Optimization of web page for mobile devices

Xinyi Yin
Department of Computer Science
National University of Singapore,
Singapore 117543.
yinxinyi@comp.nus.edu.sg

Wee Sun Lee
Department of Computer Science and
Singapore-MIT Alliance,
National University of Singapore,
Singapore 117543.
leews@comp.nus.edu.sg

ABSTRACT
Delivering web pages to mobile phones or personal digital assistants has become possible with the latest wireless technology. However, mobile devices have very small screen sizes and memory capacities. Converting web pages for delivery to a mobile device is an exciting new problem. In this paper, we propose to use a ranking algorithm similar to Google's PageRank algorithm to rank the content objects within a web page. This allows the extraction of only important parts of web pages for delivery to mobile devices. Experiments show that the new method is effective. In experiments on pages from randomly selected websites, the system needed to extract and deliver only 35.4% of the objects in a web page in order to provide 91% of a viewer's desired viewing content. This provides significant savings in the wireless traffic and downloading time while providing a satisfactory reading experience on the mobile device.

Keywords
PDA (Personal Digital Assistant), HTML, WWW (World Wide Web)

1. INTRODUCTION
Web content is currently designed for the desktop personal computer (PC) with a big monitor and rich memory resources. PC users can use a convenient input device such as a mouse to retrieve any web page from any website. Downloading time is rarely a problem as the PCs are usually connected to the internet through high capacity lines and the large screen allows many irrelevant objects such as advertisements to be placed on the screen without overly distracting the user.

In the past five years, many mobile devices with medium and small sized screen and limited memory have appeared. For example, it is now possible to browse the web using personal digital assistants (PDA) such as the Palm or Pocket PC. The mobile phone, which is currently the most popular mobile device, has many features that make browsing the internet possible. However, these devices are not ideal platforms for surfing the web. First, the wireless bandwidth is quite limited and very expensive. Secondly, the screen size varies and can be very small, for example 120*90. Third, some devices, such as mobile phones, have very limited memory capability. Normally, the content of a single web page will be larger than what a mobile phone can hold.

Researchers have spent a lot effort in solving the problem of enabling such devices to view the web content in a satisfactory manner. Some of the solutions work in the push model, like [16], where the selected content is pushed to the PDA through a synchronization process. Others use pull model, like Opera browser, where the content is extracted and optimized. Normally, these methods display the whole web page. The disadvantage of this approach is the long downloading time when bandwidth is limited and the large amount of scrolling required in order to get to the relevant parts of the web page.

This paper presents a system that provides automatic conversion of web content into a form that is optimized for mobile devices. Our approach is to extract and present only the important parts of the web page for delivery to the mobile device. Such a method saves not only download time but also the time spent scrolling on the small screen devices. Errors in extraction by the system can be corrected by allowing the user to request the whole page if they are not satisfied with the extracted content. If the extraction error can be kept at a minimal level, such a system will provide a more pleasant experience for surfing the web on a mobile device.

The basic technology behind the approach is a ranking algorithm for elements of a web page. The idea behind the ranking algorithm is to first represent a web page as a graph model and then exploiting the graph structure to rank the elements. To obtain the graph, we first divide the page into inseparable basic elements. We assume that the user is entering a web page from a link. Based on the type, size, physical position shape and similarity to the anchor text of the in-link, we give each basic element an initial rank value. We use weighted edges to represent relationships between two basic elements. The weights are a function of attributes of the two elements, such as word similarity and physical proximity of the elements within the page. This graph representation of a web page is quite different from the commonly used tree-based analysis of web pages. The graph model of a single web page is made up of hundreds of basic elements that are linked to each other in a very

complex manner. Such structure is similar to the whole Internet, which is also made up of many interrelated web pages

The most successful ranking algorithm for web pages is a random walk model used by the Google search engine. The web is treated as a graph on which surfers move randomly from page to page according to the links on the page. The ranking of the web page is then the expected number of surfers visiting the page at any time. The manner in which a person reads a web page is similar to how a surfer surfs the web. The reader enters the page through a link and is drawn to elements that are related to the anchor text in the link and are located in central positions on the page. After reading an element, the reader moves on to a highly related element. By modeling the strength of connections between elements according to their similarity, we are using a simplified model of the movement of the readers' attention on the web page. We then rank the elements according to the expected number of readers reading the particular element at any time. Based on the rankings, we select a rectangle covering all the important elements of the web page and transmit the content of the rectangle.

