Context Browsing with Mobiles - When Less is More

Yevgen Borodin Jalal Mahmud I.V. Ramakrishnan

Department of Computer Science
Stony Brook University
Stony Brook, NY 11794, USA
{borodin, jmahmud, ram}@cs.sunysb.edu

ABSTRACT

Except for a handful of "mobile" Web sites, the Web is designed for browsing with personal computers with large screens capable of fitting the content of most Web pages. On the contrary, browsing with handhelds, such as small-screen PDA's or cell phones, usually involves a lot of horizontal and vertical scrolling, thus, making Web browsing time-consuming and strenuous. At the same time, one is often interested only in a fragment of a Web page, which again may not fit in the limited-size screens of mobile devices, requiring more scrolling in both dimensions. In this paper, we address the problem of browsing fatigue in mobile Web access using the notion of *context*. Our prototype system, CMo, reduces information overload by allowing its users to see and navigate between fragments of a Web page. On following a link, CMo captures the context of the link, employing a simple topic-boundary detection technique; it uses the context to identify relevant information in the next page with the help of a Support Vector Machine, a statistical machine-learning model; then, CMo displays the most relevant fragment of the Web page. Our experiments show that the use of context can potentially save browsing time and improve mobile browsing experience.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Search and Retrieval; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—*architectures, navigation*

General Terms

Algorithms, Design, Human Factors, Experimentation

Keywords

Context-Directed Browsing, CMo, Partitioning, Semantic Blocks, PDA, Mobile Browsing, Content Adaptation

1. INTRODUCTION

Recent years have seen a trend for miniaturization of personal computers. Handheld devices, such as PDAs and even cell phones, have long become useful tools for mobile computing. With the expansion of wireless Internet, handhelds are also gaining popularity in Web browsing applications. A number of popular Web sites now also have mobile versions. The majority of Web developers, however, are still primarily targeting personal computers with large screens, capable of fitting most Web pages. Unfortunately, a major limitation of most mobile devices is that their small screens are unable to convey the richness of the Web content.

Web browsing on mobiles incurs a number of even more serious problems. Depending on the design and layout of Web pages, they often do not fit on small screens, requiring considerable horizontal and vertical scrolling. Sometimes Web pages get deformed, as mobile Web browsers try to wrap the words and and show everything in one column while rendering the pages of the screen. On top of that, the cost and the speed of data transfer with mobile devices also have room for improvement.

At the same time, one is often interested only in a fragment of a Web page, e.g. news article or product description. For example, in Figure 1 (a), to read an *Investors.com* article on a smartphone[1], one has to wait for the page to load, locate the beginning of the article, and then continuously scroll left and right to see the end of the line, at the same time, scrolling down to the end of the article. All this makes Web browsing on mobiles a tedious and tiring exercise.

In this paper, we address the problem of browsing fatigue in mobile Web access using the notion of *context* to identify and present the most relevant information to the user, while preserving the richness of Web content. Our prototype system, CMo, reduces information overload by following the "less-is-more" design concept: CMo segments Web pages into semantic clusters of information and allows its users access and easily navigate between these segments, starting with the most relevant segment and loading them on demand, one at a time, see Figures 1 (b) and (c).

Identification of relevant information on any distinct Web page is subjective until the user selects a link. But when the link is clicked, the subject of user interest can be inferred from the link and its surrounding context. On following a link, CMo captures the context of the link, employing a simple topic-boundary detection technique. Then, the system uses the context and a Support Vector Machine (SVM), a statistical machine-learning model, to identify relevant information in the next page. Finally, CMo displays the most relevant segment of the Web page. In case when the relevant information is not identified correctly, the user is only one pen tap away from the beginning of the Web page.

[1]This and subsequent snapshots were obtained using Microsoft Visual Studio.Net PDA emulators. User evaluation was done on real PDAs.

MobiSys 2007, Jun 11–14, 2007, Puerto Rico.

(a) Regular Browsing (b) CMo Front Page (c) Browsing with CMo

Figure 1: Browsing with a Smartphone

Figure 2 shows a succession of steps used to find a new PDA on *Bizrate.com*. The user finds the "PDAs & Handheld Computers" category in the listing of product categories, shown in Figure 2 (a). When he or she follows the link, indicated by the mouse cursor, CMo uses the words of the link to find the segment of the next page that contains the desired information, see Figure 2 (b). After the PDA of interest has been found, the user follows the "PalmOne TX PDA" link in Figure 2 (b). The words of the link and its context - text around the link - are used to find a detailed description of the PalmOne PDA on the following page, which is immediately displayed on the screen without having to scroll on the page, see Figure 2 (c). If at any point CMo fails to identify the relevant information correctly, the user can click on the "FIRST" link, shown in CMo's navigation bar in Figure 2 (b), to start from the beginning of the page.

Adapting Web content for mobile browsing is an essential problem attracting many researchers. It is unreasonable to expect that all Internet content providers will supply both regular and mobile versions of their Web sites. But even if they did, why settle for stripped-down mobile Web sites and miss out on the richness of regular sites?

In this paper we present a context-directed browsing prototype for mobile devices, CMo, which brings together Content Analysis, Natural Language Processing (NLP), and Machine Learning algorithms to help mobile users quickly identify relevant information on following a link, thus, considerably reducing their browsing time. Our experiments show that the use of context can potentially save browsing time and improve mobile browsing experience.

The rest of the paper is organized as follows. In Section 2 we introduce the architecture of CMo. Section 3.1 describes a technique for partitioning Web pages, exploiting the observation that semantically-related information is often aligned in the similar fashion, for example, see how news headlines or menu items are aligned in Figure 4 (a). We explain our algorithm for context collection, based the cosine-similarity topic detection in Section 3.2, and relevant information identification algorithm based on the statistical model learned by a Support Vector Machine in Section 3.3). A through performance testing and a preliminary user evaluation is presented in Section 4. We review related research literature in Section 5, followed by our concluding remarks and future work in Section 6.

2. SYSTEM ARCHITECTURE

CMo, our context-enabled browsing system for mobile devices, takes its roots from the CSurf non-visual Web browser [21]. The CMo prototype, designed for handheld devices, is set up as a Java Servlet on the Apache Tomcat [34] server acting as a proxy between the handheld users and the Internet. The architecture of CMo is composed of the following modules: Interface Manager, Context Analyzer, Frame Tree Processor, and Browser Object (Figure 3).

Users interact with CMo through any mobile Web browser installed on their handhelds. After accessing CMo's welcome page, the user enters the address he or she want to browse and presses "GO", See Figure 1 (b). The HTTP requests are handled by the **Interface Manager**, which, after further processing, generates HTML content and returns the segments of the originally requested Web page to the mobile browser. Each page generated by CMo has a navigation bar with choices of navigating between the segments, viewing images or their ALT tags, and returning to CMo's front page to enter a new address, see Figure 1 (c) or 2 (b).

