be cast as a clustering problem on an appropriately defined graph.

Let $\mathcal{N}$ be the set of nodes in the DOM tree. Consider the graph whose node set is $\mathcal{N}$ and whose edges have a weight, which represents a cost of placing the end points of an edge in different segments. Let $\mathcal{L}$ be a set of labels; for now, we treat the labels as being distinguishable from one another. Let $D_p : \mathcal{L} \to \mathbb{R}^{\geq 0}$ represent the cost of assigning a particular label to the node p and let $V_{pq} : \mathcal{L} \times \mathcal{L} \to \mathbb{R}^{\geq 0}$ represent the cost of assigning two different labels to the end

points of the edge (p, q). Let $\mathcal{S} : \mathcal{N} \to \mathcal{L}$ be the segmentation function that assigns each node $p \in \mathcal{N}$ the segment label $\mathcal{S}(p)$. Consider the following cost of a segmentation $\mathcal{S}$:

$$\sum_{p \in \mathcal{N}} D_p(\mathcal{S}(p)) + \sum_{p,q \in \mathcal{N}} V_{pq}(\mathcal{S}(p), \mathcal{S}(q)). \quad (1)$$

Finding a segmentation to minimize this objective is precisely the elegant *metric labeling* problem first articulated by Kleinberg and Tardos [14]. This problem is NP-hard but has a $(\log |\mathcal{L}|)$-approximation algorithm that is based on LP-rounding, when V_{pq} is a metric.

While this objective is mathematically clean, it is still fairly general and less appealing from a practical point of view since it does not capture the nuances of the segmentation problem at hand. In particular, consider the following constraint.

CONSTRAINT 1 (RENDERING CONSTRAINT). *Every pixel on the screen can belong to at most one segment.*

The rendering constraint arises out of the DOM tree structure: the area on the screen occupied by a child node is geometrically contained inside that of its parent node in the DOM tree. Therefore, any segmentation should respect the rendering constraint, i.e., if it places the root of a subtree in a particular segment, then it has to place all the nodes in the entire subtree in the same segment. The question now is two-fold: is there a version of the objective in (1) that can be optimized by a simple combinatorial algorithm and how Constraint 1 can be incorporated into the formulation? We propose two ways of addressing these issues: the correlation clustering formulation and the energy-minimizing cuts formulation.

Correlation clustering formulation. Observe that the actual labels of segments do not really matter in segmentation. This leads to the following modifications to objective in (1): treat the label set $\mathcal{L}$ as indistinguishable, i.e., $D_p \equiv 0$, $V_{pq} = c_{pq}$, a constant, and to prevent trivial solutions because of these modifications, change the objective in (1) to minimize

$$\text{cclus}(\mathcal{S}) = \sum_{\mathcal{S}(p) \neq \mathcal{S}(q)} V_{pq} + \sum_{\mathcal{S}(p) = \mathcal{S}(q)} (1 - V_{pq}). \quad (2)$$

This is the *correlation clustering* problem, which has been addressed in a rich body of work. Notice that the second term in (2) penalizes pairs of nodes (p, q) that are better off being in separate segments (i.e., V_{pq} is small) , but are placed in the same segment by $\mathcal{S}$. Correlation clustering has simple and practical combinatorial approximation algorithms, including a recent one by Ailon, Charikar, and Newman [1] that approximates Equation (2) to within a factor 2.

Enforcing Constraint 1 directly in Equation (2), however, becomes tricky. Heuristically, there are two possible options for doing this. The first option is to consider only the leaf nodes of the DOM tree and perform segmentation on them. The problem with this approach is that leaf nodes in the DOM tree are often too small to have reliable features. By construction, they carry very limited information such as plain text, italicized, bold, within an anchortext, and so on. This makes the features sparse and noisy, thereby adversely impacting the quality of segmentation. In fact, we will use this approach as a baseline in our experiments. The second option is to go ahead with Equation (2) on $\mathcal{N}$, but post-process and apply Constraint 1 top-down. The problem here is that, even if a reasonably good solution to Equation (2) is found, post-processing might damage its quality in course of enforcing Constraint 1. We will not address the second option in this paper.

Energy-minimizing cuts formulation. Here, we still retain minimizing objective in (1) as our goal, but instead impose more structure on the costs V_{pq}. This will serve two purposes. The first is that we can obtain a combinatorial algorithm for the problem, as opposed to solving a linear program. The second is that the factor of approximation will be 2, instead of $\log |\mathcal{L}|$. The details of the structural assumption on costs is provided in Section 4.2.

We now specify how we handle Constraint 1 in this framework. We create a special label ξ, called the *invisible label*, and consider the segmentation $\mathcal{S} : \mathcal{N} \to \mathcal{L} \cup \{\xi\}$. The segment corresponding to the invisible label is called the *invisible segment*. The invisible label has two properties:

(1) only internal nodes of the DOM tree can be assigned ξ and

(2) for any $p, q \in \mathcal{N}$ where q is a child of p in the DOM tree, either both p and q must belong to the same segment, or p must belong to the invisible segment; in other words, either $\mathcal{S}(p) = \mathcal{S}(q)$ or $\mathcal{S}(p) = \xi$.

Intuitively, the invisible segment consists of nodes that were not meant to form a coherent visual segment (they are invisible when the HTML is rendered on screen), but are present only to maintain the tree structure of the DOM. Note that (2) ensures that if any internal node belongs to the invisible segment, all of its ancestors must be invisible as well. Therefore, by carefully setting D_p and V_{pq} when $\mathcal{S}(p) = \xi$ we can elicit segmentations from our algorithm that follow the DOM tree structure to different degrees.

4. ALGORITHMS

The form of the objective function is closely intertwined with the algorithms that we can use to optimize it. In the following, we look at extensions of two different algorithms, one based on correlation clustering and the other on energy-minimizing graph cuts, for the segmentation problem. Then, we discuss in detail the exact form of the individual costs $D_p(\cdot)$ and the pairwise costs $V_{pq}(\cdot, \cdot)$, and how these are learned from available training data.

