

Adapting Web Pages for Small-Screen Devices

Most of the Web pages in existence today were designed for desktop PCs, which makes viewing them on mobile Web browsers extremely difficult. The authors developed new page-adaptation technique analyzes a Web page's structure and splits it into smaller, logically related units that can fit onto a mobile device's screen. The Web page can then be adapted to form a two-level hierarchy with a thumbnail representation at the top level for providing a global view and an index to a set of subpages at the bottom level for detailed information.

**Yu Chen, Xing Xie,
Wei-Ying Ma,
and Hong-Jiang Zhang**
Microsoft Research, Asia

Despite the proliferation of increasingly smaller Internet-enabled devices, their small form factors continue to constrain their usage. In trying to browse the Web on a phone, handheld computer, or personal digital assistant, you spend most of the session scrolling through the screen to find the content of interest.

Current approaches to solving this problem either introduce new formats and mechanisms to make new Web pages adaptive to different display sizes or attempt to automatically transform existing Web pages into formats that can be browsed on mobile devices. The main difference between these two approaches is that the former requires additional effort on the Web author's part.

In this article, we propose a page-adaptation technique that splits an existing Web page into smaller, logically related units. To do this, we must first solve two technical problems: how to detect an

existing Web page's semantic structure, and how to split a Web page into smaller blocks based on that structure. We've implemented our technique in different places – in Web browsers for mobile devices, in a proxy server for adapting Web pages on the fly, and as an authoring tool plug-in for converting existing Web pages.

Page Analysis

When creating a Web page, authors usually employ templates or start with a layout in mind to guide the design. Such a layout often includes headers, footers, or sidebars for the page as well as plans for how many distinct topics to incorporate in the body. Although this high-level content structure usually disappears after the page is built and sent to a client, we can sometimes recover it from embedded clues. This structure is helpful for splitting Web pages into logically related blocks.

Authors often partition content with

Related Work in Web Page Transformation

Many efforts have addressed the problem of Web browsing on small terminals, and researchers have proposed various solutions. As a start, the W3C Device Independence Activity (www.w3.org/2001/di) is standardizing the description of device characteristics and methods to assist authors in creating sites and applications that can support device independence. L. Karkkainen and J. Laarni analyzed the main properties of small devices and their influence on Web design guidelines.¹ J. Eisentein and colleagues proposed a model-based technique to develop user interfaces for multiple devices.² K.B. Lee and R.A. Grice designed a selector for handling XML information streams to extract customized information based on small-device user requests.³ A. Borning and colleagues presented a Cascading Style Sheet-compatible mechanism that lets Web page designers designate layout constraints explicitly in mathematical equalities or inequalities.⁴

Another approach is to eliminate the annoying horizontal scrolling requirement by presenting all content in a single narrow

column (with Opera Small Screen Rendering, for example; www.opera.com). Although fast and simple to implement, this method increases page height significantly and requires users to scroll up and down excessively. Y. Hwang and colleagues improved the one-column method by considering the importance of both Web components and Web page structure.⁵

Another popular approach to transforming a Web page is to partition it into a set of subpages and generate a table of contents, with or without hierarchy, to help navigation. T.W. Bickmore and colleagues proposed a reauthoring technique that required Web pages to include sections and section headers, but these designations are rarely used in Web authoring today.⁶ Work by S. Bjork and colleagues and O. Buyukkokten and colleagues focuses on text summarization.^{7,8}