(a) Product Category (b) Product Summary (c) Product Description

Figure 2: Product Search Example

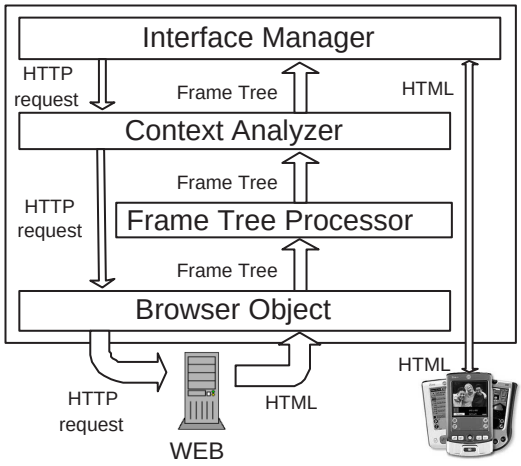


Figure 3: Architecture of CMo

Context Analyzer is called twice for each Web page access. When the user follows a link, e.g. pointed to by the arrow in Figure 2 (a) or (b), the module collects the context of the link, which includes the words of the link as well as the text around the link. When a new Web page is retrieved, the module executes our SVM-based algorithm to locate the content segment, which is estimated to be most relevant with respect to the context of the followed link. The most relevant segments are displayed on the screen of the PDA in Figures 2 (b) and (c) respectively. The context processing algorithms are described in detail in Section 3.

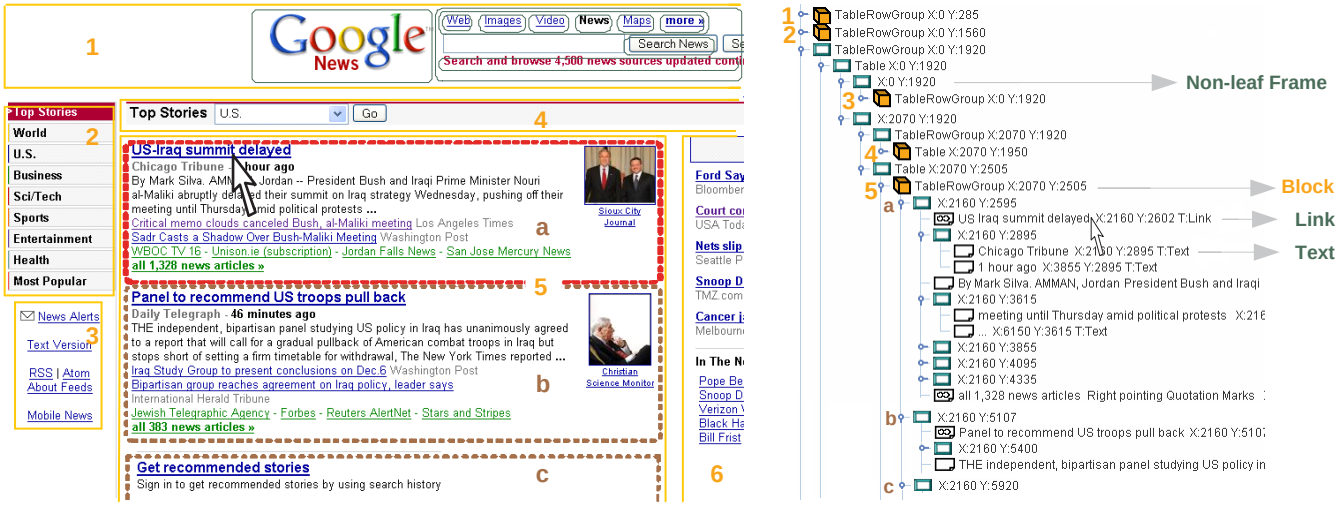
The **Browser Object** module downloads Web content every time the user requests a new Web page to be retrieved. The module is built on top of the Mozilla Web Browser [24] coupled with JREX [17] Java API wrapper. Mozilla engine takes care of all the standard browser functionalities such as support for cookies, secure connection, etc. We have extended JREX to extract a *Frame Tree*, Mozilla's internal representation of a Web page, *after* the Web page has been rendered on the screen. This way, Mozilla takes care of any dynamic content, cascading style-sheets, malformed HTML, and other rendering problems. This relieves CMo from having to deal with heavy DOM-tree objects, while giving it even more information about the content and the style.

Frame Tree Processor uses JREX API to extract the Frame Tree representation of the Web page (Figures 4 (b) and 5 (b)) from the Browser Object. We define a *Frame Tree* as a tree-like data structure that contains Web page content, along with its formatting information, which specifies how that Web page has to be rendered on the screen.

A frame tree is composed of nested *frames*[2], so that the entire page is a root frame, containing other nested frames down to the smallest individual objects on the page. For example, Figure 4 shows a snapshot of Google News front page and the corresponding frame-tree partially expanded to demonstrate the types of frames. We distinguish between the following classes of frames: text, links, images, image-links, and non-leaf frames. Section 1 of Figure 4 (a) shows several nested frames enclosed by rounded boxes. We will continue referring to any node of a frame tree as a *frame*.

Frame Tree Processor uses a number of heuristic algorithms to clean, reorganize, and partition the frame tree. The module also detects *blocks* representing semantic clusters of information in a Web page. Figure 4 shows the front page of *news.google.com* split into logical sections (blocks) on the left, and the corresponding subtrees of the frame tree, marked with 3-D block icons, on the right. Block 1 contains the title and the search bar, blocks 2 and 3 contain taxonomies, block 4 - language selector, block 5 - news headlines, etc. Subsequently, the Context Analyzer identifies the most relevant block, before passing the frame tree to the Interface Manager. Section 3 describes the corresponding algorithms in detail.

[2]not to be confused with HTML frames.



(a) Source Page (b) Source Frame Tree

Figure 4: Context Identification

Following the processing stages, the Interface Manager uses the frame tree to generate a linked-list of separate HTML Web pages, each representing a block of the original Web page. Each newly generated page has links to the first, previous, next, and last pages in the linked-list, Figure 2 (b). Subsequently, the Interface Manager returns to the mobile browser the generated HTML page that corresponds to the most relevant block of the original Web page. In case when the segment was not identified correctly, the user can simply tap on "FIRST" link to go to the block corresponding to the beginning of the original Web page.

CMo is a prototype system, and it does not yet handle all types of Web content. For example, form objects and dynamic content are not reflected on the HTML paged generated by CMo. We will next describe the techniques and algorithms of context processing in more detail.

3. CONTEXTUAL BROWSING

This section presents the core of CMo's Context Analyzer module, that drives *contextual browsing*. The two main algorithms enabling contextual browsing are *Context Identification* and *Relevant Block Identification*. Both of these algorithms utilize a *Geometrical Clustering* algorithm used by the Frame Tree Processor module to partition Web pages into segments, containing semantically related content.

To collect the context, a topic-detection algorithm is applied to the information surrounding the followed link. We gather the text that shares a common topic [1] with the link, and use this context to identify the relevant information on the destination Web page. Then, a support vector machine [35] is used to compute the relevance score of these sections with respect to the context. Subsequently, the Web page is presented to the user starting with the highest ranking section. If the relevant section was not identified correctly, the user can always skip to the beginning of the page. The following subsections discuss the algorithms in detail.