4.1 Correlation clustering

The correlation clustering problem starts with a complete weighted graph. The weight $v_{pq} \in [0, 1]$ of an edge represents the cost of placing its endpoints p and q in two different segments; similarly, $(1 - v_{pq})$ represents the cost of placing p and q in the same segment. Since every edge contributes, whether it is within one segment or across segments, the segmentation cost function is automatically *regularized*: trivial segmentations such as one segment per node, or all nodes in one segment, typically have high costs, and the best segmentation is somewhere in the middle. In fact, the number of segments is picked automatically by the algorithm.

Note that the costs depend only on whether two nodes are in the same segment or not, and not on the labels of particular segments themselves. This imposes two constraints on using correlation clustering for segmentation. First, it precludes the use the invisible label ξ with its special properties

as discussed in Section 3. Hence, in order to satisfy Constraint 1, we must restrict the set of nodes to the set of leaf nodes, leaf($\mathcal{N}$), of the DOM tree. Second, pairwise costs between two nodes p and q must depend only on whether they belong to the same segment or not:

$$V_{pq}(\mathcal{S}(p),\mathcal{S}(q)) = \begin{cases} v_{pq} & \text{if } \mathcal{S}(p) \neq \mathcal{S}(q), \\ 1 - v_{pq} & \text{if } \mathcal{S}(p) = \mathcal{S}(q). \end{cases}$$

Thus, the objective function becomes

$$\text{cclus}(\mathcal{S}) = \sum_{\substack{p,q \in \text{leaf}(\mathcal{N}) \\ \mathcal{S}(p) \neq \mathcal{S}(q)}} v_{pq} + \sum_{\substack{p,q \in \text{leaf}(\mathcal{N}) \\ \mathcal{S}(p) = \mathcal{S}(q)}} (1 - v_{pq}).$$

We use the algorithm of Ailon, Charikar, and Newman [1] for correlation clustering to find a segmentation whose cost is within a factor of two of the optimal. For sake of completeness, we present a brief description of the algorithm, called CCLUS.

The algorithm CCLUS is iterative. At each stage, a node p in the current graph is chosen uniformly at random and removed from the graph. A new cluster is created with just p in it. Next, all the nodes q such that $v_{pq} \geq 1/2$ are removed from the graph and placed in the cluster along with p. The process is repeated on the remaining graph. Since the algorithm is randomized, several independent trials are performed and the solution with the least objective value is output as the answer.

4.2 Energy-minimizing graph cuts

In this section we discuss solving the objective in (1) using energy-minimizing graph cuts. First, we give some background on graph cuts.

There has been a lot of work in the computer vision community on representing energy functions as specially constructed graphs, and finding their minimum via max-flow cuts on those graphs [5, 15]. The class of energy functions that are "graph-representable" is very broad and there are algorithms to find local minima within a constant factor of the global optimum. Thus, this approach offers a good balance between expressibility and efficiency. In addition, our proposed objective in (1) can be made graph-representable with only a slight constraint on its form, which makes this technique even more attractive.

With this in mind, we now describe the definition of $D_p(\cdot)$ and $V_{pq}(\cdot,\cdot)$. First, we define V_{pq} to be non-zero only in two cases: (1) when nodes p and q are visually close neighbors in the rendering of the webpage, and (2) when p is a parent or child of q. Thus, pairwise relationships now encode the information available from the visual rendering and the DOM tree structure, while single-node D_p values, to be described in detail later, will encode node-specific feature information. Thus, *all* sources of information mentioned at the beginning of Section 3 are taken into account.

Next, we discuss the constraints on D_p and V_{pq}. Some of these come from our particular domain, while the others are required for efficient algorithms.

Domain constraints. The following constraints are dictated by our domain (note that the invisible segment label ξ is considered to be a member of $\mathcal{L}$).

$$\begin{aligned}
D_p(\xi) &= \infty \quad \text{for } p \in \text{leaf}(\mathcal{N}) && (3)\\
V_{pq}(\alpha,\beta) &= \infty \quad \text{for } \mathit{parent}(p) = q, \alpha \neq \beta, \beta \neq \xi && (4)\\
V_{pq}(\alpha,\xi) &= V_{pq}(\beta,\xi) \quad \text{for } \mathit{parent}(p) = q, \alpha, \beta \neq \xi && (5)\\
V_{pq}(\alpha,\beta) &= V_{qp}(\beta,\alpha) \quad \forall \alpha,\beta \in \mathcal{L} && (6)\\
V_{pq}(\alpha,\alpha) &= V_{pq}(\beta,\beta) \quad \forall \alpha,\beta \in \mathcal{L} && (7)\\
V_{pq}(\alpha,\beta) &= V_{pq}(\gamma,\delta) \quad \alpha \neq \beta, \gamma \neq \delta; p,q \in \text{leaf}(\mathcal{N}) && (8)
\end{aligned}$$

In other words, (3) leaf level nodes cannot become invisible, (4) when parent and child have different labels, the parent must become invisible, (5) the cost of a parent-child edge is independent of the child's label if the parent is invisible, (6) pairwise costs are symmetric, (7) the cost of belonging to the same label is independent of the label, and (8) for leaf nodes, the costs of belonging to different labels is independent of the labels. The last two constraints follow from the fact that there is no need for an ordering on the set of labels (segments) in our case; all that we want to know is if two nodes p and q belong to the same label (segment) or not.

