Browsing on Small Displays by Transforming Web Pages into Hierarchically Structured Sub-Pages

Xiangye Xiao and Qiong Luo and Dan Hong and Hongbo Fu
Hong Kong University of Science and Technology
and
Xing Xie and Wei-Ying Ma
Microsoft Research Asia

We propose a new web page transformation method to facilitate web browsing on handheld devices such as Personal Digital Assistants (PDAs). In our approach, an original web page that does not fit on the screen is transformed into a set of sub-pages, each of which fits on the screen. This transformation is done through slicing the original page into page blocks iteratively with several factors considered. These factors include the size of the screen, the size of each page block, the number of blocks in each transformed page, the depth of the tree hierarchy that the transformed pages form, as well as the semantic coherence between blocks. We call the tree hierarchy of the transformed pages an SP-tree. In an SP-tree, an internal node consists of a textually-enhanced thumbnail image with hyperlinks, and a leaf node is a block extracted from a sub-page of the original web page. We adaptively adjust the fanout and the height of the SP-tree so that each thumbnail image is clear enough for users to read, while at the same time, the number of clicks needed to reach a leaf page is few. Through this transformation algorithm, we preserve the contextual information in the original web page and reduce scrolling. We have implemented this transformation module on a proxy server and have conducted usability studies on its performance. Our system achieved a shorter task completion time compared with that of transformations from the Opera browser in nine of ten tasks. The average improvement on familiar pages was 44%. The average improvement on unfamiliar pages was 37%. Subjective responses were positive.

Categories and Subject Descriptors: H.4.3 [**Information Systems Applications**]: Communications Applications—*Information Browsers*; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—*Navigation*

General Terms: Algorithms, Design, Human Factors

Additional Key Words and Phrases: Proxy, slicing tree, small displays, thumbnails, web page adaptation, web browsing

1. INTRODUCTION

Internet-enabled Personal Digital Assistants (PDAs) have become more and more powerful and compact. For instance, the HP iPAQ hx4700 Pocket PC features a 624 MHz processor, 128 MB ROM, 64 MB RAM, and 4' display, all contained in a $3' \times 5' \times 0.6'$ thin pad. As these devices gain increasing popularity, it is desirable to make web browsing, one of the most common activities on desktops, also convenient on these devices. However, the majority of current web sites are designed for desktop displays (see Figure 1). Viewing a web page designed for desktops becomes a heavy burden on the smaller screen of a PDA. Jones et al. [1999] indicated that small display users took much more effort and time than their large screen counterparts to perform the same navigation tasks on web pages designed for large screens. In addition to the small screen, low bandwidth and

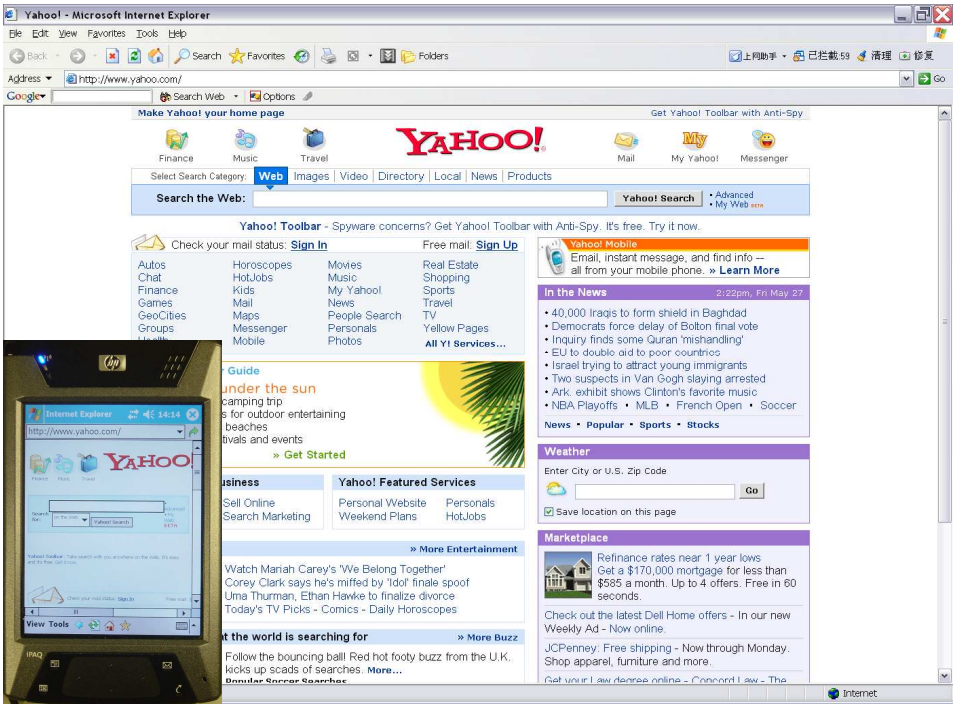

Fig. 1. Displaying the original Yahoo! home page on a desktop and a PDA.

limited input devices also make web browsing on handheld devices difficult.

One solution to this problem is to create special web pages for handheld devices manually, but manual reauthoring is tedious, and not adaptive to all web pages. Another way is to adapt the original web page designed for desktops automatically to fit on the small screen. Compared with manual reauthoring, automatic and dynamic page reauthoring is cost-effective. Therefore, automatic on-the-fly adaptation of web pages has become popular.

Currently, only a handful of browsers on PDAs (e.g., Handweb [Smartcode Software], Plamscape[ILINX], and ProxiWeb [ProxiNet]) support limited automatic web page adaptation for small displays. The performance of these browsers is limited either because PDA users have to scroll constantly with the system or because the visual information in the transformed pages is insufficient for PDA users to pick detailed information. In this article, we explore a new and efficient approach to do automatic page transformation for small displays.

In our approach, an original web page that does not fit on the screen is transformed into a set of sub-pages, each of which fits on the screen. The transformed sub-pages preserve the contextual information in the original web page and reduce scrolling. Our approach is based on the following two hypotheses: First, when PDA users access a web page that they previously accessed on desktops, they recognize the content and links of their interests in the web page based on their previous browsing experience. In particular, page layout and visual context information are

crucial for users to identify their interests in a page. Second, when PDA users access a web page that they are not familiar with, the summary of the content of each block in the page will be helpful for them to identify their interests.

Therefore, for an original web page that does not fit on a small screen, our browser first displays its textually-enhanced thumbnail image, which fits on the small screen. When a block within the image is pen-tapped, the textually-enhanced thumbnail image of this block is then displayed. Again, this thumbnail image fits on the small screen. Since we are zooming in the page, users can observe additional information on the selected block. This tap-and-zoom-in process may continue for a few more times until the target of interest is found (Figures 2 (a)-(d)). During the entire browsing process, the original page layout and context information are preserved and scrolling is seldom needed.

We use textually-enhanced thumbnails in our approach. Compared to plain thumbnails, which keep the original visual information of the web pages, textually-enhanced thumbnails draw additional textual summary of the web pages on the images [Woodruff et al. 2001]. This additional feature can help users identify interests in content. Hence, for a user, if she is familiar with the page, she can browse through the thumbnail images; otherwise, the textual summaries drawn on the images can facilitate the user to browse. For example, Figure 3 shows a plain thumbnail image and its textually-enhanced version produced by our system. Compared to the plain thumbnail image B of the original web page A, textually-enhanced thumbnail image C is enhanced by the textual summary of A, which consists of important pieces of text extracted from the web page.

There are several interesting research questions to be answered in our approach. How do we divide a large page into several smaller ones so that each of the sub-pages fits on the screen? What parts of a page are "good" to be put into one sub-page? What is the right number of sub-pages in one page, since too many sub-pages in one page is hard for users to browse, and too few sub-pages may result in too many pen-taps during navigation? Moreover, what are the effects of user browsing experience with the transformed pages in comparison with the original pages, as scrolling is nearly eliminated but the number of pen-taps increases?

In our system, we first use a vision based page segmentation algorithm to divide a large web page into small blocks. Then, we propose an algorithm to merge small blocks to form new leaf blocks given the screen size limit and organize these new leaf blocks into a tree, which we call an *SP*-tree (SP is the abbreviation of *S*ub-*P*age). In the SP-tree organization of the set of transformed pages for an original web page, each internal node is a thumbnail of the original web page (the root) or that of an intermediate sub-page, and each leaf node is a leaf page (a sufficiently small block in the original web page).

We show the SP-tree of an example web page in Figure 4. When constructing the SP-tree, we require the out-degree of the nodes in the tree to be no more than 8 and the height of the tree no more than 3, that is, $F = 8$ and $H = 3$. In the example reported in this figure, the size of the screen is 450 pixels $\times$ 300 pixels, i.e., $s_w = 450$ and $s_h = 300$. On the left of Figure 4, the leaf blocks are extracted using the SP-tree construction algorithm. They are bordered with thick lines. These leaf blocks correspond to the leaf nodes of the SP-tree. The SP-tree of the web page is

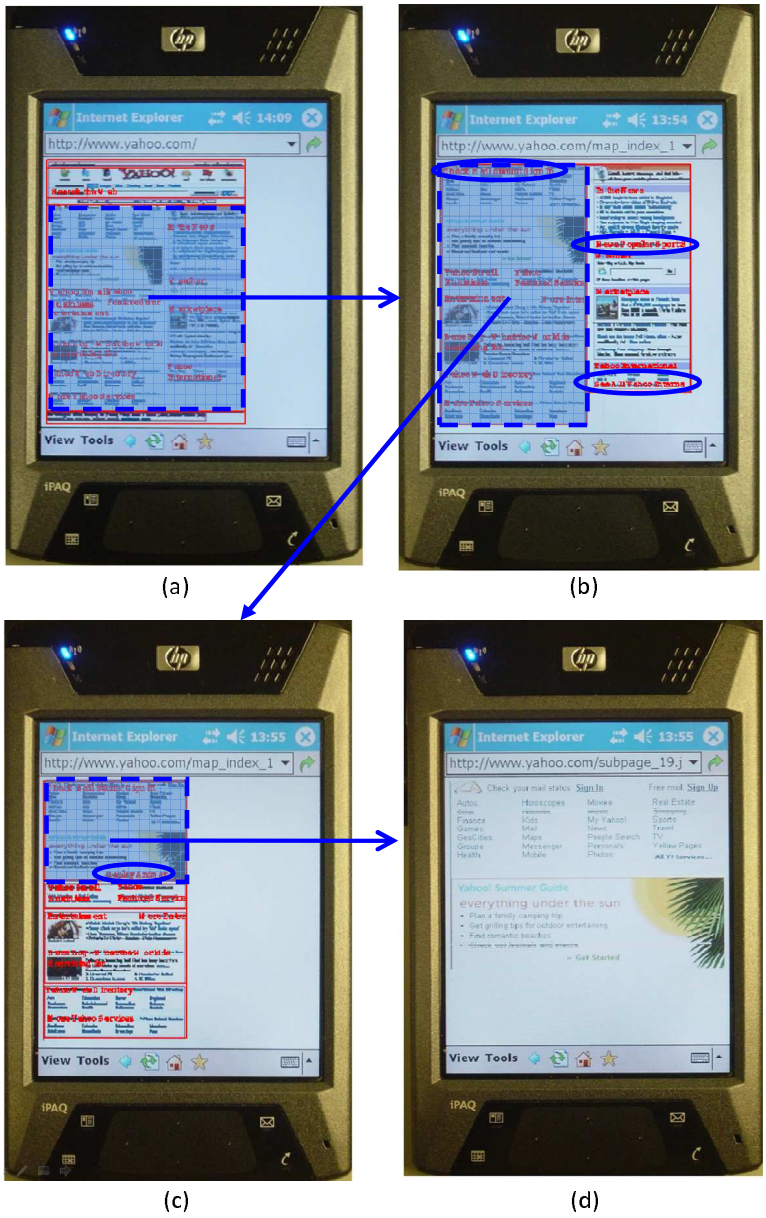

Fig. 2. Transformed pages of the original Yahoo! home page. After the user pen-taps a block in a sub-page, which is indicated by the shaded box, the corresponding sub-page of the block, which is pointed by the arrowed line, is presented to the user. The circled portions are the additional summaries that appear at the current sub-page, but not at the upper-level pages.