3.1 Geometric Clustering

As described in Section 2, instead of implementing its own segmentation algorithm, CMo utilizes a data structure created by the Mozilla's rendering engine. While rendering a page on the screen, Mozilla creates a tree-like structure of nested *frames* that holds the content of the Web page: the leaf frames of the tree contain the smallest individual elements, e.g. a link, an image, etc.; non-leaf frames "enclose" one or more leaf frames and/or other non-leaf frames; and the root frame "contains" the entire page. We refer to this data structure as a **frame tree**. For example, Figures 4(a) and (b) show a Web page and its corresponding frame tree.

We use an observation that semantically related information exhibits spatial locality [27, 26] and often shares the same alignment on a Web page. Since a frame tree represents the layout of a Web page, we infer that geometrical alignment of frames may imply semantic relationship between their respective content. If all descendants of a frame are consistently aligned either along X or Y axes, we call such a frame **consistent**.

A *Maximal Semantic Block*, or simply **block**, is the largest of the consistent frames on the path from a leaf to the root of a frame tree. Thus, it is likely to be the largest possible cluster containing semantically related items of information. For example, Figure 4(a) shows how we geometrically cluster the information on *news.google.com* into maximal semantic blocks: title and search bar block - labeled as 1, taxonomies - 2 and 3, Language selector - 4, and headline news - 5 and 6. Figure 4(b) shows a snapshot of the frame tree, in which the subtrees corresponding to blocks are numbered accordingly.

The *FindBlocks* algorithm is used to find the blocks in a frame tree. The algorithm runs a depth-first search over the frame tree and recursively determines whether the frames are consistent, ignoring the alignment of leaf-frames. A frame is *consistently X-aligned* if all of its non-leaf descendants are X-aligned. Similarly, a frame is *consistently Y-aligned* if all of its non-leaf descendants are Y-aligned. Otherwise, the frame is not considered to be consistent. In such case, all of its children are marked as *blocks*.

Algorithm *FindBlocks*
Input: *Frame*: node of a frame tree
Output: *Blocks*: set of maximal semantic blocks

```
1.  Identify all children C_1, C_2, ..., C_m of Frame
2.  Frame.IsConsistent ←true
3.  for j ← 1 to m
4.      do if C_j.IsLeaf = false
5.           then FindBlocks(C_j)
6.                if C_j.Alignment = NONE
7.                  then Frame.IsConsistent ←false
8.  if Frame.IsConsistent = false
9.    then for j ← 1 to m
10.            do if C_j.Alignment ≠ NONE
11.                 then Blocks ←Blocks ∪ {C_j}
12.   else Frame.Alignment ←GetAlignment(Frame)
13.        if Frame.Alignment = NONE
14.          then for j ← 1 to m
15.                 do if C_j.Alignment ≠ NONE
16.                      then Blocks ←Blocks ∪ {C_j}
17. return Blocks
```

The *FindBlocks* algorithm uses the *GetAlignment* algorithm to check whether the children of a frame have maching alignment. That is, the *GetAlignment* algorithm determines that a frame is *X-aligned* if all of its children are aligned on the left, right, or center of the X-axis. Y-alignment of a frame is computed in a similar fashion.

Algorithm *GetAlignment*
Input: *Frame*: node of a frame tree
Output: *Alignment* : alignment of *Frame*'s descendants

```
1.  Identify all children C_1, C_2, ..., C_m of Frame
2.  XFirst ←C_1.X
3.  YFirst ←C_1.Y
4.  XAlignedDescendants ←true
5.  YAlignedDescendants ←true
6.  Alignment ←NONE
7.  for j ← 2 to m
8.      do if C_j.IsLeaf = false
9.           then XCord ←C_j.X
10.               YCord ←C_j.Y
11.               if XCord ≠ XFirst
12.                 then XAlignedDescendants ←false
13.               if YCord ≠ YFirst
14.                 then YAlignedDescendants ←false
15.               if C_j.Alignment ≠ XAlign
16.                 then XAlignedDescendants ←false
17.               if C_j.Alignment ≠ YAlign
18.                 then YAlignedDescendants ←false
19. if XAlignedDescendants = true
20.   then Alignment ←XAlign
21. if YAlignedDescendants = true
22.   then Alignment ←YAlign
23. return Alignment
```

The maximal semantic blocks, obtained by the Geometric Clustering algorithm, are further used by the Context Identification and Relevant Block Identification algorithms.

3.2 Context Identification

Once the Geometric Clustering algorithm has segmented the Web page into maximal semantic blocks, and the user selected a link to be followed, the Context Identification algorithm collects the context of the link. Before we proceed to describe our algorithm in greater detail, we formally define the notion of context as:

Context of a link is the content around the link that maintain the same *topic* as the link.

Consider Figure 4, showing the front page of The Google News Web site and the corresponding frame tree. The context of the link, indicated by an arrow, is the text surrounded by the dotted line. Notice how the topic changes from one headline to another.

A block, produced by the Geometric Clustering algorithm, ideally represents a segment of text on the same subject, but may have several topics within it. Therefore, we limit topic boundary detection and context collection to the block containing the link. Context collection begins from the link and expands around the link until the topic of the text changes. A simple cosine similarity technique is used to detect the boundaries of the topic, see equation (1).

The *FindContext* algorithm initializes the *Context* multiset with the words and word combinations (bigram and trigram), excluding the function words [3], from the link and its non-link siblings; the text in the link siblings is ignored because links tend to be semantically independent of each other, i.e. have different topics. It then collects all text pertaining to the same topic around the link, adding the words to the *Context* multiset.

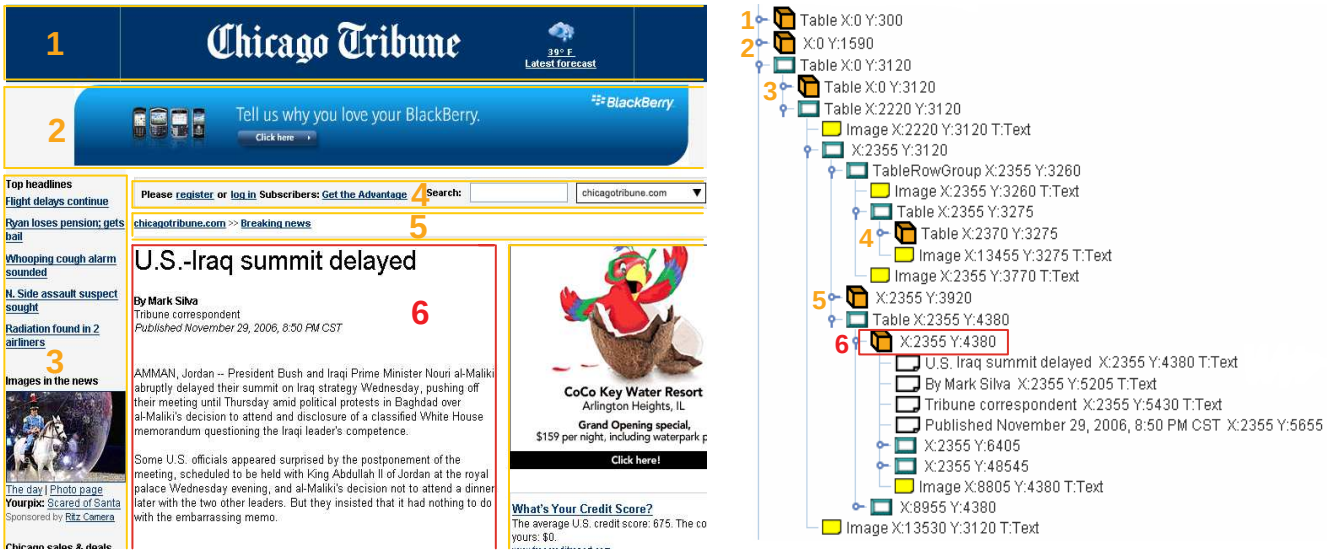
In the Google News example, Figure 4(a), the user follows the link "US-Iraq summit delayed", indicated by the mouse pointer. We initialize the multiset with the text collected from the link node of the frame tree, also indicated by a mouse pointer in Figure 4(b), as well as from the non-leaf sibling which follows the link node. The multiset now contains single words (e.g. "summit", "delayed", "president", "bush" "iraqi", "prime", "minister", etc.), their bigrams (e.g. "summit delayed", "president bush", "prime minister"), and trigrams (e.g. "iraqi prime minister").