Constraints for efficient energy minimization. In addition to the above constraints, one other condition must be met before efficient algorithms based on graph-cuts can be used. This is called *graph representability* [15]: an energy function is graph-representable if a graph can be constructed such that, for every configuration of its variables, the value of the function is given (up to a constant) by the minimum s-t cut on the graph. A energy function of the form of Equation (1) is graph-representable if and only if, for any segments $\alpha,\beta,\gamma \in \mathcal{L}$ such that $\mathcal{S}(p) = \alpha, \mathcal{S}(q) = \beta$ is a feasible state ([15, Theorem 4.1]):

$$V_{pq}(\alpha,\beta) + V_{pq}(\gamma,\gamma) \leq V_{pq}(\alpha,\gamma) + V_{pq}(\gamma,\beta) \qquad (9)$$

In our case, this translates to the following:

THEOREM 2 (GRAPH-REPRESENTABILITY). *The energy function 1, under the domain-specific constraints (3)–(8), is graph-representable if and only if, $\forall i,j \in \mathcal{L}$ such that $i \neq j$, $V_{pq}(i,j) \geq V_{pq}(i,i)$.*

PROOF. We proceed by case analysis on Equation (9). If $\gamma = \alpha$ or $\gamma = \beta$, the result follows vacuously.

If $\gamma \neq \alpha, \beta$ and $\alpha = \beta$, we replace all instances of β by α in Equation (9) to get

$$\begin{aligned}
V_{pq}(\alpha,\alpha) + V_{pq}(\gamma,\gamma) &\leq V_{pq}(\alpha,\gamma) + V_{pq}(\gamma,\alpha) \\
\Rightarrow 2 \cdot V_{pq}(\alpha,\alpha) &\leq 2 \cdot V_{pq}(\alpha,\gamma) \quad \text{using (6), (7),}
\end{aligned}$$

which implies that $V_{pq}(\alpha,\alpha) \leq V_{pq}(\alpha,\gamma)$ for all $\alpha \neq \gamma$. This is exactly the statement of the theorem.

If $\gamma \neq \alpha \neq \beta$ and $p,q \in \text{leaf}(\mathcal{N})$), from Equation (8), we have $V_{pq}(\alpha,\beta) = V_{pq}(\alpha,\gamma) = V_{pq}(\gamma,\beta)$. Using this in Equation (9) yields $V_{pq}(\gamma,\gamma) \leq V_{pq}(\gamma,\beta)$. Once again, we obtain the statement of the theorem.

If $\gamma \neq \alpha \neq \beta$ and $\text{parent}(p) = q$, any feasible solution must have $\beta = \xi$, otherwise $V_{pq}(\alpha,\beta) = \infty$ from Equation (4). Also, $\gamma \neq \beta = \xi$, implying that $V_{pq}(\gamma,\beta) = V_{pq}(\gamma,\xi) = V_{pq}(\alpha,\xi)$. Replacing β by ξ in Equation (9), and using the above formula, we get:

$$\begin{aligned}
V_{pq}(\alpha,\xi) + V_{pq}(\gamma,\gamma) &\leq V_{pq}(\alpha,\gamma) + V_{pq}(\alpha,\xi) \\
\Rightarrow V_{pq}(\gamma,\gamma) &\leq V_{pq}(\alpha,\gamma),
\end{aligned}$$

which finishes the proof.

Finally, the case when $\gamma \neq \alpha \neq \beta$ and $p = \text{parent(q)}$ follows by symmetry. □

In words, the above result states that the cost of merging two nodes to the same label must always be lower than the cost of breaking them apart. This implies, in turn, that if we have two nodes that we believe should belong to separate segments, then the pairwise costs V_{pq} cannot be used to encode this. The only way to do this is via the $D_p(\cdot)$ term and our choice of $\mathcal{L}$.

The form of $D_p(\cdot)$ and $V_{pq}(\cdot,\cdot)$. Consider a pair of nodes p and q that we believe should be separated in the final segmentation. Instead of pairwise costs, we can encode this by saying that their single-node costs $D_p(\alpha)$ and $D_q(\alpha)$ are very different for any label α, and $D_p(\cdot)$ and $D_q(\cdot)$ achieve their minima at different labels. Hence, any segmentation that maps them to the same label would pay a high price in terms of single-node costs. We encode this by using all internal nodes of the DOM tree as the set $\mathcal{L}$ of labels (plus the "invisible" label ξ), and constructing $D_p(\alpha)$ so that it measures the distance between the feature vectors corresponding to the node p and the internal node (label) α.

This particular choice of $\mathcal{L}$ has several advantages. First, it is available before the segmentation algorithm needs to run, and so a pre-defined set of labels can be provided to the algorithm as input. Second, since the labels are nodes themselves, the feature vectors we use for the nodes p can immediately be applied to them as well. Third, the labels are "tuned" to the particular webpage being segmented; for any statically chosen set of labels, there would probably exist many webpages where the labels were a poor fit. Finally, the most common reason behind a desire to separate nodes p and q in the final segmentation is that their feature vectors are very distinct; in this case, they will typically be close to different labels, leading to a segmentation that separates them. Thus, the form of $D_p(\cdot)$ and the chosen label-set act as a cost function trying to separate nodes with distinct properties, while $V_{pq}(\alpha,\beta)$ with $\alpha \neq \beta$ provides a push towards putting them together. $V_{pq}(\alpha,\alpha)$ does not serve any purpose, so we set it to zero:

$$V_{pq}(\alpha,\alpha) = 0 \quad \forall \alpha \in \mathcal{L}.$$

Algorithm. Our algorithm GCuts is based on the algorithm in [5, 15]. We start with a trivial segmentation mapping all nodes to an arbitrary visible label: $\mathcal{S}_0 : \mathcal{N} \to \mathcal{L}\backslash\{\xi\}$. Then, we proceed in stages, called α-*expansions*. In each α-expansion, we pick a label $\alpha \in \mathcal{L}$, and try to move subsets of nodes from their current labels to α so as to lower the objective function. The optimal answer for each α-expansion can be found using the minimum s-t cut of a graph, whose construction we will describe shortly. Note that this is the stage where the graph-representability of the energy function becomes critical. After the best max-flow cut is found, nodes connected to s have their labels unchanged, while nodes connected to t have their labels changed to α. Now, α-expansions are iteratively performed for all possible labels $\alpha \in \mathcal{L}$ until convergence. This algorithm produces a solution that is within factor two of the optimum [5].