Fig. 3. Plain (B) and textually-enhanced thumbnail (C) images of the MSN home page (A).

shown on the right of Figure 4.

We have implemented a transformation module, which we call *SPTransform*, on a proxy server and have conducted experimental evaluation of it on PDAs. Participants in our study found that this transformation method helped them browsing web pages on PDAs.

The remainder of this article is organized as follows. In Section 2, we discuss related work. We describe the interaction paradigm of our system in Section 3. We overview the architecture of our system in Section 4. We present the SPTransform algorithm in Section 5 and experimental results in Section 6. Finally, we conclude our work in Section 7.

2. RELATED WORK

Schilit et al. [2002] classify the techniques of fitting desktop content into a small display into four categories: *scaling*, *manual authoring*, *transducing* and *transforming*. *Scaling* can reduce scrolling, but it reduces web page readability as well. *Manual authoring* is laborious since it requires professional web designers to manually tailor web pages to fit on particular devices. In contrast, automatic web page reauthoring methods, *transducing* and *transforming*, release web designers from the

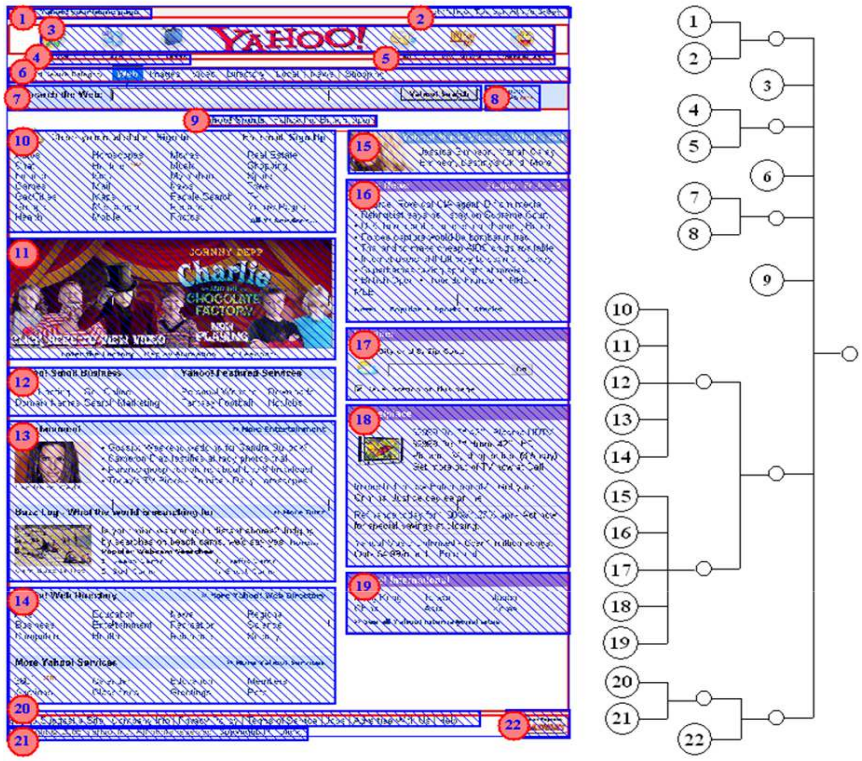

Fig. 4. The Yahoo! home page and its SP-tree, $F = 8, H = 3, s_w = 450, s_h = 300$.

heavy manual workload. *Transducing* such as AvantGo [Sybase], translates HTML and images into some formats that are supported by mobile devices so that the display of the web page matches the device characteristics, such as screen size and image resolution. *Transforming* goes further to modify both content and structure of web pages originally designed for desktop browsing to make them suitable to display on small screens. It may introduce a new organizational structure, such as a table of contents or other categories of overview pages.

The existing *transforming* methods can be further categorized into three classes according to the resulting transformed pages they provide to users: *single column*, *Fisheye*, and *overview + detail*.

The single column method adapts the multi-column layout table of a web page into a table with a single narrow column so that the horizontal scrolling can be reduced. Opera SSR is an example system that zooms out font size and image size, and transforms page layout to a single-column view for small displays. While this single-column view eliminates scrolling in the horizontal dimension, it greatly increases the amount of scrolling in the vertical dimension. Xie et al. [2005] also proposed a one-column view that displays blocks of a page in a list and places important blocks in the top of the list.

The Fisheye visualization [Furnas 1986; Gutwin and Fedak 2004] shows a focus region at a readable scale while spatially compressing page content outside the focus

region, as its name suggests. It is achieved by showing the more remote regions in successively less detail. Fishnet [Baudisch et al. 2004] is an example of a browser that shows a Fisheye view.

The overview + detail method splits a web page into multiple sections and provides an overview page with links to these sections. The overview page can be either a thumbnail image as in our work, or a text summary of the web page. The objects in an overview + detail system form a multi-level hierarchy.

The overview + detail kind of transformation methods can be further categorized into two types. One provides a textual overview and the other provides a graphical overview, i.e., thumbnail overview.

Both the Document Segmentation and Presentation System (DSPS) [Lee et al. 2003] and the Stanford Power Browser [Buyukkokten et al. 2000a; 2000b; 2001a; 2001b; 2002; Kaljuvee et al. 2001] provide textual summaries in the overview page and reveal the detailed web page content progressively. One advantage of using textual summaries is its simplicity in the display, but the downside is that the visual context (e.g., styles, images) in the original web page is lost. Consequently, users' browsing experience on desktop PCs cannot be utilized on PDAs to help discover the target information.

In contrast to textual overviews, thumbnail overviews [Baluja 2006; Baudisch et al. 2004; Björk et al. 1999;2000; L.-Q. Chen et al. 2003; Y. Chen et al. 2003; 2005; Milic-Frayling and Sommerer 2002; Rodden et al. 2003; Wobbrock et al. 2002; MacKay et al. 2004; Xie et al. 2005] preserve the visual context information of the original web pages.

Baluja [2006] first displays a web page as a thumbnail image, which is divided into 9 regions. By pressing the corresponding number keys 1-9 on a phone, users can select a region to zoom in. In addition to zooming into detail, Collapse-to-zoom [Baudisch et al. 2004] allows users to collapse areas. WEST, proposed by Björk et al. [1999;2000] provides three modes of viewing blocks of a page: a thumbnail view, a keyword view, and a link view. However, the view rendered by the original HTML source is not available. DRESS, proposed by L.-Q. Chen et al. [2003], adapts the layout of a page to generate an overview page under various constraints on display size. It allows users to enlarge the display size of a desired block on the overview page. To achieve this, a binary slicing tree based document representation was proposed. Y. Chen et al. [2003; 2005] use heuristic methods to split a web page into blocks. In their transformation method, an index page is generated, which contains a thumbnail image of the original page and hyperlinks to sub-pages corresponding to blocks. In addition to a thumbnail overview and detailed view of blocks, SmartView [Milic-Frayling and Sommerer 2002] displays number of hits on the overview page and highlights query terms within a detailed view of a block. SearchMobil [Rodden et al. 2003] extends SmartView with the aim of providing support for searching. It displays the locations of search terms on an overview of a page if it is retrieved by a search engine. WebThumb [Wobbrock et al. 2002] provides a function to pick up and zoom into an item from the thumbnail view. Gateway [MacKay et al. 2004] allows users to select blocks on a thumbnail overview of a page. These selected blocks are then expanded and superimposed on the overview. Xie et al. [2005] proposed a block importance model and three views of a page based

on the model: a two dimensional thumbnail view, a one-column view, and a main content view that only displays important blocks. The two dimensional thumbnail view is of the thumbnail overview + detail type.

Although the thumbnail overview + detail transformation method has been studied in the literature, previous work is limited to a two-level hierarchy. That is, a single thumbnail overview page linked to a set of sub-pages for detailed information. When an original web page is large, the thumbnail overview is either too large to fit on the small screen or too crowded for users to identify the blocks of interest. As a result, scrolling and pen-taps may not be effectively reduced for large pages.

Our work falls into the thumbnail overview + detail category of transformation. Instead of the two-level hierarchy of overview + detail transformed pages, we allow transformed pages to form a multi-level hierarchy. Each transformed page fits on the screen and consists of a small number of blocks. This generalization reduces input effort for users by providing sufficient but not overwhelming visual context information. Additionally, our work uses textually-enhanced thumbnails instead of the traditional plain thumbnails.

People have studied how thumbnails help users to recall their memories on previously visited web pages and how summaries help users to make correct relevance decisions [Czerwinski et al. 1999; Woodruff et al. 2001; Kaasten et al. 2002; Sweeney and Crestani 2006]. Woodruff et al. [2001] compared the efficiencies of textually-enhanced thumbnails, plain thumbnails, and textual summaries in navigating web pages to find different types of information. Their studies show that, for all categories of questions, the performance of enhanced thumbnails is comparable to textual summaries or plain thumbnails, whichever is better. Lam and Baudisch [2005] also provided text-readable thumbnail images by enlarging the font size of text in thumbnails. When zoomed in, the abbreviated text is replaced by the complete text. However, in their system, the hyperlinks and forms on the original web pages are no longer accessible on the transformed images. Sweeney and Crestani [2006] have studied the relationship between summary size and screen size. Their study indicates that, for all screen sizes, shorter summaries are more effective. Their results also show that, since the screen size of a PDA is small, a short and precise summary of a page is especially helpful for users to identify their interests in the page.

Previous study shows that a web page on small displays should present its summary information before detailed content [Jones et al. 1999]. This requirement is the driving force for page summarization. Document summarization has been studied in the literature [Luhn 1999; Mani and Maybury 1999]. Text summaries can consist of important words, sentences, or their combination. Stanford's Power Browser project [Buyukkokten et al. 2001b] presents five methods for summarizing the textual units of a web page: *Incremental*, *All*, *Keywords*, *Single-Sentence Summary*, and *Keyword/Single-Sentence Summary*. Their results shows that the combination of keywords and single-sentence summary (the *Keyword/Single-Sentence Summary* method) is the best method among the five in terms of task completion time. The Power Browser uses the well-known TF/IDF technique [Salton and McGill 1986] to find keywords and summary sentences. TF/IDF works well on full-text resources. However, web pages, especially those browsed on PDAs, contain much less text

than traditional environments. In addition, there is rich formatting information in web pages, which is useful to identify important sentences/words. Therefore, we chose to use mainly formatting information to determine summary words.

Our previous work [Xiao et al. 2005] provides a brief overview of our page transformation system. DRESS has a common problem of the two-level overview + detail transformation methods. That is, when the layout of a web page is complex, the blocks in the overview page are sometimes too crowed to be distinguishable. Additionally, the tree representation of a web page proposed in DRESS is always binary, which is different from our work. We do not use the block importance model proposed by Xie et al. [2005], because importance of blocks is sensitive to user purposes and tasks. For example, on the home page of CNN, the "Top Stories" block is important for news readers; the "Market Indexes" block is important for investors; the navigation bar block is the target of users who intend to link to a sub-site of CNN through the home page. Thus, their proposed three view modes based on the block importance model do not work for users with various purposes or interests.