After the initialization stage, we collect the context of the link, starting from the parent frame of the link node, by expanding the context to include the frame's siblings. We divide the siblings into the *PredList* and *SuccList*, containing the predecessor and successor siblings respectively, to expand the context window in both directions. Next, we calculate the geometric distances[4] between the initial frame and its siblings and sort the siblings accordingly.

Again, in our example, the parent frame of the link in the frame tree is the node labeled as *"a"* in Figure 4(b). The node does not have any predecessor siblings. Its successor siblings, labeled as *"b"* and *"c"*, are respectively 2512 and 3325 pixels away from frame *"a"*. Hence, we start with the sibling "b", construct multiset *SText* from the sibling's text, and compare its content to the content of the *Context* multiset. The comparison is done using cosine similarity of the multisets. More formally, for any two multisets M_1 and M_2, their cosine similarity is defined as:

[3] *Function Words* or grammatical words are words that have little lexical meaning or have ambiguous meaning, but instead serve to express grammatical relationships with other words within a sentence, or specify the attitude or mood of the speaker (Wikipedia.org)

[4] Geometric distance between two frames is the Euclidean distance between their upper-left corners on the screen.



(a) Destination Page

(b) Destination Frame Tree

Figure 5: Most Relevant Block Identification

$$Cos(M_1, M_2) = \frac{M_1 \cap M_2}{\sqrt{|M_1|}\sqrt{|M_2|}} \quad (1)$$

In the above formula, each multiset, created from a passage of text, is considered to be a vector. The cosine of the angle between the vectors is equal to 1 if the passages are identical, and 0 if they are dissimilar. We consider two multisets to be similar if their cosine similarity is above a threshold. We have statistically computed the threshold (See section 4.2 for details) that best determines whether a topic changes between the *Context* and the *SText* multisets.

If the cosine similarity between the multisets is above the threshold, i.e. topic boundary is not detected, the multisets are merged. Otherwise, we stop expanding the context window in that direction. The process continues until the *Block* boundary is reached or when there is no direction to expand. At that point, the algorithm returns the *Context* multiset as the context of the link.

Algorithm *FindContext*
Input: *LinkNode*: leaf-frame containing the link
Output: *Context*: multiset with collected context

1. *Context* ←non-function words, their bigrams and trigrams from *LinkNode* and its non-link siblings
2. Let *ancesBlock* be the ancestor *Block* of *LinkNode*
3. **if** *ancesBlock* ≠ *LinkNode.Parent*
4. **then** *Node* ←*LinkNode.Parent*
5. *Expand* ←**true**
6. **repeat**
7. *ChildList* ←*Node.Parent.Children*
8. Let *PredList* and *SuccList* be the lists of predecessors and successors of *Node* in *ChildList*, sorted by their geometric distance from *Node*
9. *StopExpand* ←**false**
10. **repeat**
11. *Sibling* ←*PredList.Next*
12. *SText* ←non-function words, their bigrams, trigrams from *Sibling*
13. *Similarity* ←*Cos*(*Context*,*SText*)
14. **if** *Similarity* > *Threshold*
15. **then** *Context* ←*Context* ∪ {*SText*}
    else *StopExpand* ←**true**
16. *Expand* ←**false**
17. **until** *PredList.IsLast* **or** *StopExpand*
18. Repeat line 9 to 17 for *SuccList*
19. *Node* ←*Node.Parent*
20. **until** *Node* = *ancesBlock* **or** *Expand* = **false**
21. **return** *Context*

Continuing with our example in Figure 4, we collect the text from the closest sibling frame *"b"*, corresponding to the news item "Panel to recommend US troops pull back". The multiset *SText*, constructed for this frame, now contains {"panel", "recommend", "us", ..., "panel recommend", ..., "panel recommend us", ...}. We compute the cosine similarity of the *Context* and a *SText* multisets, which turn out to be below our threshold. The algorithm detects a topic boundary between the content of the multisets and, therefore, stops expanding the context window and returns the *Context* multiset. The context of the followed link, Figure 4(a), is enclosed by the dotted line.

3.3 Relevant Block Identification

After the context of the link has been gathered on the source page, the Browser Object module downloads the destination Web page and generates a new frame tree. Again, we use our Geometric Clustering algorithm to segment the page into maximal semantic blocks, marked *1* through *6* in Figure 5. Then, the Relevant Block Identification algorithm matches the context against every block in the frame tree and computes the relevance of each block with respect to the collected *context*.

Intuitively, a relevant block identification algorithm should use a block ranking function to weigh the blocks and, then, pick the top-scoring block as the most relevant one. Formally, block ranking is a function which takes a vector $\vec{f}$ of

Table 1: Description of Block Features

Feature	Description
$f_{unigram}$	exact match of context words
f_{bigram}	exact bigram match (pairs of words)
$f_{trigram}$	exact trigram match (triples of words)
$f_{stemUnigram}$	match of word stems*
$f_{stemBigram}$	match of stemmed bigrams
$f_{stemTrigram}$	match of stemmed trigrams

block feature values $f_1, f_2, \ldots, f_n$ and a vector $\vec{w}$ with feature contributions $w_1, w_2, \ldots, w_n$, and returns the weight W of the block:

$$\mathbf{F} : (f_1, f_2, \ldots, f_n) \times (w_1, w_2, \ldots, w_n) \rightarrow W \qquad (2)$$

A naive approach is to manually design such function and fix the individual weights for each feature vector. For example, a function can be based on addition of feature values:

$$\mathbf{F}(\vec{f} \times \vec{w}) = w_1 \cdot f_1 + w_2 \cdot f_2 + \ldots + w_n \cdot f_n \qquad (3)$$

However, manually designing such function is not practical, justifiable, or scalable over a multitude of features. Therefore, we learn a block ranking function using a statistical learning method: we define "block ranking" as a learning problem and use a support vector machine (SVM) [35, 10], a well-known statistical model used in classification and regression analysis, to learn a block relevance model. Then, we use the machine-learned model to determine the relevance of blocks with respect to a given context. The blocks are ranked according to their relevance to the context.