Now, we describe the construction of the graph that is specific to segmentation. Define $P^{(s)}$ and $P^{(t)}$ to be two sets of nodes, with each parent node $p \in \mathcal{N} \setminus \text{leaf}(\mathcal{N})$ being represented once in each of the two sets. We start with an empty graph of nodes $\mathcal{V} = \text{leaf}(\mathcal{N}) \cup P^{(s)} \cup P^{(t)} \cup \{s,t\}$. The graph construction proceeds in three stages. The first stage handles the single-node costs. Consider a node p, currently with a label β. If $D_p(\alpha) > D_p(\beta)$, then add an edge $s \to p$ with weight $(D_p(\alpha) - D_p(\beta))$, else add an edge $p \to t$ with weight $(D_p(\beta) - D_p(\alpha))$. This construction is performed for all nodes $p \in \mathcal{N}$ (Figure 1, left).

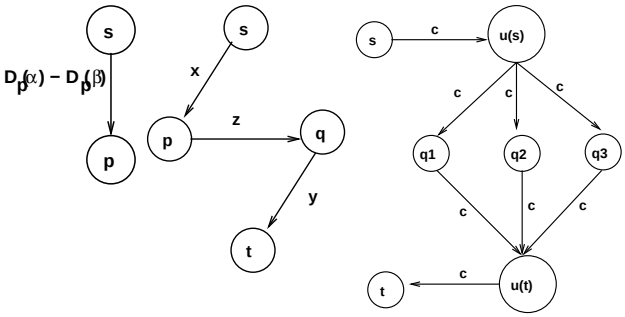


Figure 1: Representing single, pairwise leaf-node costs (assuming $x, y, z > 0$), and parent-child costs.

Pairwise costs between leaf nodes are handled in the second stage. Consider a pair of nodes $p, q \in \text{leaf}(\mathcal{N})$ that are visually rendered as neighbors. Let their current labels be β and γ respectively. We compute the following three values:

$$\begin{aligned} x &= V_{pq}(\alpha,\gamma) - V_{pq}(\beta,\gamma), \\ y &= V_{pq}(\alpha,\alpha) - V_{pq}(\alpha,\gamma), \\ z &= V_{pq}(\beta,\alpha) + V_{pq}(\alpha,\gamma) - V_{pq}(\beta,\gamma) - V_{pq}(\alpha,\alpha). \end{aligned}$$

If $x > 0$, we increase the weight of the edge $s \to p$ by $|x|$, otherwise we increase the weight of the edge $p \to t$ by $|x|$. If the edge did not already exist, it is assumed to have existed with weight 0. Similar edges are added for node q with weight y. Finally, we add edge $p \to q$ with weight z ($z > 0$ by the graph-representability of the objective). This process is repeated for all ordered pairs $p, q \in \text{leaf}(\mathcal{N})$ (Figure 1, middle).

While parent-child pairwise relationships could theoretically have been handled just as for pairs of leaf nodes, the graph generated in that fashion has the following problem. Consider one parent DOM node with many children. Suppose in a particular iteration of our algorithm all the children as well as the parent are labeled β, but the minimum energy would be obtained by having half the children labeled as γ (and the parent as ξ). This configuration will never be found via iterated α-expansions: an α-expansion with α set to γ will fail since the parent cannot be set to ξ in the same step (without which some parent-child pairwise cost would become ∞). Similarly, an α-expansion with α set to ξ will fail, since all the children will remain in label β, and then it is cheaper for the parent to remain in β as well.

Thus, we handle pairwise parent-child costs in a different fashion, and this constitutes the third stage of graph construction. Consider a parent node u, and its children $Q = \{q_1, \ldots, q_k\}$. Let $u^{(s)}$ and $u^{(t)}$ be the nodes corresponding to u in the nodesets $P^{(s)}$ and $P^{(t)}$ respectively. Let $c = \sum_{q \in Q} V_{qu}(\mathcal{S}(q), \xi)$ be the total cost incurred for moving parent u to the invisible label ξ. Then, we add the following edges to the graph, all with weight c: $s \to u^{(s)}, u^{(s)} \to$

$q_i(\forall i \in 1 \ldots k), q_i \rightarrow u^{(t)}(\forall i \in 1 \ldots k)$, and $u^{(t)} \rightarrow t$ (Figure 1, right). It can be shown that this construction handles parents in the correct fashion (proof omitted in this version).

4.3 Functional forms and learning of weights

So far, we have described cost functions to score the quality of segmentations as well as gave algorithms to optimize them. These cost functions can be decomposed into two principal types of expressions. In GCuts, $D_p(\alpha)$ is used to denote the cost of assigning label α to a node p, while $V_{pq}(\alpha, \beta)$ is cost of assigning label α to node p and label β to node q, when p & q are visually adjacent on a webpage. In the CClus algorithm, these pairwise costs are between all pairs of nodes irrespective of whether they are adjacent or not. In this section we will describe in detail the forms of these costs and how we learn their parameters from labeled data.

Node features. The cost functions compute the costs of placing nodes together or apart in the final segmentation taking into account certain features of nodes. For instance, if two nodes are far apart in the visual rendering of the webpage, all things being equal, we should have to pay a large cost if we want to place them in the same segment. However, the cost for placing them together should reduce if they share the same background color. On the flip side, even if two nodes are visually adjacent on the webpage, we should have to pay a high cost to place them in the same segment if they have text in dramatically different fonts or font sizes. Hence, we need to define, as node features, cues that help us compute the costs of placing a pair of nodes together or apart.

We define two main types of features: visual and content-based. The visual set of features include the webpage's position on the rendered webpage, its shape (aspect ratio), its background color, types, sizes, and colors of text, etc. On the other hand, the content-based features try to capture the purpose of the content in a DOM node. For instance, they include features such as average size of sentences, fraction of text within anchors, DOM nodes, tagnames, etc.

Learning from labeled data. Once feature values have been extracted for the nodes and the labels, we use machine learning tools to estimate the weight of each feature when assigning a cost for placing a pair of nodes together or away. The particular method used for learning depends on the form of the cost function. However, all methods learn the cost functions using the same set of manually labeled webpages. In a manually segmented webpage each DOM node is assigned a segment ID.