3. INTERACTION PARADIGM

In this section, we describe how PDA users interact with the transformed web pages produced by our system. We use the home page of Yahoo! (http://www.yahoo.com/) as an example scenario. The original web page is shown in Figure 1. The interaction with the transformed web pages on a PDA is illustrated in Figure 2.

A user first boots a general web browser, such as MS Internet Explorer, installed in her PDA. She connects with the proxy server where our transformation system is located by changing her proxy server settings.

The user types in the URL of a web page or selects one from her bookmarks. In our example, it is http://www.yahoo.com/. Then, the requested page which has been transformed by our system is presented on the screen of her PDA, as shown in Figure 2 (a). The transformed page is a root web page. Its underlying image is a screen-fitting textually-enhanced thumbnail image of the original web page. In this image, blocks are bordered with solid lines. Each block is embedded with a hyperlink to a sub-page that provides a closer view of the block. Users can pen-tap within a bordered block to obtain the corresponding sub-page.

After the user pen-taps a block in the root page, which is indicated by the shaded box in Figure 2 (a), the corresponding sub-page of the block is presented (see Figure 2 (b)). It is an intermediate sub-page that takes a thumbnail image of an intermediate block as the underlying image. Similar to the root page, the underlying image is textually enhanced, screen fitting, and embedded with a hyperlink on each sub-block bordered with lines.

Compared to the root page or upper-level sub-pages, the view of a block is enlarged in lower-level sub-pages. For example, the block indicated in shaded box in Figure 2 (a) is enlarged in Figure 2 (b). Additionally, with the enlargement of the thumbnail view of a block, it is enhanced with more summary text. As shown in Figure 2 (b), the circled sentences are additional summary sentences that appear at the current pen-tap. They do not appear in the thumbnail image of the upper-level page, as shown in Figure 2 (a).

The user continues to pen-tap the block of her interest to get an increasingly detailed view on a lower-level transformed sub-page. If she finds that the current block is not of her interest, she goes back to previously accessed pages using the standard back button provided by the web browser.

When she taps on a bordered block that corresponds to a leaf page, such as the block indicated by the shaded box in Figure 2 (c), the leaf page extracted from the original page is displayed to her. Different from an intermediate sub-page, a leaf page provides the original appearance of a block. It is rendered from the HTML source code of the block clipped from that of the original page. Users can click on anchor text or images and submit forms as they can do on the original page.

4. SYSTEM OVERVIEW

We have implemented our page transformation system on a proxy server. In this section, we describe the system architecture and its major components: thumbnail image generation, index page generation, and leaf page generation.

4.1 System Architecture

Figure 5 illustrates the architecture of our page transformation proxy server, and shows how users' requests for web pages are processed. The proxy acts as a cache for its client PDAs. We chose to implement the page transformation module on a proxy server rather than on a specific PDA browser for two reasons. One is that a proxy server is commonly used in local area networks as firewalls or accelerators and is more powerful than PDAs. Therefore, page transformation can be done more efficiently on a proxy server than on a PDA and can be shared by multiple PDAs. The other reason is that PDA users usually have their browsers of choice and may be reluctant to switch to a new kind of browser or to accept browser extensions. Consequently, our system resides on a proxy server and transparently serves transformed pages to PDAs that go through this proxy.

As shown in Figure 5, when a PDA sends a request to a web site through our proxy, the proxy forwards the request and transforms the received web page into a tree hierarchy of thumbnail index sub-pages as internal nodes and leaf sub-pages as leaf nodes. It returns the root index page to the PDA for display and stores the other sub-pages locally. When the user selects a block in the index page, it then serves the user the corresponding sub-page that has been stored on the proxy.

The page transformation system located at the proxy consists of four components: page splitting and summarization, generation of thumbnail images, index pages, and leaf pages (Figure 6). The page splitting and summarization module builds the tree representation of the set of transformed pages, given the original web page, the screen size, the fanout and the height thresholds of the SP-tree. The textual summary of each leaf block is also extracted in this module and stored in an attribute of its corresponding leaf node. We will defer the detailed presentation on page splitting and summarization modules to Section 5 and discuss only the other three modules in this section. All of these three modules take the SP-tree, as input.

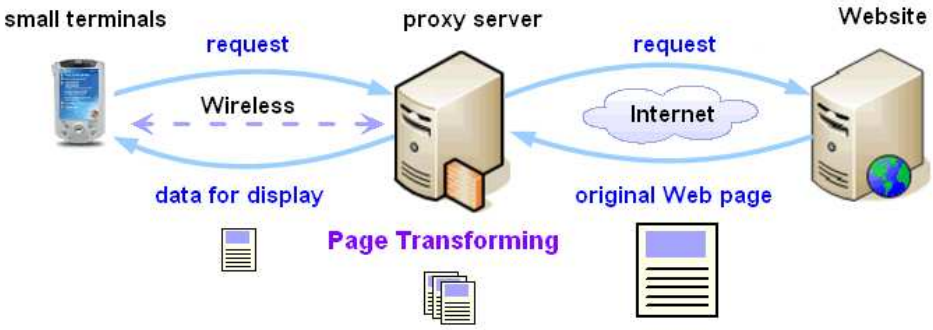

Fig. 5. Page transformation proxy server architecture.

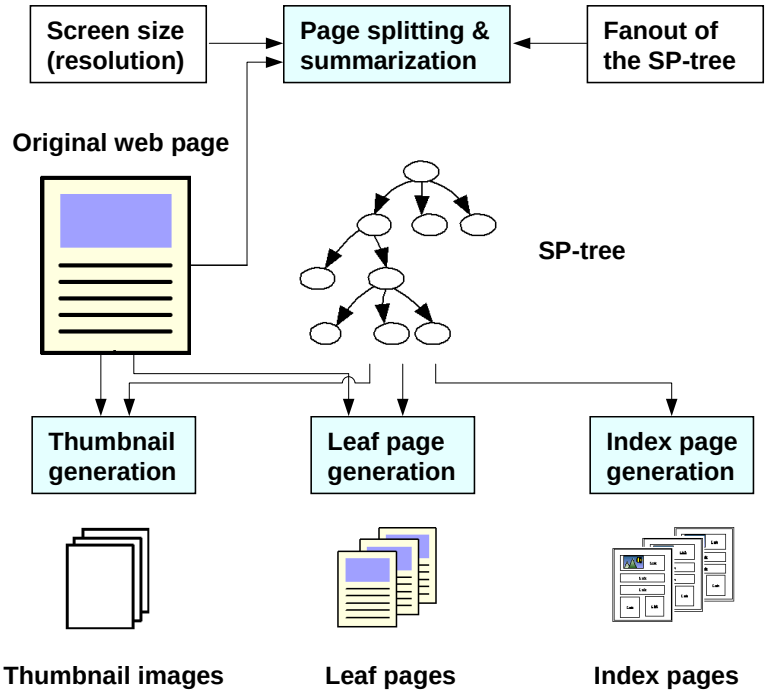

Fig. 6. Page transformation process.

4.2 Component Description

Thumbnail Image Generation. The thumbnail generation module produces the textually-enhanced thumbnails of the blocks corresponding to the internal nodes of the SP-tree through one pass of the Depth-First Search (DFS) traversal of the tree. We use DFS so that we can generate a textually enhanced thumbnail for each internal node during the traversal.

As shown in Figure 7, the procedure takes an internal node i of the SP-tree and

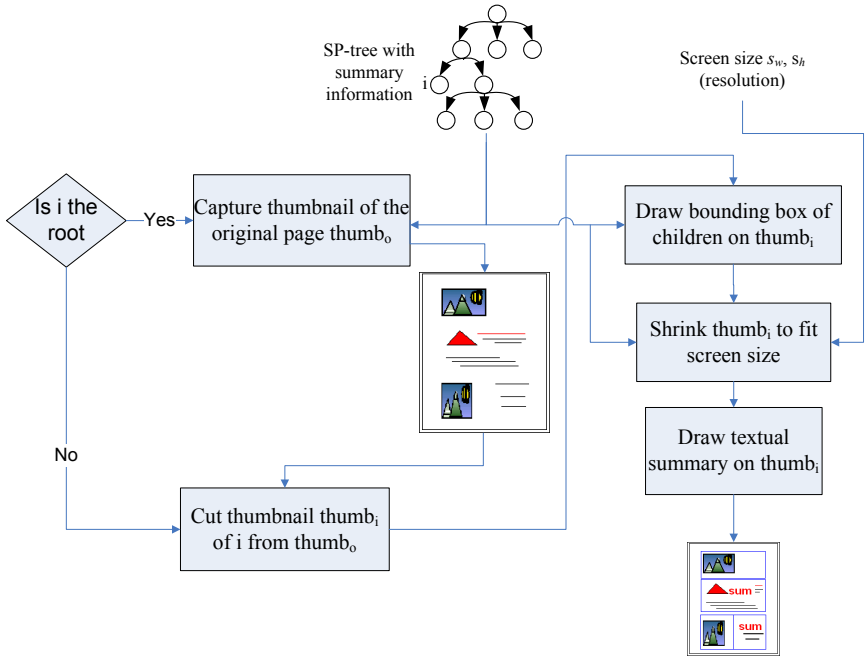

Fig. 7. Textually-enhanced thumbnail generation process for an internal node in the SP-tree.

the screen information as input and outputs the textually-enhanced thumbnail of node i's corresponding block. Firstly, we examine whether internal node i is the root or not. If i is the root of the SP-tree, we capture the image of the entire original web page, which is denoted as $thumb_o$. Otherwise, we perform image clipping to cut out the rectangular region $thumb_i$ of the block corresponding to node i from the original image $thumb_o$. Since each node in the SP-tree has location attributes (*ObjectRectTop*, *ObjectRectLeft*, *ObjectRectWidth*, and *ObjectRectHeight*) to record the position and the size of its corresponding block in the original web page, we can utilize these location attributes to cut out a block from $thumb_o$.

After that, we draw the bounding boxes of node i's children on the clipped image $thumb_i$. Then, we resize node i's image $thumb_i$ to fit on the screen. The final step is to draw textual summaries of node i's children on the resized thumbnail image. When drawing textual summaries, we constrain the drawing rectangle of a child's summary to be inside the bounding box of the child block. When the image of a block is enlarged, more summary text can be drawn. The process of generating summary sentences of blocks is described in Section 5.3.