As many other machine learning tools, SVM takes a feature vector as input and produces its classification (when SVM is used in classification problem). We define two classes for our block relevance model: relevant and not relevant, and describe each block of the destination page with a set of feature values, which we compute by trying to match single words, bigrams, trigrams, and their stemmed[5] counterparts, contained in context, to the text in the blocks. The features are listed in Table 1. On each successful match the corresponding feature value is incremented.

We used a freely available SVM package, libsvm [10], to learn the block relevance model. To train the SVM model, we collected and labeled training examples, where each block is represented by a tuple $(\vec{f}, l)$, where $\vec{f}$ is a feature vector for that block and l is its relevance label. To simplify the labeling, a block can be either relevant with respect to some context: $l = 1$, or not relevant: $l = 0$.

To compile labeled training data, we manually collected Web logs from about 1000 Web pages using our data-collection tool described in Section 4.1. Each Web log is a tuple of (*Source Page*, *Destination Page*), with the context and the link selected on the source page, and the most relevant block selected on the destination page. We identified the set of blocks $B_1, B_2, \ldots, B_m$ on the destination Web page, computed their feature values, and labeled the blocks as relevant (*1*) or not relevant (*0*). The training examples were, then, used to train the SVM and learn the SVM model for relevant block identification.

The machine-learned SVM model is now used to predict the relevance of a given block with respect to some context.

[5]Word stemming is done using Porter's stemmer [29]

Figure 6: Chicago Tribune News Using CMo

Given a set of blocks $B_1, B_2, \ldots, B_m$, we compute the feature values for each block by matching the context against the text in the block. Next, we use the learned SVM model to label the blocks as either relevant or not-relevant, and get the associated probability values. Then, we pick the highest ranking block, in terms of the probability values, as the most relevant one. Block 6, expanded in Figure 5(b), was chosen by the SVM as the most probable candidate for contextual relevancy. Subsequently, CMo will return section 6 to the mobile device to be displayed on the screen, see Figure 6. The other sections of the original Web page can also be browsed using CMo navigation bar at the top of the screen.

4. EVALUATION AND EXPERIMENTS

4.1 Data Collection

We used twenty five Web sites for data collection - 5 Web sites in each of 5 content domains: *news*, *books*, *consumer electronics*, *office supplies*, and *informational*. The "informational" category included various Web sites, such as LIRR[6] and Medicaid[7].

To have an efficient infrastructure for experiments, we designed a visual tool with an embedded Web browser that allowed viewing frame trees and collect training data. We manually collected *1000* (source-destination) pairs of Web pages to calculate a threshold for topic identification and to train the SVM model. During the data collection stage,

[6]http://www.mta.nyc.ny.us/lirr
[7]www.cms.hhs.gov/home/medicaid.asp

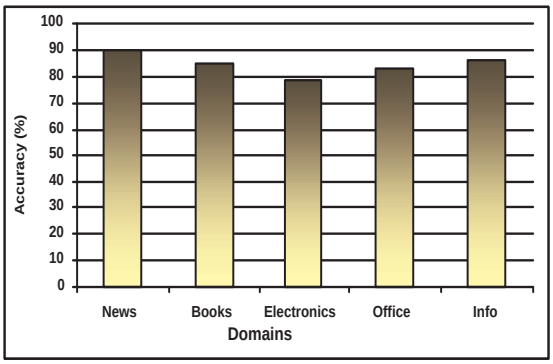


Figure 7: Accuracy of Context Identification

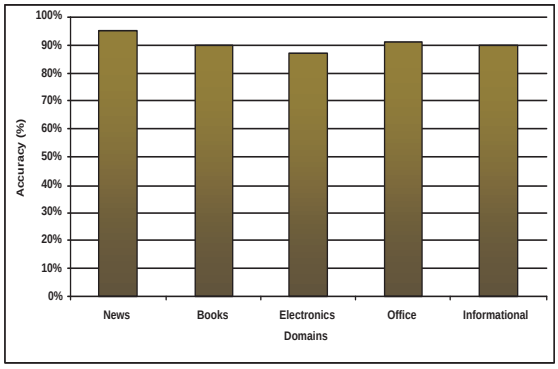


Figure 8: Accuracy of Relevant Block Identification

the experimenters were asked to select any link and the context around it on the source pages. Then, they were told to follow the link and, on the next page, select one block containing what, they thought, was the most relevant information with respect to the link they had chosen. The pairs of frame trees, corresponding to the source and the destination Web pages, were automatically saved together with user selections.

4.2 Accuracy of Context Identification

We used the collected source pages to statistically compute the accuracy and the threshold for our topic boundary detection algorithm, as described in Section 3.2. We used 50% of the page samples to estimate the threshold value and the remaining 50% were used to calculate the accuracy of topic identification algorithm.

We defined the *accuracy* of context identification as cosine similarity between the human- and computer-selected context, with the cosine similarity value ranging between 0 and 1, where 1 signified a 100% match. We used this measure for threshold estimation, as well as quantitative evaluation of the context-identification algorithm.

We designed a greedy algorithm that started with an unrealistically high threshold, used our context identification algorithm to find the context of the selected links in the 500 sample Web pages, compared the results with the human-selected context, and, then, adjusted the threshold value iteratively until it converged to the accuracy that locally could not be improved any further.

Specifically, we set the threshold $T_1 = 1$, $n = 1$, and $\delta = 0.1$; we compared the accuracies A_n and A_{n+1} while adjusting the threshold $T_{n+1} = T_n - \delta$ iteratively, as long as $A_n < A_{n+1}$. Then, we used a binary-search approach to converge to the optimal threshold T_{opt} between T_n and T_{n+1}, where the accuracy A_{opt} was the local maximum. Finally, we used the remaining 500 collected Web pages to calculate the average accuracy of context identification in each of the 5 domains. The accuracy is summarized in Figure 7.

CMo scored in the range of *80%* to *90%* accuracy compared with human-selected context. CMo scored the highest with the "News" and "Informational" Web sites; higher accuracy of context identification in these two domains can be explained by the fact that they are better organized and have more textual content. Context identification accuracy received the worst score (80%) in the "Electronics" domain, and average scores in "Books" and "Office Supplies" Web sites, most likely, because e-commerce Web sites crowd their pages with more diverse information, preferring images over text. While CMo handles *ALT* tagged images, many online stores disregard Web accessibility guidelines, making it difficult to use images as context.

4.3 Accuracy of Relevant Block Identification

The collected data was also used for SVM training and the evaluation of the algorithm's accuracy. Using the relevant block identification algorithm on each pair of the collected Web pages, we computed a feature vector for each block of the destination Web page, described in (Section 3.3). The human-selected most-relevant blocks were labeled with 1's (i.e. relevant), while the rest were labeled with 0's. We divided the training data in two sets: training (90%) and cross-validation (10%).

Once the SVM model was learned, we tested it on 100 Web page pairs of the cross-validation set to predict block labels. The labels were then compared with the human-selected ones, and the accuracy of the prediction was calculated with a hit-miss approach: the number of correctly identified blocks over the total number of blocks. The learned model showed an average of 91% accuracy in its prediction of the most relevant blocks, summarized in Figure 8.