Learning $D_p(\cdot)$ in GCuts. As mentioned before, the $D_p(\alpha)$ cost in GCuts computes the cost of assigning DOM node p to a particular label α. The DOM nodes and the label can both be represented as a vector of features and cost of assignment then can be computed as the Euclidean distance between the two feature vectors. Given labeled data then our problem can be formulated as learning a Mahalanobis distance metric D such that the node p should be closer to the label that it has been assigned to than to all other labels. However, while the labeled data provides a segmentation of the nodes in a webpage, they don't provide the label for each of the segments. Hence, we adopt the following indirect way to learn the Mahalanobis distance metric D.

While the labeled data doesn't give us the true label for node p it does tell us which nodes Q are in the same segment as p and which nodes Q' are not. Consider nodes p and q that are in the same segment according to the ground truth. If our distance metric D' assigns a very small distance between p and q then it will also make sure that p and q are close to the same labels ($|D'(p,\alpha) - D'(q,\alpha)| \leq D'(p,q)$ from triangle inequality). Hence, we cast the problem of learning a distance metric D between a node and a label as that of learning a distance metric D' that would make try to ensure that pairs of nodes in the same segment are closer to each other than pairs of nodes across segments.

In order to learn this metric D' we employ a large margin method for learning a Mahalanobis metric using semi-definite programming [19]. This approach tries to learn a distance metric under which the k nearest neighbors of a datapoint belong to the same class while datapoints belonging to other classes are separated by a margin. In terms of our application, the method ensures that the k nearest neighbors to a DOM node (in terms of features) belong to the same segment, while DOM nodes in other segments are separated by at least a fixed distance. We will skip the further explanation of the approach here due to paucity of space, and the reader is referred to the original publication for details.

Learning $V_{pq}(\cdot,\cdot)$ in GCuts and CClus. The $V_{pq}(\cdot,\cdot)$ cost function for a pair of node p and q computes the cost of placing the nodes in the same segment. Since, from our manually labeled data we can determine for any pair of nodes whether they should be in the same segment or not, we can learn the parameters for V_{pq} in a more straightforward manner than for D_p above. Hence, for all pairs of nodes that are being included in the training set we obtain a class: 1 if the nodes are in the same segment, and 0 if they are not. Also for each such pair of nodes we obtain a feature vector, which represents tendencies of the nodes to occur together or apart. Then the problem becomes the standard learning problem of predicting the class based on the feature vector. We learn this classifier using the Logistic Regression algorithm [17], which has been shown to work well for two-class problems such as this, and which outputs the probability of belonging to one of the classes. We use this probability value as the V_{pq} score for a pair of nodes.

There are subtle differences in how the pairs are chosen for training when V_{pq} is being learned for GCuts and CClus. In GCuts, V_{pq} is only defined over node pairs that are visually adjacent to each other, while for CClus we train V_{pq} using all pairs of DOM nodes.

Learning λ values from validation-set. The two counterbalancing costs in the objective functions that our algorithms optimize are computed over a different set of entities. For instance, in GCuts, D_p is added once for each node in the graph, while V_{pq} is added once for each edge that is cut. Hence, a trade-off parameter λ is needed. This parameter is multiplied with V_{pq} in order bring the two costs into a useful equilibrium. The value of λ is estimated by varying it over a range while monitoring the changing segmentation accuracy of our approaches over a validation-set of labeled webpages.

5. EXPERIMENTAL EVALUATION

Now we present an empirical evaluation of our segmentation algorithms. First, we measure our system's performance on a set of manually segmented webpages. Our results indicate that the segmentations of webpages obtained by our algorithm match our intuitive sense of how information on a webpage is organized. Next, we evaluate the importance of various aspects of our system such as parent-child edges and the automatic learning of distance measures. Finally, we end our evaluation by demonstrating that segmentations output by our system can be used in a straightforward way to improve the accuracy of the duplicate webpage detection task.

5.1 Accuracy of segmentation

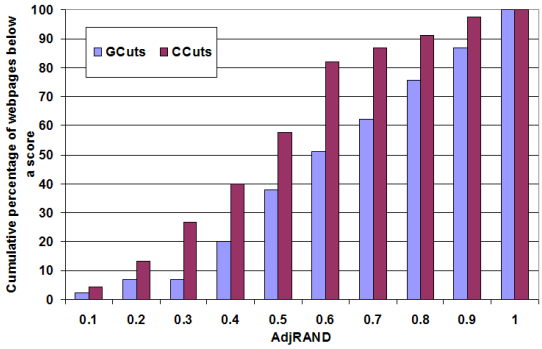
In order to evaluate the accuracy of our system we obtained a set of manually segmented webpages. We then ran our algorithms on these webpages and compared the segmentations generated by them with those obtained manually. Here we describe the results of these experiments.

Dataset used. We manually constructed a set of 1088 segments obtained from 105 webpages randomly sampled from the Web. For each segment all elements of a webpage that belong to it were marked out. These segments were then divided into three datasets — training-set, validation-set, and test-set — in, roughly, a 25:25:50 proportion respectively. Care was taken to ensure that segments from a webpage all belonged to only one of the datasets. Using procedures described in Section 4.3 the training-set was used to learn the distance measures used in both GCuts and CClus. Similarly, by monitoring the segmentation performance on the webpages in the validation set, we set the best values of λ for each approach. Finally, all accuracy numbers reported in this section were computed over the webpages in the test-set.

Evaluation measures. The segmentations output by our algorithms group the visual content of webpages into cohesive regions. As described above our test-data also consists of a manually constructed grouping of visual elements of a webpage. Hence, in order to evaluate the accuracy of our segmentation algorithms we use measures that are commonly used in the machine learning literature to compute the accuracy of clusterings w.r.t to ground truth.