References

1. L. Karkkainen and J. Laarni, "Designing for Small Display Screens," *Nordic Forum for Human-Computer Interaction Research (NordCHI 02)*, ACM Press, 2002, pp. 227–230.
2. J. Eisentein, J. Vanderdonck, and A. Puerta, "Applying Model-Based Techniques to the Development of UIs for Mobile Computers," *Proc. Int'l Conf. Intelligent User Interfaces (IUI 01)*, ACM Press, 2001, pp. 69–76.
3. K.B. Lee and R.A. Grice, "An Adaptive Viewing Application for the Web on Personal Digital Assistants," *Proc. 21st Ann. Int'l Conf. Documentation (SIGDOC 03)*, ACM Press, 2003, pp. 123–132.
4. A. Borning, R.K. Lin, and K. Marriott, "Constraint-Based Document Layout for the Web," *ACM Multimedia Systems J.*, vol. 8, no. 3, 2000, pp. 177–189.
5. Y. Hwang, J. Kim, and E. Seo, "Structure-Aware Web Transcoding for Mobile Devices," *IEEE Internet Computing*, vol. 7, no. 5, 2003, pp. 14–21.
6. T.W. Bickmore and B.N. Schilit, "Digester: Device-Independent Access to the World Wide Web," *Proc. 6th Int'l World Wide Web Conf. (WWW 97)*, Elsevier, 1997, pp. 1075–1082.
7. S. Björk et al., "WEST: A Web Browser for Small Terminals," *Proc. 12th Ann. ACM Symp. User Interface Software and Technology (UIST 99)*, ACM Press, 1999, pp. 187–196.
8. O. Buyukkokten et al., "Efficient Web Browsing on Handheld Devices Using Page and Form Summarization," *ACM Trans. Information Systems*, vol. 20, no. 1, 2002, pp. 82–115.

one of two types of visual separators: HTML tags create *explicit* separators, and blank spaces between content in the Web page create *implicit* separators. These separators are also helpful for page splitting.

Our page-analysis algorithm consists of three steps. First, we analyze the HTML Document Object Model (DOM) tree and detect the high-level content blocks (which contain location and size information for headers, footers, sidebars, and the body). Then, we analyze the content inside each high-level content block to identify explicit separators to determine where to split the blocks. Finally, we detect implicit separators to help split the blocks further. The overall goal for page analysis is to help split Web pages into appropriate blocks so that users can browse page blocks on small devices instead of scrolling and browsing through large Web page.

High-Level Content Block Detection

An HTML parser generates the page's HTML DOM

tree and identifies the position and dimension information for each node in the tree.

Our algorithm traverses the DOM tree from `<body>` to leaves to select appropriate nodes and put them into a high-level content block. The algorithm decides whether to keep each node as a unit or move one level down to consider its child nodes as units. If a node is too large in display size, compared with the screen size, keeping it as a whole block will produce erroneous results. On the Yahoo! home page (www.yahoo.com), for example, the `<body>` contains a single child node `<center>`, which aligns all the content to the middle of the page. If `<center>` were kept as a whole, the algorithm wouldn't be able to detect the header or footer.

We then try to classify a node into header, footer, left sidebar, or right sidebar blocks according to common Web page designs. If it doesn't belong to any of these categories, we check to see if it's small enough to put into the body block. We use a pair of thresholds — one for width and one for height

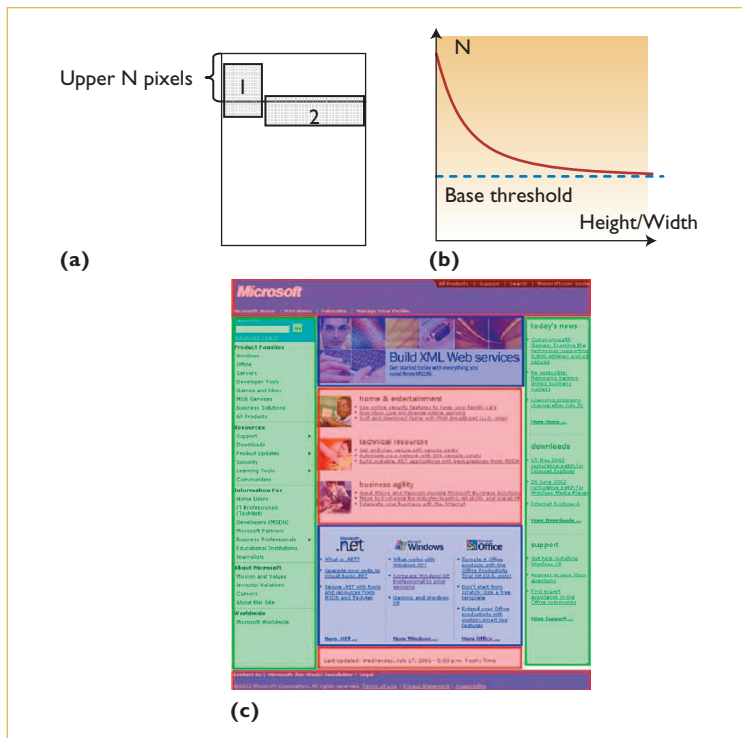


Figure 1. Difficulties in detecting headers. In our approach, the height/width ratio of (a) a node region's shape gives the dynamic threshold for detecting (b) headers and footers. The result of high-level content block detection on (c) Microsoft's home page (www.microsoft.com) shows different high-level content blocks marked with different colors.