Our algorithm showed reasonable performance in all five content domains. It is notable that our algorithm again achieved the best result of over *95%* in the news domain. The relevant information identification algorithm depends on the geometric organization of Web pages and the performance of the Geometric Clustering Algorithm; the latter performs the best on well-structured Web sites. The structural organization is often much better in News Web sites than in other domains, which explains CMo's higher performance in that domain.

The "Informational" category was not among the high performers, because informational Web sites tend to have more homogenous content and less structure, compared to the news Web sites. The relevant block identification algorithm averaged about *90%* accuracy in the e-commerce (Books, Electronics, Office Supplies) categories. The decrease of accuracy, compared to the News Web sites, was due to the ambiguity introduced by the high similarity among the different items occurring within the page. Another contributing factor was the presence of user reviews, having more word matches and, thus, scoring higher than product descriptions.

Table 2: Task Description

Task	Website	Followed Link	Relevant Information
T1	NYTimes	"Protesters Seek Leader's Ouster in Lebanon"	slogan used by protestors
T2	CNN	"Iraqi Shiite with Iran ties to visit Bush"	name of the Iraqi Shiite leader
T3	LATimes	"L.A. Fire Chief submits resignation to mayor"	name of the FireChief
T4	NewsDay	"Glavine stays with Mets"	No. of matches won by Glavine
T5	Khazana	"Books: Cambridge University Press"	the book "A Dictionary of Plant Pathology"
T6	TheatreBooks	"Opera"	the book "The Grove Book of Operas"
T7	Yahoo Shopping	"Canon Powershot SD 630 digital camera"	stores where the item is available
T8	Amazon	"Apple 4GB ipod Nano pink"	sales rank of the item

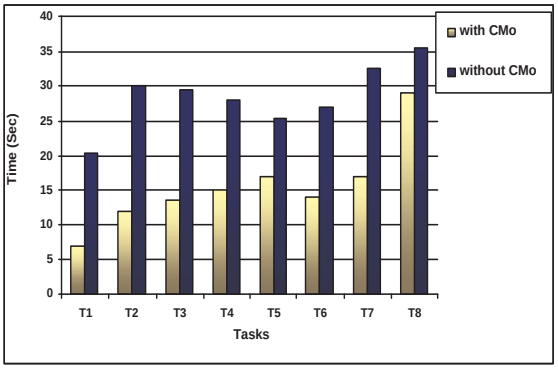


Figure 9: Time Taken with and without CMo

4.4 Browsing Efficiency Evaluation

We conducted within-the-group user-evaluations with 8 people completing 8 tasks (8 times each) on 8 Web sites. For the evaluation we used an IPAQ Pocket PC equipped with Microsoft Pocket PC operating system with wireless Internet connectivity. Prior to the experiments, the evaluators were taught the basics of using a standard mobile Web browser installed on the PDA, and were also shown how to use CMo.

The tasks and the links were preselected, so that the relevant blocks were identified correctly after following the link. When blocks are not identified correctly, the users can always proceed to the beginning of the page, in which case the time and the number of stylus taps to complete the tasks may be comparable to browsing without CMo. Thus, a much larger population sample would be required to properly report the performance comparison that includes this scenarios.

We performed the evaluation on 4 major shopping and 4 news Web sites listed in the second column of Table 2. Incidentally, most of these Web Sites had mobile versions. However, we chose to evaluate on their regular counterparts, because mobile sites already condense and adapt their content for better browsing with handhelds. In contrast, we are striving to preserve the richness of Web content by following the "less-is-more" design principle: we show less content at a time, but present the relevant information first.

The evaluators were asked to perform the tasks that would require following a link from one page to another and finding the answer to the questions listed in Table 2, columns 3 and 4 respectively. Each evaluator completed 4 tasks twice; each task was performed a total of 8 times: 4 times with CMo and 4 times without CMo. Within-the-group experiment implied that half of the users first evaluated Web browsing with CMo and then without CMo, and the other half performed the tasks first without and then with CMo. This was done to offset the effects of "site familiarity", because the same users had to repeat each task twice.

Since our goal was to provide faster access to relevant information, we measured and averaged the time taken to complete each task starting from the moment when the user followed the link and the next page finished loading to the PDA. We assessed only the user interaction time and did not measure the time taken to load the page, because the results would have been device-specific, depending on the Internet bandwidth, and also because CMo was only a prototype implementation with lots of room for speed optimization. Nevertheless, the evaluators remarked that loading a Web page with CMo took considerably less time.

Apart from the interaction time, Web browsing on handhelds usually involves hand movements, which is why we also measured the number of stylus taps that were necessary to complete the tasks. We define a *pen tap* as a screen touch with a stylus. Dragging the scroller of the browser window counted as one tap.

Figure 9 summarizes the time in seconds taken to complete the tasks $T1$ through $T8$ with and without CMo. Our system averaged 46% improvement over regular Web browsing with a PDA. Figure 10 shows a similar comparison of the number of pen taps averaging 41% improvement. Both charts show that browsing on shopping sites took more time and taps than on news Web sites. The logical explanation to this is that, for our set of tasks, there were more items in shopping Web sites, while each of the news Web pages mostly had a single news article.

In summary, our preliminary evaluation results showed promise that context-directed browsing could substantially improve mobile Web browsing and make it less tiring and time-consuming even when browsing regular non-mobile Web sites.

5. RELATED WORK

The work described in this paper has broad connections to research in content adaptation for mobile devices, Web content analysis, and context analysis.

Content Adaptation. Adapting Web content for browsing with mobile devices is an ongoing research activity. Initial efforts relied on WML (Wireless Markup Language) and WAP (Wireless Application Protocol) for designing and displaying Web pages [19, 5, 18] in mobile Web browsers. These approaches imposed additional burden on Web designers to create separate WML content. In contrast, we do not require any additional effort on the part of content providers - CMo takes the original Web pages, processes them, and presents them to the users.

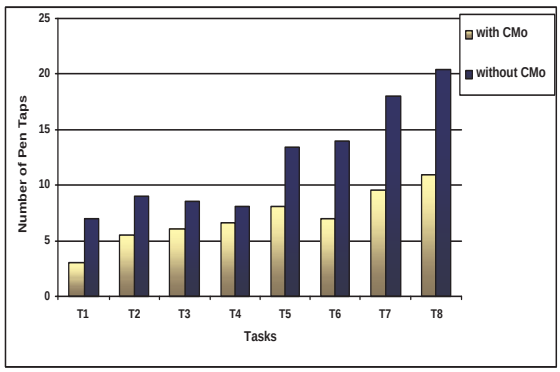


Figure 10: Pen Taps with and without CMo

Subsequent research automatically adapted regular Web content for small-screen devices. The works described in [6, 9, 8, 7, 37] focused on organizing Web pages into tree structures and summarizing their content. They were effective for ad-hoc exploratory browsing. However, summary structures often cause needless navigational steps when a user is interested in some specific content. CMo, on the other hand, allows its users to avoid unnecessary navigational steps by presenting only the relevant information that can be identified after following a link. CMo could also use summarization to condense the relevant content.