The first such measure we use is the *Adjusted RAND* index (AdjRAND) [12], which is a preferred measure of accuracy of clustering [16]. The RAND index between two partitionings of a set of objects measures the fraction of pairs of objects that are either grouped together or placed in separate groups in both partitionings. Hence, higher RAND index values assigned to segmentations output by our algorithms (w.r.t. to manually labeled segmentations) indicate better quality. AdjRAND adjusts the values of RAND index so that it is upper bounded by 1 and scores 0 for a random segmentation.

In order to further showcase the differences between our algorithms we use *Normalized Mutual Information* as a second measure of accuracy of segmentation. NMI, introduced by Strehl and Ghosh [18], is the mutual information between two partitionings normalized by the geometric mean of their entropies $\text{NMI}(X,Y) = \frac{I(X,Y)}{\sqrt{H(X)H(Y)}}$. As with AdjRAND, higher values indicate higher quality, with 1 being the maximum. NMI has been commonly used in recent work for computing the accuracy of clustering algorithms.



(a) Cumulative distribution of AdjRAND scores for webpages in the test-set.

	GCuts $\lambda = 0.5$	CClus $\lambda = 0.6$
AdjRAND	**0.6**	0.46
NMI	**0.76**	0.64

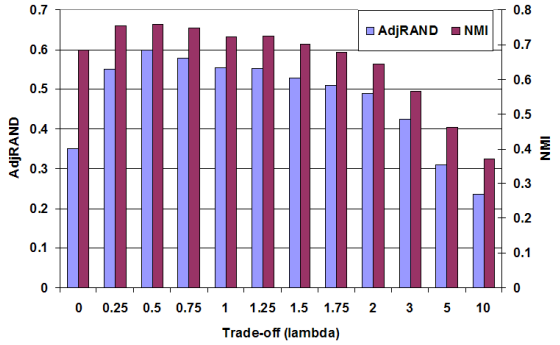
(b) Accuracy averaged over all webpages in test-set.

Figure 2: Comparing segmentation performance of GCuts and CClus.

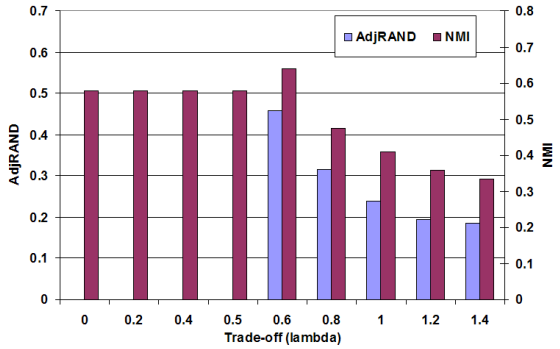
Comparing GCuts and CClus. Here we present our comparison of the accuracy of segmentations obtained by GCuts and CClus algorithms. As mentioned above, the best values of λ for both the algorithms was obtained using the validation-set of webpages; for GCuts, $\lambda = 0.5$ while for CClus, $\lambda = 0.6$. The segmentation accuracies of the two algorithms averaged over the webpages in the test-set values are shown in Figure 2(b). As we can see, in terms of both AdjRAND and NMI, GCuts far outperforms CClus. The performance difference in terms of AdjRAND and NMI are 30% and 18.75% respectively.

The results reported in Figure 2(b) are aggregate numbers over all webpages in the test-set. Now we break-down the results on a per-page basis. In Figure 2(a) we plot the cumulative percentage of webpages for which the segmentations output by the two algorithms score less than a certain value in terms of AdjRAND. Hence, the approach for which the bar graph rises faster from the left to right performs worse than the other approach. As we can see, GCuts outperforms CClus by a large margin. For example, GCuts scores less than 0.3 in AdjRAND for only 5% of all webpages in the test-set. The corresponding number for CClus is around 27%. On the flip side, GCuts scores higher than 0.6 in AdjRAND for around 50% of all webpages while for the CClus algorithm this number is less than 20%.

In Figure 3 we plot the average accuracy of segmentations obtained by both algorithms on the test-set documents against varying values of λ. As we increase λ, the costs for separating contents of the webpage into segments increase and the results tend to have fewer segments. Similarly, decreasing λ increases the costs for placing content together in a segment, and hence segmentations tend to have more numerous and smaller segments. In the extreme cases, very large (or very small) λ results in just one segment (or as many segments as nodes). As we can see in Figures 3(a) and 3(b), varying λ results in segmentations that vary widely



(a) GCuts



(b) CCLus

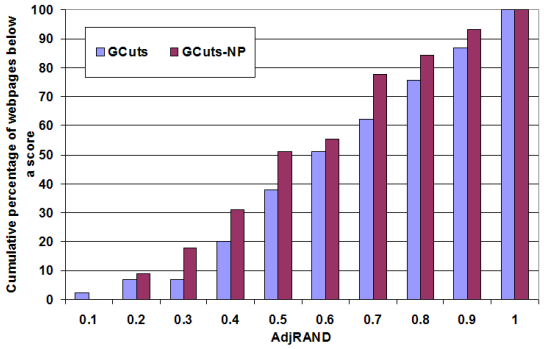
Figure 3: Performance of GCuts and CCLus averaged over the test-set webpages with varying λ.

in accuracy. Moreover, the values for λ that result in best accuracy seem to be similar to those predicted by using the validation set: 0.5 for GCuts and 0.6 for CCLus.

Impact of parent-child edges. As we have previously seen, GCuts outperforms CCLus by a significant margin. In this section we ask how much of this performance boost is due the fact that GCuts is able to use parent-child relationships from the DOM tree in constructing segmentations while CCLus only operates on the leaf-level graph. Here we use GCuts-NP to denote the use of GCuts with parent-child edges given zero weight.

As we explained in Section 3 we believe the addition of appropriately weighted parent-child edges to the leaf-level graph should help in obtaining a better segmentation. This is because parent-child edges are constructed by the creator of the page and hence implicitly encode some cues about how the page-creator wanted to structure information. Moreover, the similarity and dissimilarity between content is computed by comparing content properties such as fraction of links per word, font sizes, etc. The values for these properties are much more robust at high level DOM nodes, which contain sufficient text or occupy sufficient visible area, than at the leaf-level.