– to determine whether the node is small enough in its display size; if it exceeds these thresholds, we split it further.

Because the header block is at the top of a Web page, we can define a threshold N and let the upper N pixels of the page be the header region. If the node-rendered region is inside the header region, we classify the node into the header block. The problem then becomes how to select an appropriate value for N . In Figure 1a, two node regions overlap with the upper N pixels: node 1 doesn't belong to the header block, but node 2 does. We can't exclude node 1 while trying to include node 2 (which we do by increasing the threshold N) because the bottom of node 2 is lower than that of node 1. To solve this problem, we use a dynamic threshold based on the height/width ratio of the node region's shape: the smaller the ratio, the bigger the threshold. Thus, a node with a flatter region has a higher possibility of being put into the header block. Figure 1b shows the dynamic threshold's desired curve.

We use a similar approach to classify nodes to the footer block and detect left and right sidebars.

Because a sidebar region depends on the page's width, the threshold must also be adaptive.

Our approach extracts *spatial features* (top, bottom, left, right, and aspect ratios for each high-level block), which we obtain from the HTML DOM tree, and then trains a nonlinear support vector machine (SVM) with a radial basis function (RBF) kernel based on a set of labeled results to classify each block into one of the five categories (header, footer, left sidebar, right sidebar, or body).

Figure 1c shows the result of high-level content block detection on Microsoft's home page (www.microsoft.com). The blocks are represented with different colors.

Explicit Separator Detection

The algorithm we just described detects high-level content blocks, but explicit separators can partition them further. The third node in the body block shown in Figure 2a, for example, contains three columns that we can further segment for easier browsing on small devices.

We can find explicit separators by analyzing the presence of tags such as `<hr>`, `<table>`, `<td>`, and `<div>` as well as thin images (here, we use thresholds on the width and height to find thin images). By searching for these explicit separators, we partition the content in Figure 2a into four sub-blocks, the result of which is shown in Figure 2b. The three icons appear in one block because they're contained in a single image that can't be segmented any further. The lower part of Figure 2b is divided into three columns. The dashed line in Figure 2b indicates the implicit separators that divide each column into two more parts.

Implicit Separator Detection

Because we can't detect implicit separators directly from HTML DOM nodes, we need special techniques to find them. In our method, we

- collect all the basic content blocks inside each high-level content block after explicit-separator detection;
- project each basic content block along the horizontal and vertical axes to generate projection diagrams;
- select the widest gap on each axis (based on the diagrams) to be an implicit separator to partition the block into smaller ones; and
- iteratively apply this previous process to the small blocks until we can't detect an implicit separator.

Because implicit separator detection is based on projecting the display area of basic content blocks onto vertical and horizontal axes, the basic content block's size is critical to the detection's precision. Large content blocks might fail to reveal fine boundaries, but small blocks could produce excess noise.

To resolve this, we use a method similar to that described by Hwang, Kim, and Seo¹ to produce the basic content blocks. Let's say we have a `<div>` containing a sequence `<a>
<a>
<a>

`. We can group `<a>
`, the most frequently appearing pattern in the sequence, and then merge the three small groups into a virtual node. This virtual node is the basic content block because the last `
` represents an empty line.

Page Splitting

In our approach, the adapted results are typically organized into a two-level hierarchy, with a thumbnail representation at the top level, to provide a global view, and an index to a set of subpages at the bottom level, with more detailed information. Page splitting gives us these adapted results because it actually generates the index page and the subpages. We will introduce these two parts in more detail.