Index Page Generation. We say that each internal node of the SP-tree corresponds to a thumbnail image, but more accurately, it corresponds to an index page, or a sub-page, with the thumbnail image embedded. This is because, in addition to the thumbnail image, the page needs to embed hyperlinks to its sub-pages at the corresponding sections of the image. We generate a hyperlink for a section based on the location attributes of the SP-tree node that corresponds to the section. The *href* values of hyperlinks are generated based on the names of the sub-pages. For

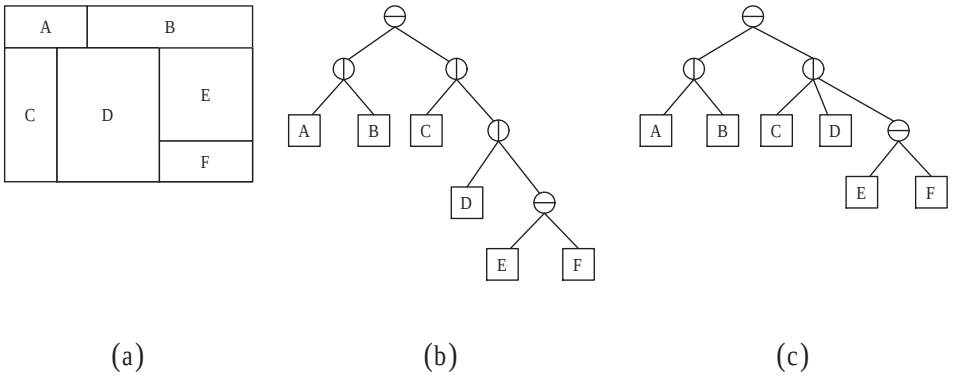

Fig. 8. (a) A slicing floorplan. (b) The binary slicing tree of the floorplan. (c) The SP-tree of the floorplan with $F = 3$.

example, we name the sub-pages in the form of *sub-page-XXX.htm*, where *XXX* is the identification number of the corresponding node of the sub-page in the SP-tree.

Leaf Page Generation. For each leaf node of the SP-tree, we generate a corresponding leaf page. The leaf page generation procedure extracts the source HTML code corresponding to the leaf node from the original HTML document. It also copies the header of the original page because Cascading Style Sheets (CSS), a common way to define style information, is usually specified in the header section of a web page. In addition, the leaf page generation procedure traces back the style information that is specified in the ancestor nodes of the leaf node in the SP-tree considering the style inheritance mechanism. Finally, it modifies the hyperlinks in the page that point to other parts of the original web page so that the updated hyperlinks point to the transformed pages correctly.

5. PAGE SPLITTING AND SUMMARIZATION

In this section, we present the page splitting and summarization module, in particular, the SP-tree construction. Text summarization is described briefly as it is not our main contribution.

5.1 SP-Tree Representation

A slicing floorplan is a decomposition of a rectangle with horizontal and vertical cuts (as shown in Figure 8 (a)). A slicing floorplan can be represented as a binary tree, called a binary *slicing* tree (e.g., Figure 8 (b)). An internal node in the slicing tree represents a cut, either in the horizontal dimension or in the vertical dimension (labeled 'h' or 'v' correspondingly). A leaf node in the slicing tree represents an atomic rectangle that has no cut through.

In comparison with the binary slicing tree, our SP-tree does not require each internal node to have exactly two children (Figure 8 (c)). Instead, it requires that each internal node has at least two children and no more than a specific number of children, which we denote as the fanout F.

Formally, an *SP*-tree is a slicing tree that satisfies the following property: for any internal node of the tree, it has at least two children and at most F children,

where $F \geq 2$.

As a web page typically consists of rectangular sections, we can naturally use an SP-tree to represent a web page decomposition scheme. Nevertheless, there are various ways of transforming a web page for the small screen given its SP-tree.

Firstly, we decide on how many levels of transformed pages will be generated out of the original page. This level may be different from the height of the SP-tree. Previous work used a two-level hierarchy, with the top level page consisting of some hyperlinks to the sub-pages and some sections from the original page. For instance, DRESS [Chen et al. 2003] studied the problem of choosing which sections from the original page to put in the top level page, even though the slicing tree they used was a binary tree of many levels. Another extreme would be to generate a deep hierarchy of transformed pages, with each level consisting of two sub-pages only. The danger of a two-level hierarchy is that there may be too many sections in the top level page so that it is hard for users to choose. In comparison, the downside of a deep hierarchy of transformed pages is that users need to pen-tap many times to navigate down to the sub-page that they are interested in. Therefore, we take the middle-of-the-road approach and adhere to the SP-tree hierarchy for transformation. Since the degree of the SP-tree is bounded by F, the depth of the hierarchy is usually shallow, but not necessarily two-level, in practice.

Secondly, for an internal node, we decide on whether we use a textual summary or a thumbnail image to represent it in the transformed sub-page. We decide to use textually-enhanced thumbnail images for the purpose of combining the advantages of both textual and graphic summaries - thumbnail images. Preserving the visual context information of the original web page helps users efficiently browse the web pages that they have previous browsing experience of on desktops. Offering the summary of each block in the web page allows users to identify the content of their interests even if they do not have previous browsing experience of the original web page.

In summary, an SP-tree represents the transformation of a web page as follows:

(1) Each internal node represents an index page, which is a thumbnail image page of a section of the original page. Each thumbnail image fits on the screen. Furthermore, the thumbnail image of a parent node embeds a hyperlink in each sub-section to point to the page of the corresponding child node.

(2) Each leaf node represents a leaf page extracted from a section in the original page. Most leaf pages also fit on the screen. A few exceptions include long text or large images. In such exceptions, if a leaf page consists of long, plain text without any html tags, only vertical scrolling is required; if it contains large images, both horizontal and vertical scrolling may be needed.

5.2 SP-tree Construction

Having described the SP-tree definition and its representation for page transformation, we proceed to present its construction.

When constructing an SP-tree representation of a web page, we should adapt the fanout and the level of the tree hierarchy so that there are a "right" number of blocks in a thumbnail image and the number of clicks to reach a leaf page is few. This adaptation is based on two observations obtained from our usability study.

One is that, as the number of sub-blocks in a thumbnail increases, it becomes more difficult for users to identify the individual sub-blocks in the image according to the visual context information. The other is that, as the height of the tree hierarchy increases, users need to perform more click actions to reach their interests and are more likely to get lost as the number of pages increases.

As shown in Algorithm 1, the SP-tree construction algorithm first divides the web page into small blocks and organizes them into a VIPS tree [Cai et al. 2003]. The leaf extraction subroutine then attempts to merge small blocks in the VIPS tree given the screen size limit. In this step, all of the leaf nodes of the SP-tree are generated. After that, the fanout-height balance procedure adapts the internal nodes of the intermediate tree output to produce a final SP-tree given fanout and height thresholds. We describe the three subroutines VIPS tree construction, leaf extraction, and fanout-height balance invoked in the SP-tree construction process in depth in the following sub-sections.

Algorithm 1 SP-TreeConstruction

Input:
$Page$ is a web page;
s_w is the screen width;
s_h is the screen height;
F is the fanout of the tree;
H is the height of the tree;
Output:
$Tree$ is the result SP-tree;
1: **begin**
2: $VIPSTree$ = VIPSTreeConstruction ($Page$);
3: $Tree$ = LeafExtraction ($VIPSTree$, s_w, s_h);
4: $Tree$ = FanoutHeightBalance ($Tree$, F, H);
5: **end**

5.2.1 *VIPS Tree Construction.* As the SP-tree nodes represent sections in a web page, we first need to identify sections or blocks in the page.

One way is to use HTML-tag-based segmentation, which utilizes the structural information provided by the HTML Document Object Model (DOM) tree. One weakness of this segmentation is that there might be too many paragraphs (segments) in a single page if the explicit paragraph delimiters (e.g., <p>, <h1>, <h2>, <table>) are considered as the start symbols of a new segment. Another weakness is that HTML-tag-based segmentation does not perform well in common cases where paragraphs do not start with <p>, or tables are used to control the layout of a page. A third weakness is that a DOM tree may be insufficient for semantically segmenting a web page. The DOM tree was initially introduced for presentation in web browsers not for description of the semantic structure of a web page. Efficient browsing of web pages on small displays necessitates semantic segmentation of web pages [Kaasinen et al. 2000].

Another way, which we use, is vision based segmentation. The VIsion-based Page Segmentation (VIPS) algorithm [Cai et al. 2003] extracts the semantic tree structure for a web page by integrating the structural information provided by the

HTML DOM tree and the visual presentation. Each node in the tree corresponds to a rectangular section, i.e., a block, and is assigned a value, Degree of Coherence (DoC), to measure the coherence of the content in the block. The larger the DoC value is, the more semantically coherent the section is. The experimental evaluation shows VIPS was correct on detecting semantically coherent blocks in 93% of the web pages [Cai et al. 2003].

As we see in Figure 9 (b), the VIPS tree of a web page has most of the properties of an SP-tree except that (1) a leaf page may be too small with respect to the screen size, that (2) the number of children of an internal node may be larger than the fanout threshold, and that (3) the height of the tree may be too big. Thus, the problem of SP-tree construction becomes first to generate the VIPS tree and then to transform it into the SP-tree by addressing these three differences.

To be conservative about the leaf page size, we generate the VIPS tree so that its leaf pages are as small as possible. This is achieved by setting the Permitted Degree of Coherence (PDoC) parameter of the VIPS algorithm to its largest value, 10 [Cai et al. 2003]. With a large PDoC value, the VIPS algorithm segments a web page at a fine granularity. Nevertheless, a leaf page may exceed the screen size in some cases, e.g., long text, or large images. In such cases, we choose to preserve the semantic coherence and not to further divide up the leaf page.

5.2.2 *Leaf Extraction.* After the VIPS tree with small leaf pages is constructed, we go through the leaf extraction step to transform the tree. The goal is to make each leaf page as large as possible within the screen size limit. This is done by merging small-sized neighboring sibling nodes in the tree. Since the following step, fanout-height balance, does not remove or insert any leaves, the leaves in the resulting tree of this step is exactly the leaves in the final SP-tree. Therefore, we name this step leaf extraction. Figure 9 (c) shows a result tree of leaf extraction.

In this step, we traverse the VIPS tree through DFS. For each internal node, if its size is smaller than the screen size, we remove all of its children because there is no need to further decompose it. Otherwise, we examine its children and see if these children can be re-partitioned to decrease the number of children and to increase the size of each new child node within the screen size limit. Algorithm 2 shows the leaf extraction process.

The subroutine *PartitionSiblingNodes* attempts to partition an array of sibling nodes into sub-arrays with the total size of the nodes in each sub-array fitting on the screen. If one of the leaf pages is larger than the screen size, it will become a single-node sub-array. If the number of nodes in a sub-array is more than one, these nodes will be merged into a new node. Note that the order of the sibling nodes in the array is unchanged throughout partitioning and that only adjacent sibling nodes can be merged so as to preserve the original appearance of the web page in terms of the spatial relationship between sections.

The problem of partitioning sibling nodes can be abstracted into the following positive number array partitioning problem:

Definition 1 *Positive number array partitioning: given an array of positive real numbers* $(e_1, e_2, e_3, \ldots, e_n)$ *and a positive real number* B *as the bound, partition the array into a number of sub-arrays* $(\ (e_1, \ldots, e_i), (e_{i+1}, \ldots, e_j), \ \ldots, \ (e_m, \ldots, e_n)\ )$ *such that the sum of the elements in each sub-array does not exceed* B*.*

Algorithm 2 LeafExtraction

Input:
$Tree$ is a VIPS tree or a sub-tree;
s_w is the screen width;
s_h is the screen height;
Output:
$Tree$ is the modified tree;

```
1: begin
2: root = Tree.root;
3: if root is an internal node then
4:    Get the width r_w and height r_h of root;
5:    if r_w > s_w or r_h > s_h then
6:       PartitionSiblingNodes(root.childnodes());
7:       for all item i ∈ root.childnodes() do
8:          LeafExtraction(i, s_w, s_h);
9:    else
10:      Remove all of the children of root;
11: end
```

If the layout of the sibling nodes to be partitioned is vertical, the bound B corresponds to s_h and the array of numbers the heights of the nodes. If the layout is horizontal, the bound B corresponds to s_w and the array of numbers corresponds to the widths of the nodes. Our goal is to merge the nodes in each sub-array so that the number of resulting nodes is the minimum. This goal translates into finding an optimal partitioning scheme for the array partitioning problem, in which the number of sub-arrays is the minimum among all possible partitioning schemes.