Figure 4 reports the results of running GCuts-NP on test-set webpages with the appropriate value of λ learned from the validation set ($\lambda = 1$). As we can see from Figure 4(b), removing the parent-child edges results in a decrease in accuracy of segmentation. The decrease is around



(a) Cumulative distribution of AdjRAND scores for webpages in the test-set.

	GCuts $\lambda = 0.5$	GCuts-NP $\lambda = 1$
AdjRAND	**0.6**	0.53
NMI	**0.76**	0.73

(b) Accuracy averaged over all webpages in test set.

Figure 4: Examining the effect of parent-child edges on segmentation performance.

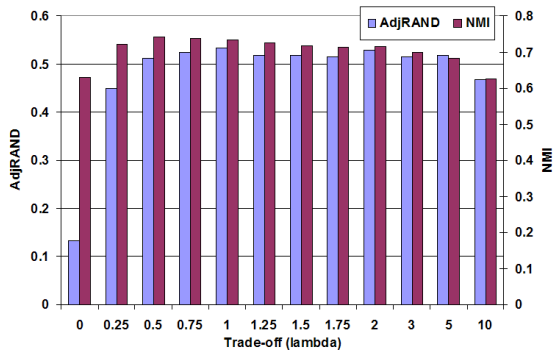


Figure 5: Performance of GCuts with parent-child edges disabled averaged over test-set webpages with varying λ.

10% in terms of the AdjRAND metric and around 5% in terms of term of NMI. This shows that being able to incorporate parent-child edges is essential to obtaining good webpage segmentation. Another interesting observation is that the accuracy of GCuts-NP is still better than the accuracy of the CCLus algorithm. This shows that the performance improvements of GCuts over CCLus are only partially due to the use of parent-child edges by GCuts. Finally, in Figure 4(a), we show the comparison of the GCuts and GCuts-NP at the level of webpages. While GCuts clearly outperforms GCuts-NP, the performances are similar towards the higher end of scores.

Figure 5 shows the performance of GCuts-NP averaged over webpages in the test-set for varying values of parameter λ. An interesting observation is that averaged performance of GCuts-NP peaks at a λ value which is higher than that for GCuts. This is because the removal of some parent-child edges from the cost function changes the two parts of

	GCuts $\lambda = 0.5$	GCuts-NL $\lambda = 1.5$
AdjRAND	**0.6**	0.52
NMI	**0.76**	0.73

Table 1: Segmentation accuracy of GCuts and GCuts-NL averaged over all webpages in test set

the cost function by different amounts. Since each parent contributes only once to the D_p part of the cost but as many times as the number of its children to the V_{pq} part of the cost, the absence of parent-child edges reduces the V_{pq} part more. Hence a larger λ is needed to obtain the right equilibrium between the two counterbalancing parts of the cost function.

Impact of learning of weights. Recall that the D_p cost takes the form of a Mahalanobis metric, while V_{pq} take the form of weighted sums of features passed to a sigmoid function. As described in Section 4.3 the Mahalanobis metric is learned using a metric learning algorithm [19] and the feature weights for V_{pq} are learned using Logistic Regression [17]. Instead of learning the parameters from labeled data, however, we could have used the Euclidean metric and a uniformly weighted sum of features in the two cases respectively. Here we measure the accuracy gains achieved by learning the distance measures used in GCuts and CClus.

Here, by GCuts-NL we denote our GCuts algorithm run with D_p and V_{pq} functions set without learning as mentioned above. Similarly, for CClus-NL too we replace the V_{pq} as mentioned above. The best values of λ for GCuts-NL and CClus-NL are still computed using the validation-set webpages.

Table 1 compares the accuracy of segmentations obtained by GCuts and GCuts-NL. Both the AdjRAND and NMI measures indicate a gain in segmentation accuracy for GCuts over GCuts-NL, agreeing with our intuition that data-based learning of the distance function helps the algorithm. For CClus-NL, the segmentation completely fails when the V_{pq} is used without learning, and so its results are not reported. The results on these experiments show that learning the parameters of the distance function from labeled data is critical to the performance of our algorithms.

5.2 Applications to duplicate detection

Duplicate webpages use up valuable search engine resources and deteriorate the search user-experience. Consequently, detection of these duplicate webpages is an active area of research by search engine practitioners [4, 6]. In this section, we will use this application to showcase the performance of our webpage segmentation algorithms.

Webpage segmentation for duplicate detection. Most duplicate detection algorithms rely on the concept of shingles [6], which are extracted by moving a fixed size window over the text of a webpage. The shingles with the smallest N hash values are then stored as the signature of the webpage. Two webpages are considered to be near-duplicates if their signatures share shingles. Moreover, typical shingling algorithms consider all content on a webpage to be equally important: shingles are equally likely to come from any part of the webpage.

This is problematic because a significant fraction of content on webpages is noisy [11]. These types of content, typically navigation bars, copyright notices, etc., do not represent the core functionality of the webpage and should not be used to compare whether two webpages are duplicates or not. Moreover, such noisy segments often manifest themselves by being repeated across several pages on a website. This might cause a shingling approach to consider two distinct webpages from the same website to be duplicates in case the shingles hit the noisy content. Similarly, false negatives might occur if two true duplicate webpages exist on different websites with different noisy content. In this section we use our segmentation algorithm as a pre-processing step before finding shingles on the webpages. Moreover, we only shingle the content that is contained within the largest segment found on the webpage, working under the assumption that the largest segment typically contains the informative content. As we will show next this simple approach results in significant increase in the accuracy of duplicate detection.

	Total Pairs	GCuts	CClus	FullText
Duplicate	1711	**1056** (**61.7**%)	587 (34.3%)	529 (30.9%)
Non-Duplicate	1707	**1706** (**99.9**%)	1620 (94.9%)	1515 (88.7%)

Table 2: Number of duplicate and non-duplicate pairs detected by the shingling approach upon using content in largest segment found by removing the noisy segments detected by GCuts and CClus. FullText indicates all text in the webpage was used.

