Getting Back to Back: Alternate Behaviors for a Web Browser's Back Button

Saul Greenberg
Department of Computer Science
University of Calgary
Calgary, Alberta
Canada T2N 1N4
+1 403 220 6087
saul@cpsc.ucalgary.ca

Andy Cockburn
Department of Computer Science
University of Canterbury
Christchurch
New Zealand
+64 3 364 2362
andy@cosc.canterbury.ac.nz

ABSTRACT
This paper concerns the ubiquitous Back button found in most Web browsers. First, we outline why Back is an effective method for revisiting WWW pages: a) It allows rapid return to very recently visited pages, which comprises the majority of pages a person wishes to return to; b) People can use it even with a naïve model of the way it works; c) People usually keep it on permanent display because it is visually compact; and d) Back works via a simple 'click until the desired page is recognized' strategy. Second, we investigate the behavior of Back. The typical stack-based behavior underlying Back is problematic because some previously seen pages are not reachable through it. To get around this problem, we offer several alternate behaviors of the Back button, all based upon a recency model. The advantage of recency is that all previously seen pages are now available via Back. Because trade-offs exist, we present both problems and prospects of these different Back behaviors in various navigational situations.

Keywords. History, page revisitation, reuse, navigation, browser design, world wide web, hypertext.

1. INTRODUCTION
A person's ability to find and navigate effectively to new information and to new web sites is extremely important, and this has driven many researchers to understand both how people navigate within the Web, and how Web sites and browsers should be designed (e.g., Rosenfeld and Morville 1998; Forsythe, Grose and Ratner 1997; Sano 1996; Nielsen 1995). Equally important, however, is a person's ability to return to pages and sites he or she has already seen: page revisitation is a regular and surprisingly strong navigational occurrence. Tauscher and Greenberg (1997), for example, found that around 60% (*sd* = 9%) of all pages an individual visits are to pages they have visited previously.

Given this statistic, we believe that Web browsers should go to great lengths to support effective page revisitation. Indeed, most browsers do provide revisitation support through various mechanisms: the Back and Forward buttons, history lists, bookmark facilities, and even site maps that graph the various sites and pages that a person has visited (see survey by Cockburn and Jones 1997; Tauscher and Greenberg 1997).

In spite of the many revisitation mechanisms now available on browsers, it is the Back button whose use predominates: Tauscher and Greenberg (1997) discovered that pressing the Back button comprised over 30% of all navigational acts. In contrast, other revisitation facilities are used infrequently e.g., < 3% for bookmarks, and < 1% for history systems.

In spite of this heavy use, the behavior of the Back button has changed little from the version first seen in NTSC's Mosaic. As well, the rationale behind its design has been lost in time. Our goal in this paper is to get back to the design premises underlying the Back button, to analyze its advantages and disadvantages, and to look at alternate behaviors for the way this button could work.

In the section that follows, we briefly outline why Back (and its Forward partner) is, in principle, a good method for returning to many (but not all) previously seen pages. We also discuss when and why it fairs well against other revisitation methods. The subsequent section defines and analyzes the current stack-based Back behavior. In the remaining sections, we propose and analyze alternate behaviors based upon a recency model. We caution that no firm recommended behavior of the Back button will be offered: each behavior has both advantages and disadvantages to particular situations.

2. WHY BACK IS A GOOD REVISITATION METHOD
In this section, we discuss several reasons why Back and Forward are an effective method for revisiting pages. These buttons allow rapid return to very recently visited pages. People can use them even when they have a naïve understanding of the way they work. People usually keep these buttons on permanent display because they are visually compact. Finally, the buttons work via a simple 'click until the desired page is recognized' strategy.

2.1 The Recency Phenomenon

We stated previously that around 60% of all personal page navigations are revisits. The question is how many of these are to pages a person has seen recently, and how many are those to pages seen long ago? For recently seen pages, it is reasonable to expect a person to use some type of Back button, for those pages are reachable with only a few clicks. For pages seen long ago, the many clicks and page reviews required would make the Back button onerous.

To answer this question, Tauscher and Greenberg (1997) analyzed sequences of individual users' page visits to produce the *recurrence distribution*. This frequency distribution plotted the times the next page visited was in fact a revisit to each of the following: the previous page; one seen two pages ago; one seen three pages ago; and so on. They found that 58% of these subject's page visits are to pages seen previously, with the rest being to new pages. They also found a very strong recency effect, where the most recently visited pages are the most likely to be visited next. For example, there is an 39% chance that the next URL visited will match a member of a set containing the 6 previous submissions. Increasing the size of the set increases the probability only slightly: there is a 43% chance when the set includes the last 10 pages seen, 48% for the last 20 and 52% for 50. As a reminder, the maximum possible probability is 58%, as the other 42% of page visits are to new pages.