The contributions of this paper include a new approach for enabling pleasant surfing experience on mobile devices and a new model for processing HTML document. Rather than the traditional tree model, we convert the HTML document into a graph which allows us to use Google's successful PageRank approach for finding important elements in the document.

We organize the paper in the following way. In section 2, we give an overview of the system. In section 3, we will give the design of the system. In section 4 we will discuss the dataset, and describe the evaluation of the system. Section 5 is about related works. In section 6, we will give our conclusion and the direction for future research.

2. CONVERTING A WEB PAGE INTO A GRAPH

2.1 Basic Elements

To construct a graph from a web page, we first identify the nodes, which are the basic elements in the web page. Then we specify the edges of the graph which encode the relationships between pairs of basic elements.

Researchers have proposed different methods to divide an HTML page into logic blocks. For example, [5] proposed a visual based method to analyze the structure of a web page, and [2] provides a method to automatically understand the semantic structure of HTML pages based on detecting visual similarities of content objects. In our system we are trying to find all the inseparable visible elements in an HTML page. We use the DOM interface provided by web browser. From bottom up we identify nodes by two simple rules:

1. Image, link, text paragraph, and other visible object will be a basic element if their parent node doesn't contain another child that is overlapping with it.
2. For overlapping objects, the minimal container of the two objects will be a potential element to be checked by the rule 1. If one object contains the other, the bigger object will be chosen. Otherwise we will seek from bottom up to check their nearest common container.

For example, a web page contains a list of links. The links are not overlapping with each other. Each of them will be treated as basic element. In another web page, a text paragraph has a name which is a link. Here we have two overlapping objects, the bigger one, text paragraph, will be chosen to checked by rule 1, if the text paragraph is not overlapping other element at a higher level, it will be chosen, other wise we will recursively search upward.

As shown below. Our algorithm will convert original web page in to a list of basic elements.

Figure 1. Original HTML page

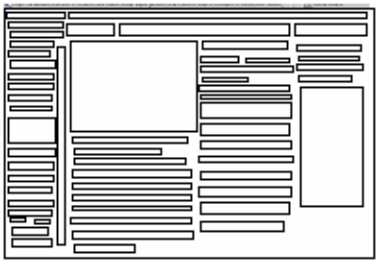

Figure 2. Decompose the original HTML page

2.2 Graph

Assume that we have *N* basic elements. We will build a graph such that the sum of the weights coming out of each node is 1. This allows us to use the weight on the edge as the probability of a reader going to the next element along that edge.

We first introduce an additional node *S* which can be considered as source of visitors to the web page. This node also serves as the sink where readers who stop reading at any particular element will go to.

Based on the features of a basic element, we will connect it to the source *S* with a weight that represents its contribution to the topic of a web page. We take the following features into consideration:

1. Size (Ps): Element with bigger size is more important than a smaller one. The contribution of size to the importance of element i can be calculated by

$$Ps(i) = Size(i) / \sum_{j=1}^{N} Size(j)$$

where size is measured by the number of pixels.

.

2. Text length (Pt): Element with longer visible text has higher importance. The length is the number of visible words. For example, A link with longer anchor text will draw more attention. The contribution of text length can be calculated by:

$$Pt(i) = Length(i) / \sum_{j=1}^{N} Length(j)$$

3. Match (Pm): Visitors from the source S will pay more attention to the content that is similar to S. We calculate the cosine similarity between the visible text in the element and the anchor text of *S*. We also use a stop word list including non-informative words that are commonly used in the internet context such as "click" "next" "more" "read" and others.