Subpage Generation

Before splitting a page, we must determine an appropriate block size that's smaller than or equal to the mobile device's screen size. Based on page analysis, we can easily extract and store content in the final set of content blocks into subpages.

HTML lets authors use Cascading Style Sheets (CSSs) and style inheritance to specify content style; to keep the block's original appearance, we must first retrieve its style information. Because a CSS is usually specified in `<head>` using `<style>` or `<link>`, we can copy the page's `<head>` section into each generated subpage. For style inheritance, we trace a target node's parent link, retrieve each parent tag's style information, and then merge the information of all its ancestors to this node. Whenever a conflict arises, the child tag overwrites its parent tag.

Index-Page Generation

To generate an index page, we first generate a thumbnail image of the original Web page and then mark the content blocks with different colors. The thumbnail provides the page overview, and it can be generated by rendering the page in memo-

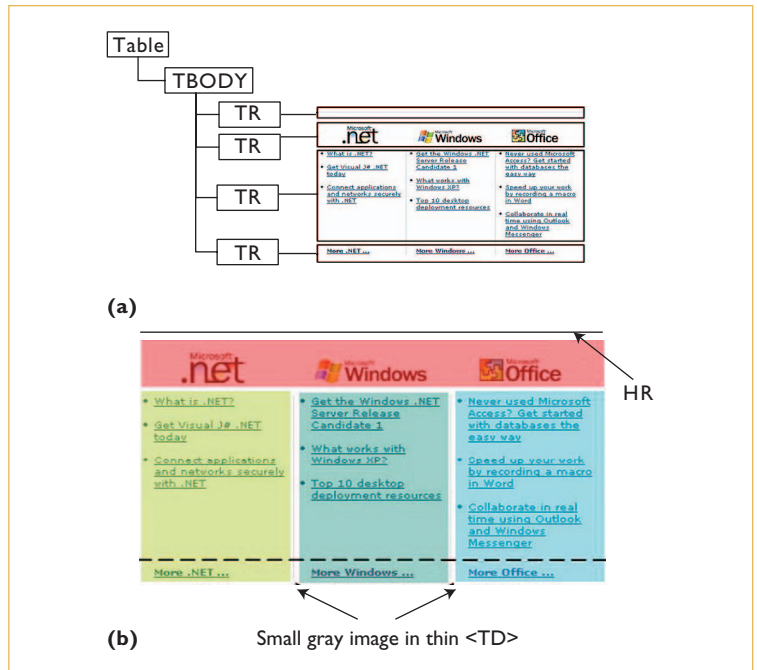


Figure 2. Partitioning. In (a) the third node in the body block from Figure 1b, we can (b) look for explicit separators to further partition the high-level content block.

ry and then using screen capture to produce a final image. Then we put an `` tag to reference the thumbnail image and a set of `<map>` tags in the index page. (A `<map>` tag contains a hyperlink to a subpage.) While browsing an index page, the user can click on any block in the thumbnail to access the corresponding subpage. Figure 3 (next page) shows an example of the index page and subpages generated from the MSN home page (www.msn.com).

Deployment Schemes

Depending on the application scenario, developers can deploy our page-adaptation module on the server side, proxy side, or client side.

Client-Side Deployment

Users could implement our page-analysis and page-splitting algorithms as plug-ins for current mobile Web browsers, but small-form-factor devices often have limited hardware or software capabilities. Client-side deployment schemes must take these limitations into consideration.

To support mobility, most handheld devices use low-bandwidth wireless networks. To address the attendant problem with lengthy download times, the adaptation algorithm on the client device should be *progressive* to give a timely response;