The implications of Tauscher and Greenberg's finding are clear. Because of the strong recency effect, there is a very strong chance that a person can return to the desired previously seen page with fairly few button clicks (note that this assumes an 'ideal' model of Back behavior, which is actually not the case as discussed in later sections). However, we caution that other revisitation methods are still required (history lists, graphical browsers, search engines, etc.) as some—admittedly few—of the desired pages may have been seen long ago.

2.2 Robust Use in Spite of Naïve Models

As will be described in Section 3, current commercial systems use a stack to implement Back and Forward button behavior: the effect is that a person can use these buttons to move up and down the single hierarchical path captured by the stack, but cannot use it to revisit any pages that branched off that path as these are popped off the stack.

In spite of this algorithmically simple model, most people have quite naïve models of how the stack-based Back and Forward buttons actually behave. When Cockburn and Jones (1996) asked eleven computer professionals about Back button behavior, only one knew that it was based on a stack, with most others incorrectly thinking it modeled a simple history list of all pages visited. They were then given a simple navigational task, the critical component which was to select a link on page A to a child page A_1; go back to A (using Back); and then select another link to child page A_2. They were then asked whether they could return to page A_1 via the Back button. Eight of the eleven predicted—incorrectly—that Back would return them to that page. They were completely unaware that page A_1 had been popped off the stack, and were surprised when they tried to follow through on their prediction.

What is surprising about these findings is that people—in spite of their naïve models—are capable of using the Back and Forward buttons regularly (*cf.* Back accounts for over 30% of all navigational acts). All this indicates that the Back button is a robust revisitation device in spite of the poor mental models people have of its actual behavior, and in spite of its occasional (from the user's point of view) failure to bring them back to pages they had recently visited.

2.3 Visual Economy

Forward and Back are relatively simple in that they require only two buttons to allow a person to navigate to and from previously seen pages. These buttons are modest in size, fitting comfortably within most browser toolbars. While users can customize their browser to remove these buttons, we suspect that most keep them around.

In contrast, other revisitation schemes are expensive in terms of screen demands. Most rely on separate or tiled optional windows to present a list, tree or graph (see surveys by Cockburn and Jones 1997; Tauscher and Greenberg 1997).

Given today's limited screen space and a person's overhead for window management, we suspect that people are reluctant to keep these add-on windows on permanent display. This may account partially for the poor use of the history window observed in Mosaic (<1% of all navigational actions). We suspect that these windows are raised for occasional use and then put away. Once 'out of sight', people are less likely to go through the effort of raising them for routine page revisitation. In comparison, the visual economy of the Back and Forward buttons means that they will be kept on the display, where they serve as an easily activated method for page revisitation.

2.4 Recognition instead of Decision Making

Back and Forward work by displaying the appropriate page in the browser window. In common use, one merely keeps clicking on the button until the desired previously-visited page is recognized. Other revisitation schemes (e.g., history lists) require considerably more cognitive overhead.

First, most other revisitation schemes represent individual pages in an abstracted form, which people may find difficult to match to the page they are looking for.

- *URLs.* Some systems represent pages as a list or graph of URLs. While some URLs are meaningful to people, most are not. Some sites use cryptic names in the URL path that do not reflect the page content; others have

URL paths that do not reflect the navigational page structures (i.e., the directory/file structure does not match the page/sub-page navigation); and others use dynamic pages producing what looks like nonsense URLs.

- *Titles.* Some systems display the label specified by the page <title> tag. These too have problems in practice. Title names often do not match the *de facto* title i.e., the prominent text or graphics appearing at the top of a page. Titles may be missing. They are often incorrect, as can happen when a person copies and modifies an old page but forgets to revise the title. Titles may not follow a naming convention that ties together related pages. Finally some sites use a single title for all the pages within it
- *Thumbnails.* A few systems use a graphical miniature of a page. When these thumbnails are small, it may be difficult to distinguish one page from another. Even if the thumbnail were large enough to give a good sense of its graphical characteristics (which introduces space concerns), people may still have problems when they visit typographically similar pages.

The second problem with other visitation schemes is that people still have to find the desired page within some kind of graphical representation. This may be a linear list (e.g., Netscape's history window), a hierarchical indented list (e.g., Microsoft's Internet Explorer 4.0 history pane), or a graph (e.g., Mosaic-G by Ayers and Stasko 1995; or WebNet by Cockburn and Jones 1996). All these demand decision time when searching for the desired candidate amongst other competitors.

It should be apparent that the Back and Forward buttons are far simpler in comparison. They present the page itself rather than a page abstraction, thus avoiding the problem of people having difficulty recognizing the page from its abstract description. They present pages one by one, making the decision a simple choice of deciding if the current page is the one being looked for.