4. Width/height ratio (Pr): The shape of an element reflects its importance. For example, for an image, the regular image is usually more important than those irregular one., We use the following formula to calculate the value for images:

$$\Pr(i) = \begin{cases} 1 & if\ 0.3 < Width(i) / Height(i) < 3 \\ 0 & else \end{cases}$$

For different categories of elements we will use different formulas. For a text block we use a formula that favors of higher Weight/height value:

$$\Pr(i) = \begin{cases} 1 & if\ Width(i) / Height(i) > 4 \\ 0.5 & else \end{cases}$$

5. Physical offset (Pp): Physical position is calculated by pixels. It is actually a very important feature. Element closer to the center point is more important than those near the edge of the page. We calculate the position information from, first, the physical distance between the center of the element and the center of the screen, second its horizontal offset information. Let the screen center be (X_C, Y_C) and the center of element *i* be (X_i, Y_i). Then

$$distance(i) = 1 - \frac{\sqrt{(Xi - Xc)^2 + (Yi - Yc)^2}}{\sqrt{4Xc^2 + 4Yc^2}}$$

$$offsetX = 2 * Xi / ScreenWidth$$

$$offsetX(i) = \begin{cases} OffsetX & if\ 0.3 < offsetX < 0.7 \\ 0 & else \end{cases}$$

$$Pp(i) = offsetX(i) + distance(i)$$

In this formula we specially take X offset into consideration because normally a web designer will put the important content in the center of the screen, and both left and right sides are for irrelevant or less important content. However, we did not make use of the vertical offset, as we see that important content in a web page can be very long.

We normalize the distance with the diagonal of the whole HTML page. All the constant value in the formula is chosen according to an ordinary web page layout.

The weight from the source *S* to node *i* is calculated by the function *W(S,i)* where

$$W(S,i) = I(S,i) / \sum_{j=1}^{N} I(S,j) \text{ and}$$

$$I(S,i) = W1 * Ps(i) + W2 * Pt(i) + W3 * Pm(i) + W4 * Pr(i) + W5 * Pp(i).$$

The weight represents the probability that the reader's eye goes to each of the element as he enters the web page; it is obvious that not all the elements are equally likely to be viewed.

From each node *i*, we also set a weight $W(i,S)=\beta,\ 0 \le \beta \le 1$ to indicate the probability that the person stops reading at node *i* and goes back to the source node *S*.

As described earlier, the weight of the edge between any two basic elements is a evaluation of how likely the reader is to continue with the second element after reading the first. It is calculated using the following features:

1. Distance Pd(i,j): The physical distance (in pixels) of two elements in the layout of an html page.

$$Pd(i,j) = 1 - \frac{\sqrt{(Xi - Xj)^2 + (Yi - Yj)^2}}{\sqrt{4Xc^2 + 4Yc^2}}$$

2. Horizontal offset (Ph): set as 1 if two element's horizontal offset is the same, otherwise 0.
3. Neighborhood (Pn): Set as 1 if two elements are neighbors, otherwise 0.
4. Match (Pm): the cosine similarity between the visible texts in the two elements.
5. Width (Pw): Set as 1 if two elements have the same width, otherwise 0.

The similarity *S(i,j)* between distinct elements *i* and *j* is calculated by the sum of the five features. For a node *i*, we have already used up a weight of β for the link back to the source. Of the remaining amount (1-β), we use a fraction α as the loop back to itself to indicate that the user continues reading on the element for a period. Hence W(i,i)=(1-β)α. The weight from distinct nodes *i* to *j* is then calculated as

$$W(i,j) = (1-\beta)(1-\alpha)S(i,j) / \sum_{k=1}^{N} S(i,k).$$

2.3 Random Walk on the Graph

In previous section, we have described the algorithm to convert any web page into graph, with the nodes representing the basic elements, and edges representing the relationship between the basic elements. In this way we convert an html web page in to a structure that is similar to the whole internet.

The most successful search engine is Google, which proposed the idea of "PageRank" to describe the importance or quality of a single webpage. In our paper we will borrow the idea of PageRank to calculate the importance and quality of each basic node in a web page. PageRank can be thought of as a model of user behavior, where a user is given a random web page and he will follows the links until he get bored. The probability of a user visiting a web page is proportional to the PageRank, which can be calculated iteratively by

$$PR^t(i) = (1-d) + d \sum_{(j,i)\in E} PR^{t-1}(j) / C(i)$$

where $PR^t(i)$ is the PageRank of node *i* at time *t*, *E* is the set of edges, *C(i)* is the number of links going out of page *i* and *(1-d)* is the probability that the user will get bored and leave a certain web page back to the source . Note that in PageRank all out links are treated equally. In contrast, we have more information based on the similarity between elements, hence have given different weights to different links.