We developed a simple greedy algorithm (Algorithm 3) to find the optimal partitioning scheme for the array. It scans the array and keeps a running sum of the elements scanned so far. Whenever the sum becomes larger than B, it outputs the index of the current element. It then resets the sum to the current element value and continues to scan and sum. When the algorithm finishes scanning the array, it produces a partitioning scheme in the form of the indexes of the elements that serve as the boundaries between the sub-arrays.

We prove that our greedy algorithm is optimal. Let $division_g$ be the resulting array of boundaries of the greedy algorithm. The number of boundaries in $division_g$ is k. Let $division_o$ be the array of boundaries of the optimal partitioning scheme. The number of boundaries in $division_o$ is k'. Since $division_o$ is optimal, $k' \leq k$. Because both boundaries are for the same array, we have $division_g[k] = division_o[k'] = array.length()$.

We compare the two boundary arrays $division_g$ and $division_o$ from the first pair of boundaries in the two arrays. Since our algorithm is greedy, $division_g[0] \geq division_o[0]$. We then move to the second pair of boundaries. Again, $division_g[1] \geq division_o[1]$ because our algorithm is greedy. We can iteratively show that $division_g[i] \geq division_o[i]$ for $0 \leq i \leq k'$. Finally, we come to the pair of boundaries $division_g[k']$ and $division_o[k']$. We have $division_g[k'] \geq division_o[k']$ because our algorithm is greedy. Since $k' \leq k$ and the division array stores a strictly increasing sequence of boundary values, we also have $division_g[k'] \leq division_g[k]$.

Then, we have $division_g[k'] = array.length()$ because $division_g[k'] \geq division_o[k'] =$

Algorithm 3 GreedyPartition

Input:
array is an array of positive real numbers;
B is a positive real number as the bound;
Output:
division is an array of indexes that serve as the boundaries between the sub-arrays;

```
1: begin
2: i = 0;
3: while i < array.length() do
4:    sum = array[i];
5:    length = 0;
6:    while sum < B and i < array.length() do
7:       length++;
8:       i++;
9:       sum+ = array[i];
10:   if length == 0 then
11:      i++;
12:   division.append(i);
13: end
```

$array.length()$ and $division_g[k'] \leq division_g[k] = array.length()$. Then, we obtain $division_g[k'] = devision_g[k]$. Because $division_g$ is strictly increasing, we have $k' = k$.

5.2.3 ***Fanout-Height Balance***. We present the details of the fanout-height balance algorithm in Algorithm 4 $FanoutHeightBalance$.

The fanout-height balance procedure adjusts the internal structure of the tree constructed by the leaf extraction algorithm to satisfy the degree requirement of the tree, i.e., fanout. It also reduces the height of the tree as much as possible without violating the fanout threshold. It does not add or delete the leaf nodes of the input tree. Besides the input tree, the other two input parameters of this procedure are the fanout threshold F and the height threshold H.

The fanout-height balance procedure deals with four cases: both the fanout threshold F and the height threshold H are given, one of the two is given, and none of them is given. In the two cases that F is given, it invokes $InternalNodeAdaptation$ to adapt the fanout of the nodes in the input tree to be within the fanout F. As a result, the height of the tree is reduced as well. We delay the detailed description of the internal node adaptation subroutine to the next sub-section. In the case that H is given and F is not, fanout-height adaptation first assigns a default value to F. Then, it attempts to reduce the height of the tree to be within H through invoking the internal node adaption subroutine and increasing F iteratively until the tree height requirement is satisfied. In the fourth case that neither fanout nor tree height is required, fanout-height balance procedure assigns a default value to F before it invokes the internal node adaptation subroutine. We determined the default F value by experiments.

In some cases, such an SP-tree whose fanout and height both satisfy the given fanout F and height H thresholds cannot be constructed. For example, a VIPS tree in Figure 9 (b) cannot be adapted to an SP-tree whose fanout is 4 and height

Table I. Results of SP-Tree Construction in four cases of the two parameters F and H.

Fanout F	Height H	Resulting SP-tree
Given	Given	Its height may be greater than H
Given	Not given	Always satisfiable
Not given	Given	Always satisfiable
Not given	Not given	Always satisfiable

is 1, i.e., level is 2. Obviously, when $F = 4$ and $H = 1$, the maximum number of leaf nodes in the tree is 4, which is much smaller than the number of leaf nodes in the VIPS tree shown in Figure 9 (b). In such cases, our algorithm will produce a tree that satisfies F but not H.

In practice, PDA users may specify the fanout, the height, both, or neither. The four cases of the two input parameters F, H of SP-tree construction algorithm and their corresponding results are shown in Table I.

Algorithm 4 FanoutHeightBalance

Input:
$Tree$ is a tree constructed by leaf extraction;
F is the fanout of the tree;
H is the height threshold of the tree;
Output:
$Tree$ is the result SP-tree;

```
1: begin
2: if F is given then
3:    Tree = InternalNodeAdaptation (Tree, F);
4: else
5:    F = 8; // the default value
6:    Tree = InternalNodeAdaptation (Tree, F);
7:    if H is given then
8:       while Tree.height > H do
9:          F++; // since F is not specified by the user
10:         Tree = InternalNodeAdaptation (Tree, F);
11: end
```

5.2.4 ***Internal Node Adaptation.*** Internal node adaptation that is invoked by fanout-height adaptation. It adjusts the internal nodes of the tree to satisfy the node degree requirement as well as to reduce the height of the tree as much as possible.

Internal node adaptation (Algorithm 5) is done through a DFS traversal of the tree. If a node has more than F children, the algorithm combines some children by adding more levels of internal nodes between the node and these children so that it has exactly F children. If a node has fewer than F children, the algorithm attempts to increase its degree up to F. Note that decreasing the node degree to F is required for an SP-tree, but increasing the node degree to F is not. The purpose of increasing the node degree within the limit is to reduce the height of the tree.

Decreasing the node degree is done through the CombineChildren(*root*) procedure in Algorithm 5. First, it removes all children of *root*. Then, it partitions the

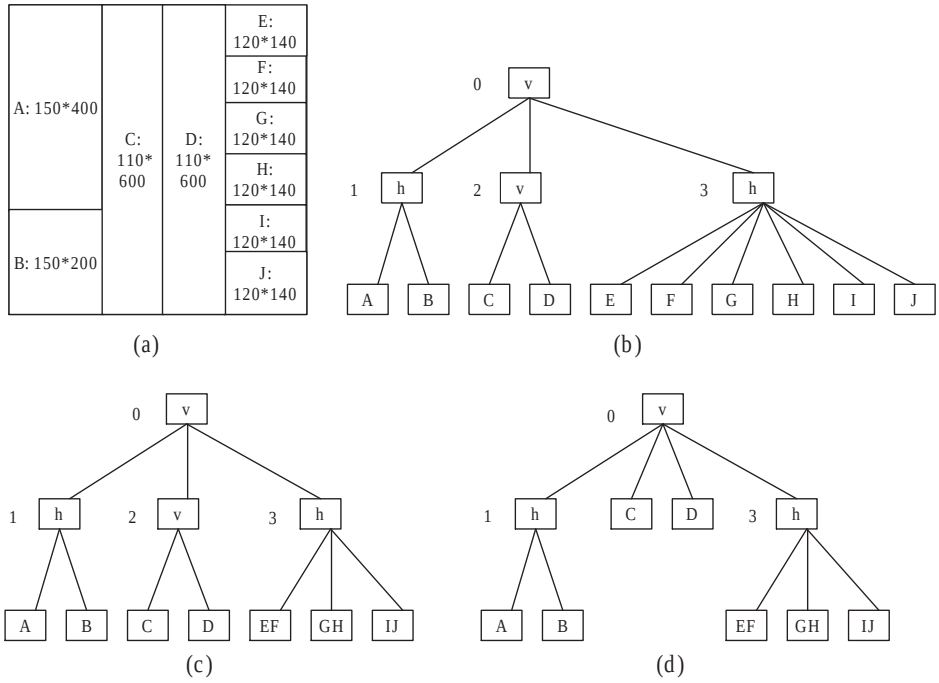

Fig. 9. (a) Floorplan of a web page. (b) VIPS tree of the web page. (c) Tree structure after leaf extraction. (d) Final SP-tree after fanout-height adaptation. ($(s_w, s_h) = (300, 300), F = 4, H = 2$).

Algorithm 5 InternalNodeAdaptation

Input:
$Tree$ is a tree constructed by leaf extraction;
F is the fanout of the tree;
Output:
$Tree$ is the modified tree;

```
1: begin
2: root = Tree.root;
3: if number of children of root > F then
4:    CombineChildren(root);
5: continue = TRUE;
6: while 0 < number of children of root < F and continue == TRUE do
7:    continue = PushUpGrandChildNodes(root);
8: for all item i ∈ root.childnodes() do
9:    InternalNodeAdaptation(i, F);
10: end
```

array of these n children to F disjoint sub-arrays with sub-arrays, except the last one, each having $\lfloor n/F \rfloor$ nodes, and the last sub-array having the remaining nodes. At this point, it begins to add F children to *root*: if a sub-array consists of more than one node, we construct a parent node for them and add the new parent node to be a child node of *root*; if a sub-array contains a single node, we directly append it as the child node of *root*. Note that this CombineChildren(*root*) procedure is different from PartitionSiblingNodes in Algorithm 2 in that the children nodes here are not partitioned based on their sizes or merged into a new node.

The node degree increase adjustment is attempted in PushUpGrandChildNodes(*root*), when a node *root* has fewer children than F. The principle of pushing up grandchild nodes is All or Nothing: either remove child node c of *root* and push up all children of c to be the children of *root*, or keep c at its place and do not push up any children of c. This principle is to preserve the semantic coherence within in a section in the original web page.

Specifically, we define a child node c to be *removable* if (1) c is an internal node that has the same label ('v' or 'h') as its parent node p, and (2) the sum of the degree of c and the degree of p minus 1 is no larger than F. If a node has multiple *removable* child node candidates, the priority of removing them is determined first by the number of descendant leaf nodes and then by the size of the node. The intuition is to reduce the height of the tree, especially the depth of the leaf nodes and that of large sections, so that the number of pen-taps can be reduced when users access the transformed pages.

PushUpGrandChildNodes(*root*) first detects all removable children of *root* and sorts them by the descending priority. If there is no removable child, the procedure will return $FALSE$. Otherwise, it will remove the child node with the highest priority, push up the node's children, and return $TRUE$.

Figure 9 (c) is the tree after leaf extraction. When this tree is input to the fanout-height balance procedure, it invokes internal node adaptation. In Figure 9 (c), node 2 has the same label as its parent node 0 and the sum of the degrees of node 2 and node 0 minus 1 is not larger than F, 4. Therefore, node 2 is a removable node. Internal node adaptation constructs the tree in Figure 9 (d), where node 2 is removed and its children C and D are pushed up. As the tree's fanout and height are already smaller than the required fanout (i.e., 4) and tree height (i.e., 2) thresholds, the fanout-height balance procedure terminates. Figure 9 (d) shows the final resulting SP-tree after fanout-height adaptation.

5.3 Text Summarization

In addition to SP-tree construction, the page splitting and summarization module also contains a text summarization subroutine that identifies important text in the page. The important text is later used to enhance thumbnails of internal sub-pages.

The text summarization subroutine first extracts all pieces of text and their visual information in a web page. A piece of text is text associated with a node in the HTML DOM tree. We extract them through a DFS traversal of the DOM tree.