2.5 Summary

In this section, we argued that the Back and Forward buttons have many positive characteristics, especially when compared to other revisitation techniques. First, they are a simple way to return to very recently visited pages, which in practice accounts for the majority of all page revisits a person wants to do. Second, they seem to work well even though people have naïve models of their behavior. Third, they are visually compact, which means that people are likely to keep them on permanent display. Finally, they promote finding a previous page by a simple 'click until the desired page is recognized' strategy. While other strategies may be more effective for finding pages visited long ago, Back suffices for most cases.

3. STACK-BASED BEHAVIOR

The navigation model underlying the Back and Forward buttons found in the two major web-browsers (Netscape and Microsoft Internet Explorer) is based on a stack of visited pages. We will illustrate the stack (and other) behaviors by showing how people can navigate through the small set of pages illustrated in Figure 1.

3.1 Description

The stack has three different types of operations, as illustrated in Figure 2 and described below.

1. Clicking or typing links adds a page to the stack top.
2. Clicking the Back and Forward browser buttons moves the stack pointer down and up the stack respectively, displaying the page at that stack location. The actual stack contents are not altered when navigating with these buttons.
3. When the user is inside the stack (at any position on the stack other than the top) and selects or types a link on a web page, all entries on the stack above the current position are popped off the stack before the new page is added. Pages popped off the stack cannot be revisited using the Back and Forward buttons.

For example, let us say a person follows the page links from pages *a* through *e* in order (Figure 1), then goes back to page *c* by pressing Back twice, and then selects a new link on page *c* to page *i*. We will use the notation $x \rightarrow y$, where '$\rightarrow$' means that the person has selected or typed a link on page *x* to go to page *y*, and $y \Leftarrow x$ which means backtrack from page *y* to page *x* via the Back button.

Figure 2a shows the stack after a person executes $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$, where all pages were added at the stack top. In Figure 2b, the two clicks of the Back button ($e \Leftarrow d \Leftarrow c$) moves the stack pointer down the stack to *c*. Going from $c \rightarrow i$ pops pages *d* and *e* off the stack (Figure 2c), and then adds page *i* to its top (Figure 2d). Thus pages *d* and *e* are no longer reachable through the Back /Forward buttons.

3.2 Advantages and Disadvantages of the Stack Model

The power of the stack technique is derived from the pruning of navigational branches that automatically occurs when users use Back followed by link selection. Essentially, each click of the Back button moves up one level of a tree of navigational branches (Figure 1), and selecting a link from a position within the tree removes the lower level branches. For example, in the navigational trace described above, the other lower level 4 and 5 branches below page *c* disappeared as soon as another child of *c* was selected.

It could be argued that this approach has merits in its relationship to likely user needs: after exploring a branch and selecting a new path of interest the user may no longer need the previous branch of exploration. This is a risky argument, however, which makes assumptions about both

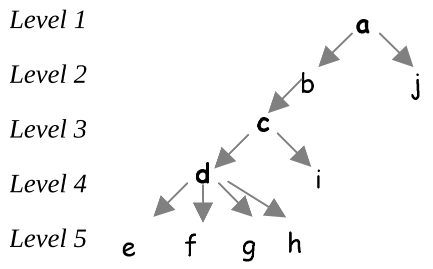

Figure 1. An example page structure. Bolded pages are hubs, all others are spokes.

the structure of the web-site and the needs (and memory capabilities) of the user.

First, there are many cases where people do want to revisit pages seen on an old (and now no longer available) branch. Using the example above, if a person wanted to go back to page *h* from page *i*, they could no longer do it via the Back button as page *h* has been pruned off the list.

Next, we could argue that the Back button isn't really required for this case, because the person can first use Back to go from $i \Leftarrow c$, and then use the normal links to navigate $c \rightarrow d \rightarrow e$. While reasonable for short pages with few links and short navigational paths, this could become onerous for more complex situations. Some pages can be long and complex: recalling and finding the correct link within the page could be difficult. Similarly, if the person followed a complex navigational path to get to a particular page, it could be challenging to remember and/or reconstruct that path later on.

Third, and as discussed in Section 2.2, current systems do a poor job of communicating the tree-pruning behaviour of the stack to its users (Cockburn and Jones 1996). The labels Back and Forward have affordances of linearity, rather than of a tree. There are few cues at the interface to help users distinguish between the underlying semantics of page display using link selection (popping the stack and adding to the new stack-top) versus the semantics of page display using the Back and Forward buttons (moving within the stack). It is not surprising, therefore, that many users periodically find that they have 'lost' pages on the history 'list'.

Summarising the pros and cons of stack-based navigation:

- ✔ It implements a 'simple' interface that requires only two buttons (Back and Forward).
- ✔ The automatic pruning of the stack of previous pages may remove branches and pages that are no longer needed (as when people just want to move 'up' the tree).
- ✔ People are able to use it even with incomplete understanding.
- ✗ Users cannot return to pages that have been popped off the stack.
- ✗ The model is poorly communicated to users through the interface. Consequently, people are surprised when pages 'disappear'.