We can similarly calculate a ranking that is proportional to the probability of a reader being at a node by using an iterative algorithm that does the following updates

$$R^t(i) = \sum_{j=1}^{N} W(j,i) R^{t-1}(j) + W(S,i) R^{t-1}(S)$$

and

$$R^t(S) = \beta \sum_{j=1}^{N} R^{t-1}(j),$$

Where $R^t(i)$ is the ranking of node *i* at time *t*. The value $R^t(i)$ converges to a value that is proportional to the probability of being at the node (the first eigenvector of the transition matrix).

3 EXTRACTING AND OPTIMIZING

3.1 Extracting Relevant Elements.

The task of a search engine is to return the top results that match the search query. Google achieves this by first gathering the matched web pages and then returning them in the order of its PageRank. However, even though we have the ranking for each element in the web page, we can not simply return the elements to the user by its rank. In a search engine, every individual webpage is independent and it does not matter if one web page is returned before or after another web page. But in our system, there are semantic and logical relationship between the elements and the order of relevant elements has to be returned as it appeared in the original web page. The user will feel unhappy if he gets an article extracted from a web page that looks nice but has the wrong ordering of the elements.

Our design goal is return user the most relevant content, which is those with the highest ranks, but still need to keep the original look and feel on the mobile device. We use a simple heuristic to retrieve complete article based on the ranking of the elements.

```
Select ()
{
   list.insert(topnode);
   T1=rank(topnode)/3;
   T2=average (S(topnode,j));
   while(node=list.getNext()!=NULL)
   {
      d=Distance(topNode,node);
      tw=(1+m*d/10)*T1;
      for(each neighbor ni of node with weight(n, ni)>t)
      {
          tr=(1-n*d/10)*T2;
          if(rank(ni)>tr)
            {
               list.insert(ni);
            }
       }
    }
}
```

As shown in the algorithm, firstly, we sort the nodes and pick the node with the highest rank. Then set two thresholds based on the rank score of the top node. T1 is a threshold on the edge weights, and T2 is a threshold on the ranks.

We assume that the top node is the center of an article; our task is to walk around from the top node following certain links to reach all nodes that also belong to the main article. T1 is used to set the minimal weight on the links that random walker should follow, and T2 is used to set the minimal rank that an element needs in order to be considered as relevant.

As the random walker moves further away from the top node, which is calculated by d=Distance(topNode, node), which is number of jumps between topNode and node; we increase the threshold tw on the weight; otherwise it may follow a weak link and reach to the center of an irrelevant element. So we set the aggressiveness parameter m, which increases the tw on each jump, and decrease the threshold on rank tr to allow element with lower rank to join. The rationale behind this is the assumption that the nodes further away from the center node, even it is relevant, will have lower rank because of the distance and offset, as expressed in section 2.

After we obtain the list, we will put the elements to its original position in web page, and find a minimal rectangle that covers all the nodes. The procedure is self evident and it guarantees the integrity of the original content.

3.1 Optimizing for Mobile Device

In previous section we obtained a rectangle within a web page that encloses the true article. The target rectangle may be larger than most mobile device; we need optimize the content and make sure it looks nice on the end device.

We have the following design goals:

1. Minimize the scroll action on small screen device, eliminate scroll in one direction.
2. Maximize the similarity between the layout of the optimized content and the original web page.

We convert the HTML layout so that the width of the re-rendered HTML page is smaller than the screen size. To maximize the similarity between the original page and the re-rendered page, we need retain the HTML hierarchy structure of the original page. Our algorithm can be described as from Figure 3 to 5.

Figure 3. Original HTML page

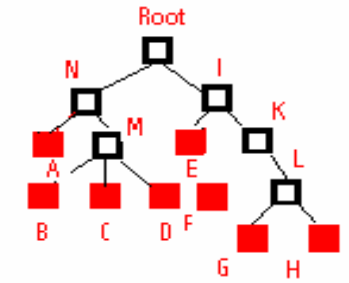

Figure 4. Layout tree

Figure 5. Optimized result

In Figure 3, the search engine indicates a larger area that will returns to a mobile device. In Figure 4, the system will put the elements in the selected rectangle in to a called "Layout Tree" structure. Layout Tree has the following features.