The visual information of text includes three categories. The first one consists of a spatial feature, i.e., the top left position of the rectangle that the text is rendered in the web page. The second one consists of appearance features including font size and weight of text. The third category is the level of the text appeared in the DOM

tree. The higher the level is, the more important the text is.

After all pieces of text and their associated visual information are extracted, we measure the importance of text using the appearance features and the occurred levels in the DOM tree. Both categories of features are normalized into values between 0 and 10. The sum of the values is the importance score of the text.

Then, we attach important text to nodes in the SP-tree tree using the spatial features. If the top left position of a piece of text falls in the rectangle of the block corresponding to a node, the text is attached to the node. Note that if a piece of text is attached to a node, it is also attached to its parent node.

Finally, for each node in the SP-tree, we rank all pieces of text attached to it by decreasing importance scores. When enhancing thumbnails of a block, we cannot draw all important text attached to its corresponding node due to the limited thumbnail image size. How many pieces of text are actually drawn on the thumbnail of the block depends on the size of the block area in the thumbnail. We draw the top ranked text by importance scores as the summary.

6. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of the SPTransform system. First, we show both positive and negative examples through a case study. Then, we report the experimental results of the system. The experiments include objective measurements and subjective evaluation.

We compared our system with Opera [Opera Software]. Opera is a commercial mobile web browser that facilitates web page browsing for small displays by zooming out font size and image size and adapting page layout to a single column. Opera is chosen for comparison because it integrates two common techniques of facilitating web browsing on small displays. One is zooming and the other is page adaptation to reduce horizontal scrolling.

6.1 Case Study

In this subsection, we first show a positive example of our transformation result. We compare it with the transformation result of Opera. Then, the weaknesses of our system are illustrated by a negative example.

Figure 10 shows the original CNN home page displayed on a desktop, the transformed page of Opera, and the root page transformed by our system.

Figure 10(b) shows that only the navigation bar is displayed in the first view by Opera. It is a very small portion of the original page. From the result, we can see that Opera modifies the multiple-column layout of a page to a single column. This transformation changes the original page layout. Thus, users may not be able to utilize their previous browsing experience on the page. Additionally, although it reduces horizontal scrolling, vertical scrolling is inevitably increased.

Figure 10(c) shows our transformation result. The root page is a thumbnail image of the original page. The children blocks are bordered with lines. The thumbnail image is enhanced by textual summary of the page. The summary text is drawn in a bigger font size. In the CNN home page, they are “Top Stories”, “Foreigners flock back to Bethlehem”, and “More News”.

Compared to the transformation of Opera, we retain the original page layout of CNN. So, users can utilize their previous browsing experience on desktops to find

(a)

(b)

(c)

Fig. 10. A positive example. (a): the original CNN home page shown on desktop. (b): the page transformed by Opera. (c): the root page transformed by SPTransform with $F = 8$.

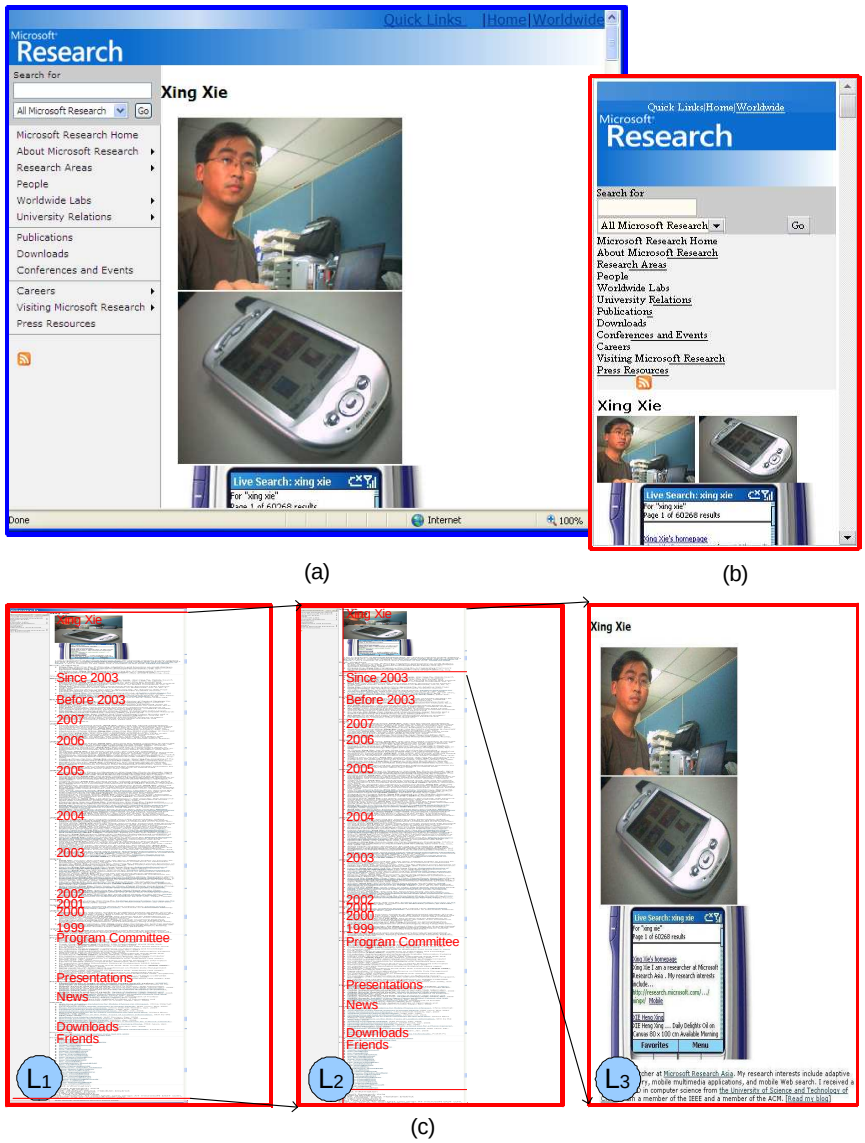

Fig. 11. A negative example. (a): the original home page of Dr. Xing Xie shown on desktop. (b): the transformed page by Opera. (c): the pages from root (L_1) to a leaf (L_3) transformed by SPTransform with $F = 4$.

their interest on PDAs. In addition, the textual summary of the page can help users identify their interests when they have not visited CNN.

We also illustrate the types of pages that our system cannot handle well through the example in Figure 11. It is Dr. Xing Xie's home page. Figure 11(a) shows the original home page of Dr. Xing Xie displayed on a desktop. Figure 11(b) shows the transformed page by Opera. Figure 11(c) shows transformed pages by

Table II. Home pages of 15 sites under study.

Page no.	URL	Page no.	URL
1	yahoo.com	9	earthlink.net
2	altavista.com	10	google.com
3	yesasia.com	11	cnn.com
4	ebay.com	12	hotmail.com
5	iwin.com	13	msn.com
6	travelocity.com	14	ask.com
7	americangreetings.com	15	lycos.com
8	bizrate.com		

our system. They are at level 1 (root), level 2, and level 3 (leaf) of the SP-tree respectively. When the middle block in the root page (denoted by L_1) is tapped, the corresponding sub-page in level 2 (denoted by L_2) is displayed. A leaf page in level 3 (denoted by L_3) is presented if the top block in page L_2 is tapped.

This negative example shows a shortcoming of our system. The shortcoming is in dealing with an extremely long page. In this situation, when fitting the screen, the generated thumbnail image is too narrow, which is not very clear to view, as shown in sub-figures L_1 and L_2 in Figure 11(c).

6.2 Impact of Fanout on SPTransform

We further study the performance of our system in more detail. We tested 15 home pages from popular web sites [Chen et al. 2003] for this study. These web sites are listed in Table II.

We test the impact of the parameter F, the threshold for the fanout of an SP-tree. We examine the page transformation time of SPTransform at the proxy for the 15 tested web pages with different F values in Figure 12.

In this figure, we see that the page transformation time drops as F increases. This is because the height of the SP-tree decreases when the maximum fanout increases. As a result, the time spent on generating thumbnail images is reduced. The improvement slows down as F becomes even larger, because it does not reduce the tree height much any more. Also, the layout of the original web page affects the transformation time. For instance, a large and complex web page such as cnn.com (Bar 11) takes the longest time and a simple one such as google.com the shortest (Bar 10). The page transformation time of google.com remains the same as F increases, because the layout of google.com is so simple that its SP-tree does not change with different F values. If we further break down page transformation time into thumbnail generation time and time spent on the other three modules (page splitting and summarization, index page generation, and leaf page generation), we can see in Figure 13 that the thumbnail image generation time dominates.

To study the characteristics of transformed pages in practice, in Table III, we list the heights of the SP-trees of the fifteen web pages with various F values, the sizes of the original web pages, and the sizes of the transformed pages. The numbers confirm that the tree height in practice is small but not necessarily two. When F increases, the tree height decreases. The total size of the transformed pages is usually a few times larger than that of the original page. However, only the pages chosen by the user will be transferred to the client device. So, our system is able to reduce the actual transmission time on the client side. We discuss this in more

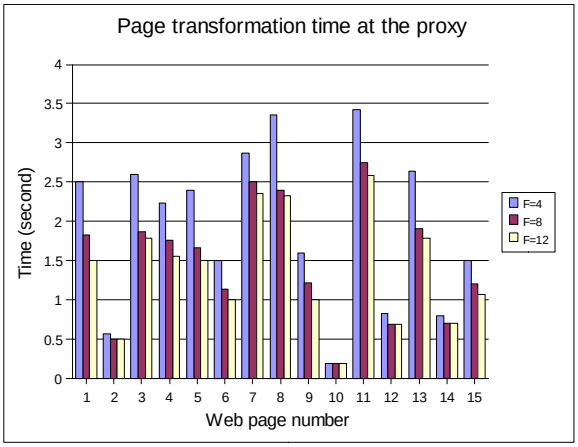

Fig. 12. Page transformation time at the proxy.

detail in Section 6.3.

Here we explain the setting of the default fanout value used in our algorithm. As discussed in this section, the height of the SP-tree increases as the fanout value decreases. When the fanout value is small, e.g., 2 or 4, the height of the SP-tree could reach 6 in some of our tested pages. This height is too large for browsing since users may pen-tap back-and-forth to find their target blocks. Furthermore, the transformation time is large for small fanout values. On the other hand, large fanout values usually result in crowded blocks in a single thumbnail, which makes the thumbnail hard to see. However, the transformation time only decreases slightly when the fanout value changes from 8 to 12 in our experiments. Hence, we found that 8 was "about right" as the fanout value in our experiments, both for user experience and the page transformation time.

6.3 Usability Study

In this sub-section, we report both objective and subjective evaluation results on the performance of our system. We compared SPTransform with Opera. Although some of the test pages in our study had mobile versions, we always used the typical desktop versions, for instance, the original web page in Figure 10, for SPTransform and Opera to transform.

6.3.1 *Experimental Setup.* We recruited ten graduate students from the Computer Science and Engineering department to be our participants. These participants are experienced web users. They are familiar with web browsing on desktop computers and also used PDAs to browse web pages.

The devices all of the participants used were HP iPAQ hx4700 Pocket PCs with a 624 MHz processor, 128 MB ROM, 64 MB RAM, and 4.0' Transflective VGA (480 × 640 pixels) TFT display, running Internet Explorer on Windows Mobile 2003. The transformation proxy server was on a PC with an Intel P4 3.20 GHz processor and 1.00 GB memory. The PDAs had a wireless connection and the proxy was on a landline local area network.