Page-splitting techniques used in content adaptation for small-screen devices are described in [11, 36]. A page-analysis technique is proposed in [11] to analyze the structure of a Web page and split it into small logically-related units that fit on the screen of a mobile device. In case when a Web page is not suitable for splitting, an auto-positioning method or scrolling-by-block is used to facilitate browsing without splitting the original Web page. In [36], a Web page that does not fit on a small screen is transformed into a set of pages, each of which fits into the screen. CMo also uses a geometric segmentation algorithm to split Web pages into sections (maximal semantic blocks), and, then, generates separate HTML pages for each of the sections of the original Web page. However, CMo goes beyond the above-mentioned systems, because it also helps its users identify the most relevant sections of Web pages.

Yin and Lee [40] proposed to construct a graph model of a Web page and, then, apply a link-analysis method, similar to Google's PageRank algorithm [4], in order to compute an importance value for each basic element of an HTML DOM tree. This allows the extraction of only the important parts of Web pages for delivery to mobile devices. In contrast, CMo uses the notion of *relevance*, rather than importance. CMo does not try to find the most important section in an individual Web page. Instead, it uses the context of the link chosen by the user in the source Web page to identify the most relevant information on the destination page.

A number of research projects have tried to condense Web pages by displaying thumbnails and summarizing Web content [3, 20, 30, 22]. For example, SmartView [22] uses a page-splitting technique to group the elements of a Web page and present them together while allowing to zoom into the individual elements. The work described in [3] segments Web pages into regions, presents each region using thumbnail, and also allows to zoom into a region by pressing a single key. In the latter work, the segmentation task is formulated as a machine learning problem based on entropy reduction and decision tree learning. Zooming and thumbnail presentation styles can facilitate the identification of relevant sections in Web pages, but they do not pinpoint the most relevant information in a page. These features, however, can further enhance CMo's usability.

Noise elimination on Web pages is described in [13, 39]. For example, Gupta et. al. [13] implemented an advertisement remover by maintaining a list of advertiser hosts and calculating the ratio of the number of words within links and in plain text. These techniques are orthogonal to our focus because CMo strives to preserve the richness of Web content, which also includes ads and banners. However, providing an option to remove banners and other commercials can further improve browsing experience for CMo users. Besides, CMo already has an option of showing images or their ALT tags.

CMo draws on the best research ideas in the area of Web content adaptation for handheld devices and further advances the field by introducing context-directed browsing for mobile devices. Other ideas, such as extracting content-rich parts of a Web page, summarizing, and presenting page segments by thumbnails can help users identify important information. However, only after the user follows a link, it may be possible to identify what information is relevant to the user. We bring in the notion of relevance to help mobile users quickly find what is important *to them* in a Web page after they followed a link. As a result of our content adaptation, we present the most relevant information first, while preserving the richness of the Web page.

Web Content Analysis. The essence of the technique underlying our context analysis algorithm is in partitioning Web pages into geometric sections. Substantial research has been done on segmenting Web documents [33, 12, 38]. These techniques are either domain [12] or site [33] specific or depend on fixed sets of HTML markups [38]. Semantic partitioning of Web pages has been described in [25, 26, 27]. These systems require semantic information (e.g. ontologies) to partition a Web page. In contrast to all of these works, our geometric clustering (partitioning) method does not depend on manually specified rules or any domain knowledge or semantic information.

Besides content adaptation, Web page partitioning techniques have also been used in content caching [31] and data cleaning [39]. VIPS [41] algorithm uses visual cues to partition a Web page into geometric segments. The algorithm extracts nodes from the DOM tree, finds vertical and horizontal separator lines between the nodes, and segments the Web page into regions based on a number of handcrafted rules. This algorithm is used in [32], where the segments are described by a set of features (e.g. spatial features, number of images, sizes, links, etc.). The feature values are then fed into an SVM, which labels the segments according to their importance.

CMo also uses an SVM to rank the blocks on the destination Web page w.r.t. the context of the link in the source page. In contrast to [32], where the SVM model was learned using features only from the content of the segment, we use the SVM to learn the block-relevance model using the feature set (See Table 1), computed from both the context of the link and the content of the block.

The fundamental difference between our research and the above-mentioned content analysis techniques is that we ex-

ploit geometrical and logical structure of Web pages both to collect context as well as to identify relevant information on the next Web page.

Contextual Analysis. The notion of context has been used in different areas of Computer Science research. For example, [16] defines context of a Web page as a collection of text, gathered around the links in other pages that are pointing to that Web page. The context is then used to obtain a summary of the page. Summarization using context is also explored by the InCommonSense system [2], where search engine results are summarized to generate text snippets. Context analysis for non-visual Web access is described in [14, 15], where context information of a link is used to get the preview of the next Web page, so that visually disabled individuals could choose whether or not they should follow the link. This idea is used in AcceSS system [28], to get the preview of the entire page. All of these works define the context of the link as an ad-hoc collection of words surrounding it. In contrast, our notion of context is based on topic similarity of words around the link. We use a principled approach for context analysis using a simple topic boundary detection method [1], confined to geometric clusters that have semantically related content.

CMo is fundamentally different from all of these works in its application. Specifically, CMo aims to help mobile users to quickly identify relevant information on following a link, thus, potentially saving browsing fatigue caused by needless scrolling and reducing users browsing time.

Our ideas for context-direct browsing were initially conceived for improving browsing efficiency in non-visual Web access. A short paper describing these preliminary ideas will appear in [21]. We have adapted these ideas for mobile Web browsing and substantially extended this preliminary work with a geometric clustering technique to identify maximal semantic segments; a topic detection [1] algorithm for context collection; an SVM block-relevance model to identify relevant information in Web pages for context-directed browsing and searching; as well as word-stemming and other NLP techniques. We developed the prototype system CMo to support context browsing in handhelds. We have also conducted preliminary performance evaluation of the browsing efficiency of CMo.

6. CONCLUSION AND FUTURE WORK

In this paper, we described the design and implementation of CMo, our context-directed Web browsing system, that acts as a proxy server and facilitates mobile browsing by logically dividing a Web page into several pages and showing the most relevant page. CMo employs our geometric clustering algorithm to segment semantically-related information in Web pages. Then, it uses various NLP and Machine Learning techniques to identify the most relevant segment to be displayed in a mobile Web browser.

In the future, it will be interesting to explore the use of other features for enhancing the performance of context-browsing. For example, by introducing more features that distinguish between parts of speech and text-formatting, the accuracy of the SVM model could be improved. Bringing advanced NLP techniques, such as summarization and WordNet ontology, to further enhance Web browsing experience is another interesting direction for research.

We are also exploring partitioning within maximal semantic blocks. Partitioning techniques can identify repeating substructures, such as news headlines, within a block of information. Giving a finer granularity and allowing to navigate between the blocks and within large blocks can further improve usability of CMo.

Currently CMo is set up to run on a proxy server handling client requests and doing all processing. In the future it should be possible to bring the context algorithms to client side. We are exploring the possibility of extending the Mozilla's Minimo [23] mobile browser with CMo functionalities to make the first standalone context-directed mobile browser.