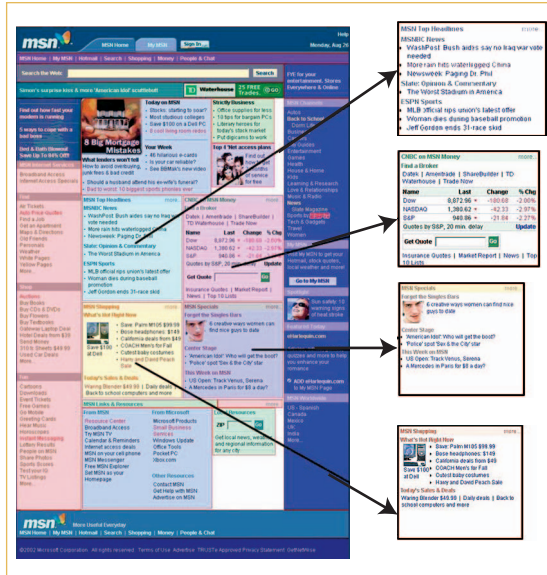


Figure 3. Page-splitting results. For the MSN home page (www.msn.com), the index page is shown on the left, and the subpages are shown on the right.



Figure 4. Page-splitting results in an experiment with 200 Web pages drawn from 50 Web sites. (a) Users marked this page-splitting result as “perfect” (from www.msn.com). (b) Users ranked this result as “good” (from www.microsoft.com/billgates/bio.asp).

that is, it should be able to deal with partial HTML documents. Therefore, users can see the adapted page before it is fully downloaded to the client.

Because our page-analysis algorithm doesn’t assume that the HTML DOM tree is complete, progressive page analysis requires only an HTML parser that accepts partial HTML documents as input. Given that most Web browsers support progressive rendering to deal with low bandwidth, they can fulfill this role. By integrating the thumbnail-generation module into the browser-rendering module, developers can thus implement pro-

gressive page splitting.

Regarding performance, we must keep two things in mind:

- Our method determines whether a tag is a high-level content block solely according to its dimensions and position.
- We apply explicit and implicit separator-detection algorithms to each high-level content block.

In other words, if the HTML content in a high-level content block is downloaded completely, its splitting result is fixed, meaning we don’t need to redo the page splitting for this block after additional HTML content is transmitted to the device; we need only reanalyze the tags in those blocks that weren’t downloaded completely. In this way, we can execute our page-adaptation method both efficiently and progressively.

Proxy-Side Deployment

Another option is to deploy the adaptation module on a proxy server; the advantage is that modification isn’t necessary for client devices or Web servers. The workflow is similar to language-translation services.

A user starts by entering a desired URL into the mobile device. The proxy server fetches the corresponding Web page from the original server, and our algorithm adapts it. The client receives the thumbnail view, with the subpages cached on the proxy server. When the user follows the links on the adapted page, he or she is redirected to the proxy server, and a new request is generated if the page hasn’t been cached. If the requested page doesn’t exist, or if some other HTTP errors occur, the proxy passes the error message directly to the client.

This deployment scheme also improves user-perceived delay: the proxy server usually has a better Internet connection and a much faster processor than mobile devices do, which makes for faster page downloads and processing.

To intercept all subsequent HTTP requests from the client, the proxy should rewrite all the links (that is, the <a> elements) in the adapted results to redirect them to the proxy. Pages with redirection functions should also be processed analogously because the links there aren’t written in <a> elements (for example, the “refresh” metacommand in the <head > section or script functions).

Problems can arise between different proxy and client browser versions – some Web sites generate

Table 1. User-perceived delay with proxy-side deployment of the page adaptation module.

Home page	DT (seconds)	PT (seconds)	PD (seconds)	OD (seconds)
MSN.com	1.2	0.3	4	14
Amazon.com	10.5	1.2	15	35
CNN.com	8.9	0.5	12	32
Yahoo.com	2.0	0.9	7	25
Microsoft.com	4.0	0.3	9	26
Google.com	0.6	0.06	4	5
Go.com	0.8	0.2	4	19
Women.com	8.0	0.6	13	60
Sina.com	2.0	0.4	7	32

*DT stands for download time from server to proxy; PT for processing time at proxy; PD for user-perceived delay; and OD for original user-perceived delay.

different pages for different browser versions, for example. To get the right version, the proxy should therefore simulate the client browser by copying the HTTP header from the client request.

Server-Side Deployment

Our adaptation method can help server-side content providers offer device-aware Web content to users.

When deployed online, the Web server provides a dedicated server on which it installs the page-adaptation module. Similarly to proxy-side deployment, this dedicated server acts as a reverse proxy to fetch content from the original server, convert it, and then transfer the adapted result to the client.