1. Each element maintains hierarchical relationship in the original HTML tree.
2. Each node has a rectangle data that records the area that it occupies in original HTML page.
3. Children of the same parent node are at the same hierarchical level in the original HTML tree.
4. The parent node's rectangle is the minimal rectangle that covers all the children's rectangles.

The layout tree is built bottom-up, As seen in Fig 4, suppose node B, C, D is at the same hierarchical in the original HTML tree, so we put it under the node M and set the rectangle of M as the minimal rectangle that covers B C D, then M and A shares the same parent nodes, and so on.

The algorithm to re-render the HTML can be described as the following recursive algorithm

```
Render(node)
{
   if( node.child=NULL )   return node.HTML;
  if(node.width>Screen_Width)
    { //if larger than the screen, recursively call each child.
       for(each child for node)
            result  + = render (node.nextChild);
     }
    else
    {    //The screen is wide enough to put all child nodes
           for( each child for node)
            result  + = node.nextChild.HTML;
    }
}
```

If we call Render() with root node as parameter, it will returns the optimized HTML page. Suppose the root's width is wider then the small screen, the algorithm recursively call render with to nodes N and I to process each sub tree. The node N's width is still wider than the screen, we recursive call Render(A) and Render(M). At the node M, suppose it is not wider than the screen width, so we returns B C D's HTML source. In case when the width of basic element is larger than the window, we will zoom in the content to fit the screen. Figure 5 shows the result.

4. EXPERIMENT RESULT AND ANALYSIS

We have implemented the system to test our ideas. We have two goals in the system. The first is to satisfy the user's information need. We try to deliver all the information that a user wants in a web page. The second is to save the bandwidth and minimize scrolling in the mobile device. We will use the following measure to evaluate the effectiveness of the system.

1. The recall value R:
R=(retrieved elements that are relevant)/(all the relevant elements)
(We calculate by the area it occupies in the web page)

2. The percentage of returned elements of the extraction:
Return=(number of retrieved elements)/(number of elements on the web page)

It is preferred that the system deliver as little content as possible while achieving the high average recall

We created the test data in the following manner.
First, we randomly selected 309 websites from Google directory. From each web site we chose an average of 4 web pages and recorded the anchor text of the links that lead to the pages.

Second, for each web page, we allowed a user read the anchor text, and ask the user to use the mouse to specify the area that she wanted to read on mobile device. We recorded the web site name, the anchor text, and the area specified by the user.

In this manner, we collected altogether 1388 web pages. We further divided the set for design and evaluation purposes. We set aside 994 sample pages to use in designing the system and adjusting the parameters. The remaining 394 samples are never seen and used only for evaluation. The design set and evaluation set does not share any page from the same web site.

To extract content from each test web page, first, we will use the web browser's DOM interface to and divide the web page into basic elements, each basic element will be given rank, and weight to its links, as described in section 2, Then we will use the extraction algorithm to extract article from the web page and the valuation of

We set a target average recall rate as the goal, and try to obtain a system that satisfies the recall rate using the design set. We then evaluate the return rate in the delivery on both the design and evaluation set. The return rate on the evaluation set should be a fair indication on future performance as we have never seen the pages during the design process. For the experiment we set the target average recall to be 90%. We did not use 100% because many of the elements selected by the user are actually ambiguous and hence the additional benefits of achieving total recall are small. We compare our system with three different algorithms.

1. Simple Match: We calculate cosine similarity between the anchor texts with every basic element. We select all the elements where the similarity is above a threshold and return elements in the smallest rectangle surrounding the selected elements.

2. Extend Match: Based on the result of the simple match, we do a second round calculation, calculate the cosine similarity between selected elements with the other elements. If the similarity is above certain threshold, we add the new element to selected list and return elements in the smallest rectangle surrounding the selected elements.

3. Rank without random walking: We give each element an initial rank and weight as described in section 2.2, without our random walking algorithm, we simply set a threshold which can achieve the recall that is above 0.90.