We demonstrated the effectiveness of our algorithms by statistically computing the accuracy of the relevant block identification and doing user evaluations to compare user-interaction time and the number of stylus taps taken to find relevant information in Web pages with and without CMo.

Using our system, mobile Web surfers can potentially save their time by having to do fewer stylus taps to find and read relevant information while navigating from one Web page to another. In cases, when CMo fails to identify relevant information correctly, the users are just one stylus tap away from the beginning of the Web page. Moreover, CMo users can enjoy the full richness of Web content without having to use mobile Web sites or do excessive scrolling in regular Web sites. Thus, CMo goes beyond traditional mobile Web browsers in its ability to combat information overload and browsing fatigue in Web browsing with mobile devices.

7. ACKNOWLEDGEMENTS

We would like to thank NSF for supporting this research (Award IIS-0534419). We are grateful to Dipanjan Das for his input at the initial stages of this research, and Dr. Amanda Stent for her invaluable advice on topic detection. And, finally, we would like to extend our appreciation to our evaluators for their time and patience, and to our developers: Amogh Ranadive, Anish Jayavant, Rakesh Jawale, Abhijit Aparadh, and Dhiraj Chawla for helping us develop the CMo prototype.

8. REFERENCES

[1] J. Allen. Topic detection and tracking: Event-based information organization. Kluwer Academic Publishers, 2002.

[2] E. Amitay and C. Paris. Automatically summarising web sites - is there a way around it? In *Proceedings of the ACM 9th International Conference on Information and Knowledge Management CIKM*, pages 173–179, 2000.

[3] S. Baluja. Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 33–42, 2006.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[5] G. Buchanan, S. Farrant, M. Jones, H. Thimbleby, G. Marsden, and M. Pazzani. Improving mobile internet usability. In *Intl. World Wide Web Conf. (WWW)*, 2001.

[6] O. Buyukkoten, H. Garcia-Molina, and A. Paepcke. Focussed web searching with PDAs. In *Intl. World*

Wide Web Conf. (WWW), 2000.

[7] O. Buyukkoten, H. Garcia-Molina, and A. Paepcke. Accordion summarization for end-game browsing on pdas and cellular phones. In *ACM Conf. on Human Factors in Computing Systems (CHI)*, 2001.

[8] O. Buyukkoten, H. Garcia-Molina, and A. Paepcke. Seeing the whole in parts: Text summanrization for web browsing on handheld devices. In *Intl. World Wide Web Conf. (WWW)*, 2001.

[9] O. Buyukkoten, H. Garcia-Molina, A. Paepcke, and T. Winograd. Power browser: Efficient web browsing for pdas. In *ACM Conf. on Human Factors in Computing Systems (CHI)*, 2000.

[10] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[11] Y. Chen, W.-Y. Ma, and H.-J. Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *Intl. World Wide Web Conf. (WWW)*, 2003.

[12] D. Embley and L. Xu. Record location and reconfiguration in unstructured multiple-record web documents. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 2000.

[13] K. G. N.-D. Gupta, S. and P. Grimm. "dom based content extraction of html documents". proceedings of the 12th World Wide Web conference (WWW 2003), 2003.

[14] S. Harper, C. Goble, R. Stevens, and Y. Yesilada. Middleware to expand context and preview in hypertext. In *Assets '04: Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*, 2004.

[15] S. Harper, Y. Yesilada, C. Goble, and R. Stevens. How much is too much in a hypertext link: Investigating context and preview - a formative evaluation.

[16] B. B.-M. J.-Y. Delort and M. R. Enhanced. Enhanced web document summarization using hyperlinks. In *HYPERTEXT'03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 208–215, 2003.

[17] `http://jrex.mozdev.org/`.

[18] E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski, and T. Laakko. Two approaches to bringing internet services to wap devices. In *Intl. World Wide Web Conf. (WWW)*, 2000.

[19] A. Kaikkonen and V. Roto. Navigating in a mobile xhtml application. In *ACM Conf. on Human Factors in Computing Systems (CHI)*, 2003.

[20] B. P. Lam, H. "summary thumbnails: Readable overviews for small screen web browsers". Proceedings of CHI, 2005.

[21] J. Mahmud, Y. Borodin, D. Das, and I. Ramakrishnan. Combating information overload in non-visual web access using context. In *IUI*, 2007.

[22] N. Milic-Frayling and R. Sommerer. Smartview: Flexible viewing of web page contents. In *Intl. World Wide Web Conf. (WWW)*, 2002.

[23] `http://www.mozilla.org/projects/minimo/`.

[24] `http://www.mozilla.com/firefox/`.

[25] S. Mukherjee, I. Ramakrishnan, and A. Singh. Bootstrapping semantic annotation for content-rich html documents. In *Intl. Conf. on Data Engineering (ICDE)*, 2005.

[26] S. Mukherjee, G. Yang, and I. Ramakrishnan. Automatic annotation of content-rich html documents: Structural and semantic analysis. In *Intl. Semantic Web Conf. (ISWC)*, 2003.

[27] S. Mukherjee, G. Yang, W. Tan, and I. Ramakrishnan. Automatic discovery of semantic structures in html documents. In *Intl. Conf. on Document Analysis and Recognition*, 2003.

[28] B. Parmanto, R. Ferrydiansyah, A. Saptono, L. Song, I. W. Sugiantara, and S. Hackett. Access: accessibility through simplification & summarization. In *W4A '05: Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, pages 18–25, 2005.

[29] M. Porter. An algorithm for suffix stripping. In *Program*, pages 130–137, 1980.

[30] A. H. H.-R. Rahman, A.F.R. and K. Ariyoshi. "automatic summarization of web content to smaller display devices". 6th International Conference on Document Analysis and Recognition, 2001.

[31] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis. Automatic detection of fragments in dynamically generated web pages. In *Intl. World Wide Web Conf. (WWW)*, 2004.

[32] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for web pages. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 203–211, 2004.

[33] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Site-wide annotation: Reconstructing existing pages to be accessible. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 2002.

[34] `http://tomcat.apache.org`.

[35] V. Vapnik. Principles of risk minimization for learning theory. In *D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, Advances in Neural Information Processing Systems 3*, pages 831–838. Morgan Kaufmann, 1992.

[36] D. H. Xiangye Xiao, Qiong Luo and H. Fu. "slicing* tree based web page transformation for small displays". CIKM, 2003.

[37] C. Yang and F. L. Wang. Fractal summarization for mobile devices to access large documents on the web. In *Intl. World Wide Web Conf. (WWW)*, 2003.

[38] Y. Yang and H. Zhang. HTML page analysis based on visual cues. In *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2001.

[39] L. Yi and B. Liu. "eliminating noisy information in web pages for data mining". ACM Conf. on Knowledge Discovery and Data Mining, 2003.

[40] X. Yin and W. S. Lee. Using link analysis to improve layout on mobile devices. In *Intl. World Wide Web Conf. (WWW)*, 2004.

[41] S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma. Improving pseudo-relevance feedback in web information retrieval using web page segnmentation. In *Intl. World Wide Web Conf. (WWW)*, 2003.