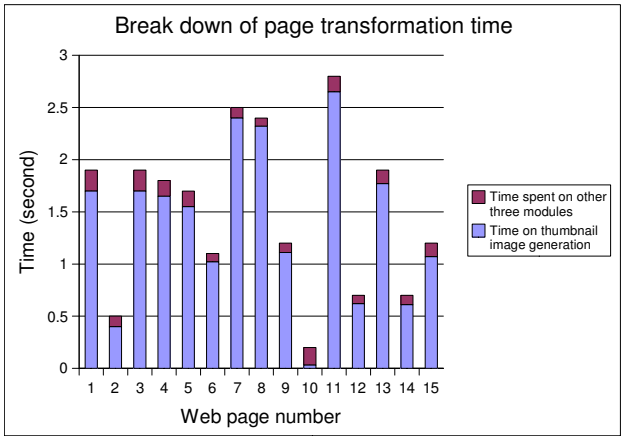

Fig. 13. Break down of page transformation time of SPTransform at the proxy.

Table III. Characteristics of each tested web page.

Page No.	Height of SP-tree			Size of original page (KByte)	Total size of transformed pages, F=8 (KByte)
	F=4	F=6	F=8		
1	3	2	2	95	336
2	3	2	2	22	71
3	4	3	3	138	391
4	6	5	4	386	340
5	5	4	3	195	472
6	3	2	2	196	348
7	4	3	2	146	374
8	5	4	4	149	216
9	4	3	2	168	77
10	2	2	2	16	237
11	5	4	4	238	770
12	2	2	2	31	80
13	4	3	3	133	448
14	4	3	2	43	75
15	3	2	2	118	362

We divided the ten participants into two groups. One group of five only browsed on PDAs connected through the SPTransform proxy. The other group of five only browsed web pages using Opera. We assigned each tester a PDA one week in advance to let them get familiar with the assigned system.

We designed ten browsing tasks for each of the ten participants to perform. The tasks and involved web pages are listed in Table IV. Among the ten tasks, all participants are familiar with the layout of involved pages in Tasks 1 to 5 and none of the participants has visited the involved pages in Tasks 6 to 10. Through this division, we distinguished the effect of previous browsing experience of participants and examined whether our system could help users browse familiar and unfamiliar pages.

We set the fanout value $F = 8$ for SPTransform to construct SP-trees of involved pages in the ten tasks. For participants of SPTransform, we required them to reach

Table IV. Ten tasks for participants to complete on PDAs.

	Involved pages	Task descriptions
Participants are familiar with the layout of involved pages		
1	http://www.msn.com/index.html	Find the Weather section.
2	http://www.yahoo.com/index.html	Find the Today's Top Searches section.
3	http://edition.cnn.com/2007/WORLD/meast/12/24/egypt.collapse.ap/index.html	Find the title of the article.
4	http://money.aol.com/news/articles/_a/merrill-lynch-selling-unit-to-ge-capital/n20071224092709990001	Find the text area to add comments on the article.
5	http://www.travelocity.com/	Find the Great Travel Deals section on the page.
Participants are unfamiliar with the layout of involved pages		
6	http://www.cse.ust.hk/~luo/me/pubs.html	Find the title of Prof Luo's ICDE publication in 2005.
7	http://research.microsoft.com/users/xingx/	Find the contact email of Dr Xie.
8	http://www.usa.canon.com/consumer/controller?act=ModelInfoAct&fcategoryid=139&modelid=15653	Find the suggested retail price of Canon EOS 40D.
9	http://tweb.acm.org/QLSpecialIssue.html	Find the important date information on the call for paper page of ACM TWEB.
10	http://www.hkex.com.hk/index.htm	Find chart of todays Hang Seng Index.

the leaf pages that the target information were located.

In our experiments, we asked the participants to each perform the ten tasks and compared the average performance of browsing with Opera or with SPTransform.

We use task completion time, input effort, and bandwidth consumption as the objective performance metrics of browsing experience. The task completion time is measured after the initial URL is entered until the user completes her task. The input effort is measured in terms of number of pen moves, including taps on the display and pulls up and down on the scroll button. The bandwidth consumption for a task is the number of bytes received at the client device during the task completion time.

We also used the Mann-Whitney U test [Siegel 1956] to analyze whether the performance difference between Opera and SPTransform is statistically significant. The Mann-Whitney U test is helpful to judge significant difference when the test size is small. We conducted the U test for task completion time, input effort, and bandwidth consumption. The significance level is set to 0.01, a commonly used value in the U test, in our experiments.

In addition to objective evaluation, we also had a subjective evaluation of our system. We designed a questionnaire for participants to answer to get their subjective opinions on the system.

Note that, since both the test web page set and the number of participants are

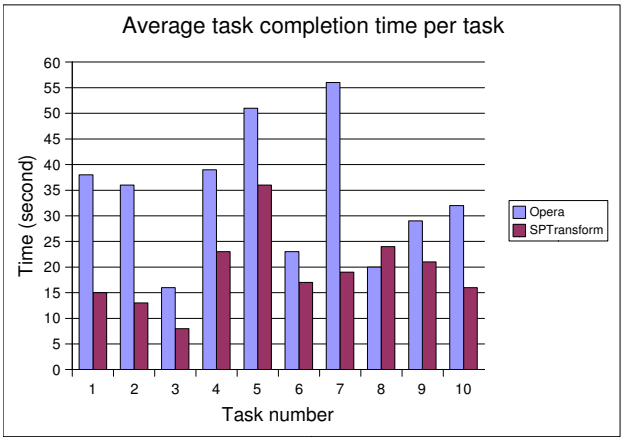

Fig. 14. Average task completion time, $F = 8$.

small in our usability study, the results reported in this section are only suggestive.

6.3.2 *Objective Evaluation on Browsing Experience.*

Average task completion time:

Figure 14 shows the average task completion time for browsing with Opera and SPTransform. Our system achieved a shorter task completion time in nine of the ten tasks. The average improvement on familiar pages in Tasks 1 to 5 was 44%. The average improvement on unfamiliar pages in Tasks 6 to 10 was 37%. Users spent more time on Task 8 using SPTransform than Opera because the suggested retail price of the camera was shown with a tiny font size and without a section name in the original web page. In addition, users were not familiar with the page. So, they had to try some sub-pages back and forth to find the target in Task 8.

We conducted a Mann-Whitney U test on task completion time. The approximate z-value = 2.61. This value shows that the task completion time improvement of SPTransform was significant.

In our experiments, our method is especially helpful in reducing the task completion time when the target in the task is an image, a chart, a controller-like textbox or other noticeable visual items. The task completion time was reduced because the time-consuming vertical and horizontal scrolling was eliminated on the transformed pages. Also, users could find their interests easily with the clear visual context.

We further divide the task completion time of SPTransform into three components: user interaction time on the client device, page transformation time at the proxy, and data transmission time on the network. The user interaction time is the time that a user spends on pen actions and content viewing. Figure 15 shows the breakdown of the task completion time in the ten tasks. The page transformation time at the proxy was only a small portion (less than 10%) of the overall task completion time.

Input effort:

Figure 16 illustrates the average number of pen moves performed for each task. Bars denoted as SPTransform are the actual number of pen moves performed by

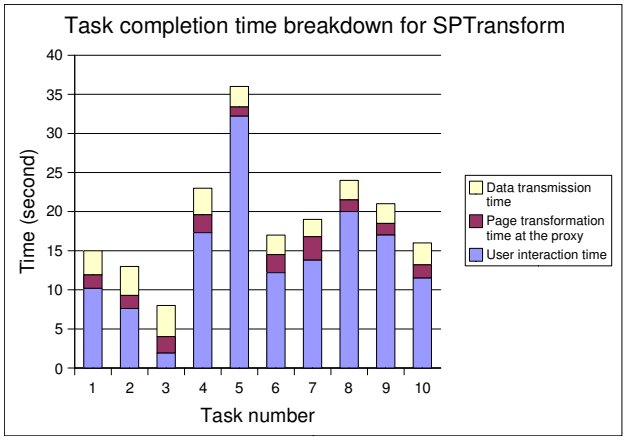

Fig. 15. Task completion time breakdown for SPTransform, $F = 8$.

participants on average. Bars denoted as SPTransform-min are the minimum number of pen moves in SPTransform if no wrong sub-page is tapped. The height of the resulting SP-tree of each involved page was not larger than 4. Since all leaf pages in our experiments happened to fit on the screen, the minimum numbers of pen moves in the ten tasks were at most 4 when they went through the correct (shortest) path along the SP-tree.

In nine out of the ten tasks, our system achieved a smaller number of actual pen moves. The improvement was 66% for Tasks 1 to 5 on familiar pages and 59% for Tasks 6 to 10 on unfamiliar pages. Similar to task completion time, the number of pen moves on Task 8 using SPTransform was larger than using Opera because the target in the task was not obvious in the thumbnails.

When the web pages transformed by SPTransform were browsed, the number of taps increased because the users might need to switch between pages by selecting links and pressing the back button. Since a pen tap is instantaneous and is easier than holding the pen to pull on the scroll bar, the input effort reduction in our system is potentially underestimated by the number of pen moves.

We conducted a Mann-Whitney U test to compare the input effort of SPTransform and Opera. The approximate z-value is 2.297. It indicates that the input effort of SPTransform is significantly less than that of Opera.

Bandwidth consumption:

When a web page is browsed using Opera, the amount of data received at the client device is the size of the web page including the HTML page transformed to single-column layout, its embedded images and style sheets. When the page is browsed using SPTransform, the amount of data received is the size of transformed pages chosen by the user including the root HTML page, pen-tapped index HTML pages, leaf HTML pages, and embedded thumbnail images in the HTML pages.

Figure 17 shows the bandwidth consumption of each task. It shows that in eight out of the ten tasks, SPTransform saved bandwidth consumption. In a Mann-Whitney U test on the bandwidth of the two systems, the approximate z-value is 1.81. It suggests that bandwidth cost of SPTransform is not significantly less than

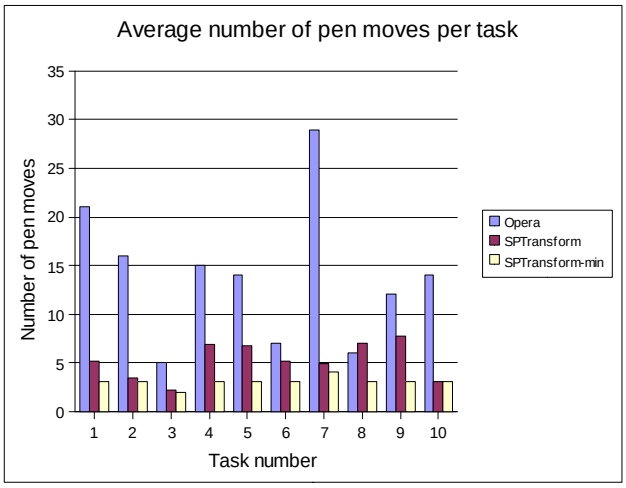

Fig. 16. Average number of pen moves, $F = 8$.