When deployed offline, developers can integrate the page-adaptation module into Web authoring tools. We've developed such a page-adaptation plug-in, called Mobile HTML Optimizer, based on the algorithms described in this article (see www.microsoft.com/frontpage/downloads/addin).

Experimental Results

In 2002, we conducted a set of experiments to evaluate our page-splitting algorithm's performance. For our test data, we selected 50 popular Web sites and then 200 typical Web pages from those sites. We adapted each Web page with our algorithm and asked testers to put the results into one of three categories:

- Perfect – the page analysis and splitting contains no errors.
- Good – the page analysis result is correct, but the page-splitting contains minor errors that don't affect the viewing of the results.

- Error – both the page analysis and splitting processes contain errors that cause a browsing problem or loss of information.

On average, more than 90 percent of our results were either perfect or good; the detailed results for each Web site appear elsewhere.²

Figure 4 shows one page-splitting result marked as "perfect" and another marked as "good." In Figure 4b, one of the block boundaries is wrongly positioned because of the imprecise information returned by the HTML parser.

User-perceived delay (PD) includes page-downloading time from server to proxy (DT), processing time at the proxy (PT), page-downloading time from proxy to client, and rendering time at the client. To study our approach's time performance, we performed another experiment in 2003. Table 1 compares PD under proxy-side deployment versus the original user-perceived delay (OD) without any adaptation. We include the time costs only for DT and PT because measuring page-download time from proxy to client and rendering time at the client would require us to modify the client browser. For the proxy server, we used a Dell Optiplex GX 270 with a 2.8-GHz Pentium 4 processor and 512 Mbytes of memory. The client device was a Pocket PC phone, which uses China Mobile's GPRS service to connect to the Internet. As Table 1 shows, our approach can significantly improve PD – reducing it by a factor of four for some Web pages.

Preliminary results show that our approach is more suitable to pages with multiple topics. For single-topic pages, such as pure news content pages, techniques like text summarization

can be combined with our approach to improve the user experience.

In the future, we'll continue to explore more advanced user interface designs. Instead of clicking on only one block, for example, we can let people use a stylus to select multiple blocks simultaneously for browsing. We'll also study the possibility of applying our page-splitting technique to other scenarios, such as Web search based on page blocks. □

References

1. Y. Hwang, J. Kim, and E. Seo, "Structure-Aware Web Transcoding for Mobile Devices," *IEEE Internet Computing*, vol. 7, no. 5, 2003, pp. 14–21.
2. Y. Chen, W.Y. Ma, and H.J. Zhang, "Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices," *Proc. 12th Int'l World Wide Web Conf. (WWW 03)*, ACM Press, 2003, pp. 225–233.

Yu Chen is an associate researcher at Microsoft Research Asia. His research interests include adaptive content delivery and systems. Chen received a BS and an MS in computer science from Peking University. Contact him at ychen@microsoft.com.

Xing Xie is an associate researcher at Microsoft Research Asia. His research interests include adaptive content delivery, mobile multimedia applications, and mobile Web search. Xie received a BS and a PhD in computer science from the University of Science and Technology of China. He's a member of the IEEE. Contact him at xingx@microsoft.com.

Wei-Ying Ma is a research manager of the Web Search and Mining Group at Microsoft Research Asia. His research interests include information retrieval, text mining, search, multimedia management, and mobile browsing. Ma received a BS in electrical engineering from the National Tsing Hua University, Taiwan, and an MS and a PhD in electrical and computer engineering from the University of California, Santa Barbara. He currently serves as an editor for the *ACM/Springer Multimedia Systems Journal* and an associate editor for the *Journal of Multimedia Tools and Applications*. Contact him at wyma@microsoft.com.

Hong-Jiang Zhang is the managing director of the Microsoft Research Asia Advanced Technology Center in Beijing. He also serves as the editor in chief of *IEEE Transactions on Multimedia*. His research interests include image and video processing, content-based media retrieval, and computer vision. Zhang has a PhD in electrical engineering from the Technical University of Denmark. He is a member of the ACM and a fellow of the IEEE. Contact him at hjzhang@microsoft.com.