4. Full implementation of our algorithm, the initial condition is set the same as 3. We improve it with our random walking algorithm.

Table 1: Experiment Result

Method	Recall_1	Return_1	Recall_2	Return_2
1	0.77	52.3%	0.67	55.4%
2	0.85	77.5%	0.78	.68.1%
3	0.91	54.6%	0.88	52.8%
4	0.92	38.3%	0.91	35.4%

In the table, "Recall_1" and "Return_1" are the experiment result on designing set. "Recall_2" and "Return_2" are the result on evaluation set. In our experiment, we set $\beta=0.25$ *and* $\alpha=0.85$. The "word match" and "extend match" algorithms can only reach to maximum recall of 0.77 and 0.85 respectively not matter how we set the parameters. Both algorithm need to deliver above 50% of the elements in order to achieve that recall. For the "rank without random walking" method in the test set can achieve a recall of 0.91, but it need to deliver 54% of the element of the web page. Initial rank really provide valuable additional information, but it is not sufficient, we need the "random walking" to find out the top node. With our algorithm we just need to deliver 38% to achieve a better recall. "Random walking" algorithm really improves the result. In the evaluation set, we observed a similar pattern.

On average the algorithm need to return only about 36% of all the elements in a web page to reach a recall above 0.90. We believe this result is encouraging for mobile device. First, 35% of the elements do not mean only 65% of traffic savings. Actually the saving in bandwidth is much higher because most of the elements that are removed are usually multimedia elements or advertisements.

Second, 0.90 of recall does not mean that user normally get only 90% of a full article. As we carefully study the web pages, we find the most of the error are caused by the ambiguity of the selection of the "relevant area". For example, in the following web page shown in Figure 6.

Figure 6. Sample Page

The red rectangle is what user defined as relevant. It covers the entire article but the edge is not precise. The black rectangle is what the algorithm returns as positive. The algorithm is 100% correct. But if we calculate the recall by our definition, it is only 0.90.

The algorithm performs consistently on both the design and evaluation case. This shows that the algorithm is stable over different websites. Based on these results, we are confident that our system and implementation achieves the design goal.

5. RELATED WORK

Google [1] proposed that web is a graph on which surfers move randomly from page to page according to the links on the page. We believe the manner in which a person reads a web page is similar to how a surfer surfs the web. The reader enters the page through a link and is drawn to elements that are related to the anchor text in the link and are located in central positions on the page. After reading an element, the reader moves on to a highly related element. Google returns the search result ranked by the page rank, while we rank the elements in a web page and return the top content for the mobile device.

The SmartView system in [11] is based on idea of "divide and view". The system performs partitioning of HTML document content into logical sections that can further be selected by the user and viewed independently from the rest of the document. The advantage of [11] is that it allows user to randomly access any website and gives the user full control of which content to read displayed without predefining a "hot area", but [11] doesn't handle the situation when a logical section is much bigger than the screen size of the target device, as it is always this case if user is surfing a web page on a mobile phone.

This work is related to the research area of web page cleaning, which believes the useful information on the web is always accompanied by a large amount of noise such as banner, advertisement, navigation bars, copyright notices. etc. [14]. Usually a web cleaning system will study and compare a lot of samples from a single site and learn the rules to identify "what is noise?" However, we are solving the same problem from the different angle. Our system answer the question "what is not noise?" and our system doesn't require samples from the same site, this features make it a very ideal solution for mobile device where we could not predict what web page a user may want to read.

The web is not personalized and device independent, most of the commercial system creates special web content for the mobile devices. (For example, web Clipping [17], NTT i-Mode [18], AvantGo [16]) This solution has its limitation. The surfing experience and content is different the cost to maintain this service and synchronize with the PC web is difficult. We believe mobile Internet is an extension of existing Internet and we should develop systems that convert the content in the Internet to a format

that is suitable for various small screen devices. The systems need to perform three functions, including scaling, manually authoring, transforming. The functions are summarized in [6]. For example, [7] and [8] use summaries of a single or multiple pages to present to the user. [9] and [16] describe the process of manually extracting only the useful information from existing web. [10] proposed a sophisticated method for performing transformation.