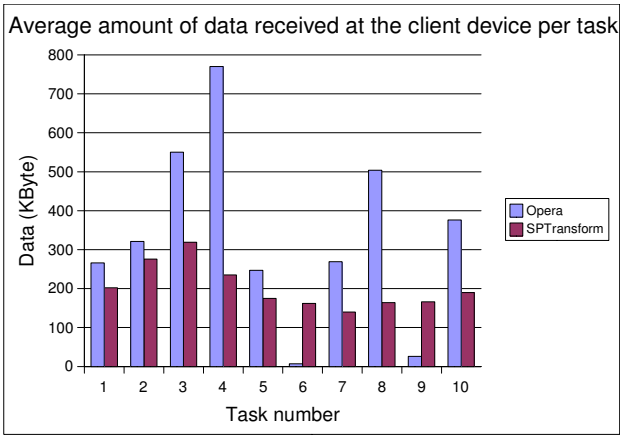

Fig. 17. Bandwidth consumption, $F = 8$.

that of Opera.

The bandwidth costs of Tasks 6 and 9 using SPTransform are larger than those using Opera. The involved pages in the two tasks, i.e., Prof Luo's publication page in Task 6 and the call for paper page of ACM TWEB in Task 9, mainly consist of text and few images. Thus, transforming them into thumbnail images increases bandwidth consumption. In comparison, the involved pages in the other eight tasks are rich in images. When using SPTransform, only selected pages are delivered. In this situation, the bandwidth consumption can be saved.

6.3.3 *Subjective Evaluation on Browsing Experience.*

In addition to objective evaluations, the ten participants were asked to finish a questionnaire after they conducted the tasks.

Table V. Four subjective questions for participants to complete after tasks, which were scored from -2 (strongly disagree) to 2 (strongly agree).

Question 1: Is web page browsing using the system enjoyable?												
	5 SPTransform users					**Total**	**5 Opera users**					**Total**
Scores	1	1	2	1	0	5	0	1	0	1	0	2

Question 2: Is it easy to find an intended object (link, section, content)?												
	5 SPTransform users					**Total**	**5 Opera users**					**Total**
Scores	2	2	2	1	0	7	0	1	0	1	0	2

Question 3: Is it easy to operate?												
	5 SPTransform users					**Total**	**5 Opera users**					**Total**
Scores	1	1	1	0	1	4	2	1	1	0	1	5

Question 4: Do you notice the speed difference between downloading pages using the assigned system and downloading using a browser without page transformation, such as Internet Browser Mobile by Microsoft Inc.?												
	5 SPTransform users					**Total**	**5 Opera users**					**Total**
Scores	-1	-1	0	-1	0	-3	-1	-1	-2	-1	1	-4

The questionnaire consists of four questions (see Table V). The users of Opera and SPTransform were asked to answer questions on Opera and SPTransform respectively. Each of the four questions was also scored from -2 (strongly disagree) to 2 (strongly agree). Table V shows the scores of the four questions given by the two groups of users.

As can be seen, SPTransform has higher scores than Opera in Questions 1 and 2. It indicates that participants found it more interesting browsing pages through the tree organized textually-enhanced thumbnails we provided than the converted single-column views. In addition, participants agreed that with SPTransform they could easily help find their targets.

The scores of SPTransform and Opera in Question 3 are positive and similar. It indicates that participants felt SPTransform and Opera were both easy to operate.

In Question 4, both groups were asked to compare the page downloading time of the assigned system to a browsing system without page transformation, such as Internet Browser Mobile by Microsoft Inc. Users gave similar scores on the two systems. The negative scores suggest that users did not notice any differences in the downloading time.

In summary, the subjective results indicate that SPTransform facilitated participants' web page browsing. At the same time, our system did not introduce noticeable downloading delays in our experiments. These subjective results are consistent with our U test results from the previous objective experiments.

7. CONCLUSION

In this article, we have proposed an SP-tree based page transformation method to facilitate web browsing on small terminals. In our approach, a web page is transformed into a set of screen-fitting thumbnail index pages and leaf pages. The thumbnail index pages and leaf pages form a multi-level tree structure with a bounded fanout. This transformation is done through slicing the original page into blocks iteratively with several factors considered, including the size of the screen, the size

of each page block, the number of blocks in each transformed page, the depth of the result hierarchy, as well as the semantic coherence within blocks. Our transformation retains the page layout of the original web page, so that users can utilize previous browsing experience on desktops for familiar web pages. Furthermore, we draw textual summary on the thumbnail images. With these textually-enhanced thumbnails, PDA users can also efficiently browse unfamiliar web pages. We have implemented our algorithm in a proxy server. In our experiments, we compared our system with Opera. Our approach improved user browsing experience in terms of task completion time and input effort with comparable bandwidth consumption.

Our current page transformation system still has some limitations. One is that we must be able to obtain the HTML source of the original web page as it is an input of our page transformation module. We need the HTML source of the original web page to build the HTML DOM tree and take this tree as input to construct the VIPS tree. We further process the VIPS tree in the leaf extraction and fanout-height balance subroutines. For web pages in sites of banks and governments, the HTML source may be unavailable.

Another limitation of our page transformation approach is that it does not handle dynamic web pages well. In the leaf page generation module, we copy the header section of the original web page to all of the transformed leaf pages. If the script functions invoked in a section of the web page are not declared in the header section, the browser will fail to find the declaration of the functions in the transformed leaf page. As a result the leaf page cannot offer to the PDA users the functions that are provided by the original web page.

REFERENCES

Baluja, S. 2006. 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*. ACM Press, New York, NY, USA, 33–42.

Baudisch, P., Lee, B., and Hanna, L. 2004. Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*. ACM, New York, NY, USA, 133–140.

Baudisch, P., Xie, X., Wang, C., and Ma, W.-Y. 2004. Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*. 91–94.

Björk, S., Holmquist, L., Redström, J., Bretan, I., Danielsson, R., Karlgren, J., and Franzén, K. 1999. West: a web browser for small terminals. In *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*. 187–196.

Björk, S., Redström, J., Ljungstrand, P., and Holmquist, L. 2000. POWERVIEW: Using Information Links and Information Views to Navigate and Visualize Information on Small Displays. *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, 46–62.

Buyukkokten, O., Garcia-Molina, H., and Paepcke, A. 2000a. Focused web searching with pdas. In *Proceedings of the 9th international World Wide Web conference on Computer networks*. North-Holland Publishing Co., 213–230.

Buyukkokten, O., Garcia-Molina, H., and Paepcke, A. 2001a. Accordion summarization for end-game browsing on pdas and cellular phones. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*. 213–220.

Buyukkokten, O., Garcia-Molina, H., and Paepcke, A. 2001b. Seeing the whole in parts: text summarization for web browsing on handheld devices. In *WWW '01: Proceedings of the tenth international conference on World Wide Web*. 652–662.

Buyukkokten, O., Garcia-Molina, H., Paepcke, A., and Winograd, T. 2000b. Power browser: efficient web browsing for pdas. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*. 430–437.

Buyukkokten, O., Kaljuvee, O., Garcia-Molina, H., Paepcke, A., and Winograd, T. 2002. Efficient web browsing on handheld devices using page and form summarization. *ACM Trans. Inf. Syst. 20,* 1, 82–115.

Cai, D., Yu, S., Wen, J., and Ma, W. 2003. Extracting content structure for web pages based on visual representation. *Proceedings of the 5th Asia Pacific Web Conference*.

Chen, L.-Q., Xie, X., Ma, W.-Y., Zhang, H.-J., Zhou, H., and Feng, H. 2003. Dress: A slicing tree based web page representation for various display sizes. In *WWW' 03: The Twelfth International World Wide Web Conference*.

Chen, Y., Ma, W.-Y., and Zhang, H.-J. 2003. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*. 225–233.

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

Czerwinski, M., van Dantzich, M., Robertson, G., and Hoffman, H. 1999. The Contribution of Thumbnail Image, Mouse-over Text and Spatial Location Memory to Web Page Retrieval. *Proceedings of Interact '99*.

Furnas, G. W. 1986. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, New York, NY, USA, 16–23.

Gutwin, C. and Fedak, C. 2004. Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques. In *GI '04: Proceedings of the 2004 conference on Graphics interface*. Canadian Human-Computer Communications Society, 145–152.

ILINX, I. Plamscape. http://www.palmscape.com/.

Jones, M., Marsden, G., Mohd-Nasir, N., Boone, K., and Buchanan, G. 1999. Improving web interaction on small displays. In *WWW '99: Proceeding of the eighth international conference on World Wide Web*. Elsevier North-Holland, Inc., 1129–1137.

Kaasinen, E., Aaltonen, M., Kolari, J., Melakoski, S., and Laakko, T. 2000. Two approaches to bringing Internet services to WAP devices. *Computer Networks 33,* 1-6, 231–246.

Kaasten, S., Greenberg, S., and Edwards, C. 2002. How People Recognize Previously Seen WWW Pages from Titles, URLs and Thumbnails. *Proceedings of Human Computer Interaction 2002 , BCS Conference Series*, 247–265.

Kaljuvee, O., Buyukkokten, O., Garcia-Molina, H., and Paepcke, A. 2001. Efficient web form entry on pdas. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*. ACM Press, New York, NY, USA, 663–672.

Lam, H. and Baudisch, P. 2005. Summary thumbnails: readable overviews for small screen web browsers. *Conference on Human Factors in Computing Systems*, 681–690.

Lee, D., Hoi, K., Xu, J., and Wang, W. 2003. Web browsing on small displays. *IEEE Distributed Systems Online 4,* 10 (October).

Luhn, H. 1999. The Automatic Creation of Literature Abstracts. *Advances in Automatic Text Summarization*.

MacKay, B., Watters, C., and Duffy, J. 2004. Web page transformation when switching devices. *Proceedings of Sixth International Conference on Human Computer Interaction with Mobile Devices and Services (Mobile HCI'04)(Glasgow, September 2004), LNCS 3160*.

Mani, I. and Maybury, M. 1999. *Advances in Automatic Text Summarization*. MIT Press.

Milic-Frayling, N. and Sommerer, R. 2002. SmartView: Flexible Viewing of Web Page Contents. *Proceedings of the 11th World Wide Web Conference (WWW 11)*.

Opera Software, I. Opera small screen rendering (ssr). http://www.opera.com/.

ProxiNet, I. Proxiweb. http://www.proxinet.com/.

Rodden, K., Milic-Frayling, N., Sommerer, R., and Blackwell, A. 2003. Effective Web Searching on Mobile Devices. *the Proceedings of the HCI Conference, Bath, September*.

Salton, G. and McGill, M. J. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc. New York, NY, USA.

Schilit, B. N., Trevor, J., Hilbert, D. M., and Koh, T. K. 2002. Web interaction using very small internet devices. *Computer 35,* 10 (October), 37–45.

Siegel, S. 1956. *Nonparametric statistics for the behavioral sciences*. McGraw-Hill, New York.

Smartcode Software, I. Handweb. http://www.smartcodesoft.com/.

Sweeney, S. and Crestani, F. 2006. Effective search results summary size and device screen size: Is there a relationship? *Information Processing and Management 42,* 4, 1056–1074.

Sybase, I. Avantgo. http://www.avantgo.com/.

Wobbrock, J. O., Forlizzi, J., Hudson, S. E., and Myers, B. A. 2002. Webthumb: interaction techniques for small-screen browsers. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*. 205–208.

Woodruff, A., Faulring, A., Rosenholtz, R., Morrsion, J., and Pirolli, P. 2001. Using thumbnails to search the web. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA, 198–205.

Xiao, X., Luo, Q., Hong, D., and Fu, H. 2005. Slicing*-tree based web page transformation for small displays. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM Press, New York, NY, USA, 303–304 (Poster).

Xie, X., Miao, G., Song, R., Wen, J.-R., and Ma, W.-Y. 2005. Efficient browsing of web search results on mobile devices based on block importance model. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*. IEEE Computer Society, Washington, DC, USA, 17–26.