The method employed in this section is not proposed as a duplicate detection scheme. In a realistic setting, one would employ a function that would label each segment found by our approach as informative or noisy content. Then all the content within the informative segments will be used for shingling. However, we wanted to isolate the performance of only our segmentation approach without involving an additional labeling function, and hence we chose to only use the content within the largest segment for shingling.

The Lyrics dataset. We use the Lyrics dataset to demonstrate the improvements given by our segmentation algorithms when they are used as a pre-processing step to duplicate detection. The dataset is constructed by obtaining webpages containing the lyrics to popular songs from three different websites. Thus, we know a-priori that the webpages from different websites containing lyrics to the same song should be considered duplicates[1]. Moreover, we know that webpages containing lyrics of different songs irrespective of what website they come from should be considered non-duplicates. The dataset consists of 2359 webpages from the websites `www.absolutelyrics.com`, `www.lyricsondemand.com`, and `www.seeklyrics.com` containing lyrics to songs by artists ABBA, BeeGees, Beatles, Rolling Stones, Madonna, and Bon Jovi. The artists were deliberately chosen to be diverse to minimize the possibility of cover songs. In our experiments in this paper we used 1711 duplicate pairs (webpages with lyrics of the same song from different websites) and 1707 non-duplicate pairs (webpages with lyrics of different songs from the same website) from the Lyrics dataset.

[1]Actually, these might only be near-duplicates, due to transcription errors on the different pages. However, this affects all algorithms equally.

Experimental setup. While segmenting the webpages in the LYRICS dataset we used the best settings of GCUTS and CCLUS found using the validation set in Section 5.1. Hence GCUTS was run with the setting $\lambda = 0.5$ and CCLUS with $\lambda = 0.6$.

The standard shingling process was used. Before shingling, all text on the webpage was converted to lowercase and all non-alphanumeric characters were removed. The window size used for shingling was 6 words, and 8 shingles with the minimum hash values were used as the signature of a webpage. A pair of webpages were tagged as duplicates if their signatures shared at least 4 out of the 8 shingles. The duplicate detection results are reported under 3 different settings depending on the input to the shingling process: (1) text content within the largest segment found by GCUTS, (2) text content within the largest segment found by CCLUS, and (3) all text content on the webpage (FULLTEXT).

Results. The results from our experiments in this section are presented in Table 2. The numbers in bold are the results of shingling after using GCUTS as a pre-processing step; we can recover around 60% of duplicate and almost all non-duplicate pairs this way. Moreover, the performance with GCUTS for pre-processing is far better than the performance with FULLTEXT: 100% improvement for Duplicate Pairs and around 12% improvement for Non-Duplicate pairs. Finally, we can see that GCUTS significantly outperforms CCLUS in terms of segmentation accuracy.

It should be noted that the LYRICS dataset is not representative of the distribution of duplicate or non-duplicate pairs on the Web and hence these improvements are not directly translatable to the Web in general. However, this dataset serves as a useful comparison for different segmentation mechanisms that can act as pre-processing steps to the shingling process. Finally, we must note that these impressive gains in performance were achieved using the simple heuristic of retaining the text contained in only the largest segment. Therefore, more sophisticated methods for discriminating between informative and noisy content (such as in [8]) can be expected to give significant further improvements when used to label the segments obtained by GCUTS.

6. CONCLUSIONS

In this paper we studied the problem of webpage segmentation. We proposed a combinatorial approach to this problem and considered two variants, one based on correlation clustering and another based on energy-minimizing graph cuts. We also proposed a framework to learn the weights of our graphs, the input to our algorithms. Our experiments results show that the energy-minimizing cuts perform much better than correlation clustering; they also show that the learning the weights helps improve accuracy. An interesting future direction of research is to improve the efficiency of the graph-cut algorithm for our special case. Another direction is to apply our algorithm in specialized contexts, such as displaying webpages on small-screen devices.

7. REFERENCES

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. In *37th STOC*, pages 684–693, 2005.

[2] S. Baluja. Browsing on small screens: Recasting web-page segmentation into an efficient machine learning framework. In *15th WWW*, pages 33–42, 2006.

[3] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *11th WWW*, pages 580–591, 2002.

[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] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *PAMI*, 23(11):1222–1239, 2001.

[6] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *WWW6 / Computer Networks*, 29(8-13):1157–1166, 1997.

[7] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Extracting content structure for web pages based on visual representation. In *5th Asia Pacific Web Conference*, pages 406–415, 2003.

[8] D. Chakrabarti, R. Kumar, and K. Punera. Page-level template detection via isotonic smoothing. In *16th WWW*, pages 61–70, 2007.

[9] S. Chakrabarti, M. Joshi, and V. Tawde. Enhanced topic distillation using text, markup tags, and hyperlinks. In *24th SIGIR*, pages 208–216, 2001.

[10] Y. Chen, X. Xie, W.-Y. Ma, and H.-J. Zhang. Adapting web pages for small-screen devices. *Internet Computing*, 9(1):50–56, 2005.

[11] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *14th WWW (Special interest tracks and posters)*, pages 830–839, 2005.

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

[13] H.-Y. Kao, J.-M. Ho, and M.-S. Chen. WISDOM: Web intrapage informative structure mining based on document object model. *TKDE*, 17(5):614–627, 2005.

[14] J. M. Kleinberg and É. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields. *J. ACM*, 49(5):616–639, 2002.

[15] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *PAMI*, 26(2):147–159, 2004.

[16] G. Milligan and M. Cooper. A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate Behavioral Research*, 21(4):441–458, 1986.

[17] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[18] A. Strehl and J. Ghosh. Cluster ensembles — A knowledge reuse framework for combining multiple partitions. *JMLR*, 3:583–617, 2002.

[19] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS 2006*, pages 1473–1480, 2006.

[20] X. Yin and W. S. Lee. Using link analysis to improve layout on mobile devices. In *13th WWW*, pages 338–344, 2004.