6. CONCLUSION

Our goal is to design a system that can deliver device independent content to mobile devices from any web page in order to fulfill the user's information need on devices that have minimal computing power, screen and bandwidth available. We achieve this by ranking the importance of each element in a given web page and generating a customized "web" for mobile devices. In this paper we proposed three interesting ideas. First, it is possible to represent the HTML web page with a graph structure Second, based on our ranking algorithm that is similar to Google's PageRank, the system can understand what is the most important topic of a web page in order to retrieve and extract the content for different devices. Third, we develop an algorithm to reformat and optimize the subset of the original web page for different mobile device. Our experiments show that in the vast majority of cases the proposed system really provides the expected results, making it a useful system.

With the development of wireless technology and emergence of various mobile devices, people will not be limited to the desktop computer. We will access the Internet through all possible devices. Instead of building different webs for different devices, we strongly believe that the right direction is to convert and deliver the same content in different ways to different devices.

7. REFERENCE

[1] Sergey Brin, Lawrence Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine. WWW7 / Computer Networks 30(1-7): 107-117 (1998)

[2] Yudong Yang and HongJiang Zhang (2001): "HTML Page Analysis Based on Visual Cues" In ICDAR (2001)

[3] Shipeng Yu, Deng Cai, Ji-Rong Wen, Wei-Ying Ma: Improving pseudo-relevance feedback in web information retrieval using web page segmentation. WWW 2003: 11-18

[4] Yu Chen, Wei-Ying Ma, Hong-Jiang Zhang: Detecting web page structure for adaptive viewing on small form factor devices. WWW 2003: 225-233

[5] Xiao-Dong Gu, Jinlin Chen, Wei YingMa, Guo-Liang Chen Visual Based Content Understanding towards Web Adaptation. In: Second International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH2002), 29-31 May 2002, Spain.

[6] Trevor, J. Hilbert, D.M., Schilit, B.N., Koh, T.K. (2001): "From desktop to phone top, a UI for web interaction on very small devices". Processings of the 14th annual ACM symposium on user interface software and technology (UIST2001)

[7] Buyukkokten, O., Garcia-Molina, H., Paepcke, A. and T. Winograd. Power Browser: Efficient Web Browsing for PDAs. In Proceedings of the ACM Conference on Computers and Human Interaction 2000 (CHI'00), 2000.

[8] Buyukkokten, O., Garcia-Molina, H., Paepcke, A. Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices. In the Proceedings of the Tenths International World Wide Web Conference (WWW 10), 2001.

[9] Bickmore, T., Schilit, B. Digester: Device Independent Access to the World Wide Web. Proc. Sixth International World Wide Web Conference, Santa Clara, 1997

[10] Bharadvaj, H. Joshi, Anupam, and Auephanwiriyakul, S., "An Active transcoding Proxy to Support Mobile Web Access", Proc. 17 IEEE Symposium on Reliable Distributed Systems, 1998

[11] Natasa Milic-Frayling, Ralph Sommerer: SmartView: Flexible Viewing of Web Page Contents, The Eleventh International World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA, 2002

[12] Corin R. Anderson and Eric Horvitz: Web Montage: A Dynamic Personalized Start Page. In Proceedings of the 11th World Wide Web Conference (WWW 2002). 2002.

[13] Corin R. Anderson, Pedro Domingos, and Daniel S. Weld. Adaptive Web Navigation for Wireless Devices. In Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01). 2001

[14] Lan Yi, Bing Liu. "Eliminating Noisy Information in Web Pages for Data Mining." To appear Proceedings of the ACM SIGKDD International Conference on

Knowledge Discovery & Data Mining (KDD-2003), Washington, DC, USA, August 24 - 27, 2003

[15] Lan Yi, Bing Liu. "Web Page Cleaning for Web Mining through Feature Weighting" To appear in Proceedings of Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), Aug 9-15, 2003, Acapulco, Mexico

[16] AvantGo http://www.avantgo .com

[17] MOZAT http://www.mozat.com

[18] Web Clipping
http://www.palmos.com/dev/tech/webclipping/

[19] NTT i-Mode http://www.ntt.co.jp/