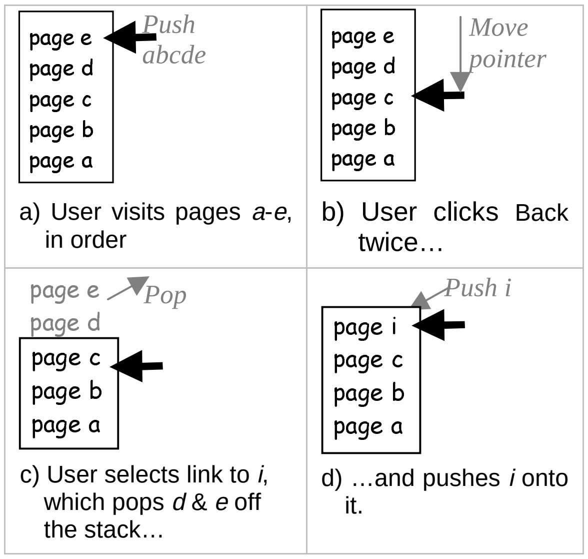
page e
page d
page c
page b
page a
Push abcde
a) User visits pages *a*-*e*, in order
page e
page d
page c
page b
page a
Move pointer
b) User clicks Back twice…
page e
page d
page c
page b
page a
Pop
c) User selects link to *i*, which pops *d* & *e* off the stack…
page i
page c
page b
page a
Push i
d) …and pushes *i* onto it.

Figure 2. An example navigational trace and its effect on the stack. Note that previously visited pages *d* and *e* are no longer available on the stack.

- ✗ It introduces two different semantics for page display: link-selection which pops items off the stack, before adding the new page to the stack top; and page revisitation with Forward and Back which moves through the stack of previously visited pages without altering it.

4. PURE RECENCY-BASED BEHAVIOR

Perhaps the greatest disadvantage of stack-based navigation is that it does not provide a complete history of previously visited pages. As an alternative, we could provide a complete history of previously visited pages by having Back and Forward move a person through a recency-based history list, where the buttons simply navigate through the pages in reverse order to how they were seen. Surprisingly, the design of a button interface to complete history lists is not as simple as might be expected. Although forming a list of all of the pages that a user visits is trivial, designing a simple yet comprehensible interface to list traversal is complex. In this section, we will explore several models of Back based on variants of a recency-ordered history list.

4.1 Pure Recency

Recency-based Back seems like a simple concept. For every page visited, add it to the top of the list. Pressing Back returns to the previous page, adding it to the list as well. Unfortunately, this does not work.

Consider the navigational path $a \rightarrow b \rightarrow c \rightarrow d$. A natural state to expect of the internal history list would be {*a, b, c, d*} where more recent pages are added on the right. Now the person wants to backtrack to page b, and it is reasonable (but wrong!) to expect this to take two clicks: $d \Leftarrow c \Leftarrow b$. On the first click from $d \Leftarrow c$, we return as expected to page *c*. The history list will now be {*a b,c,d,c*}, as *c* has just been visited. When Back button is clicked again, the resulting action is $c \Leftarrow d$ rather than the expected $c \Leftarrow b$. The list is now $\{a,b,c,d,c,d\}$. Subsequent clicks of Back simply cycle between pages *d* and *c*.

The cause of this problem is that we are treating the Back action as a normal navigational act that adds backtracked pages to the history list. This would be good if it worked. Since it does not, we have to treat Back differently, as discussed in the following sub-sections.

As a side discussion, the management of duplicate entries in the history list is an issue that must be addressed. Allowing duplicate pages means that the system can offer a literal representation of the order of pages that the user has seen. The disadvantage is that the list could become unnecessarily long and repetitious. Instead, we suggest pruning duplicate pages by keeping only a single copy of it in its most recent position on the list: this keeps recently-revisited pages near the top. Tauscher and Greenberg (1997) analyzed this, and found that substantially fewer Back presses would be required to return to a desired page when duplicates are pruned. For example, if the user navigates through pages $z \rightarrow a \rightarrow b \rightarrow c \rightarrow b \rightarrow a$, a strict sequential history list would contain $\{z,a,b,c,b,a\}$, while a sequential history list with duplicates retained only in its last position would be $\{z,c,b,a\}$: backtracking to *z* is clearly quicker in the later case. Indeed, this later scheme has been taken up by some commercial browsers: Netscape 4 uses it in their history window (but not with Back). In the remainder of this paper, we will use recency with duplicates removed. Unfortunately, removing duplicates from the history list does not overcome the problem of cycling between the last two pages on the list.

4.2 Recency: Adding Spokes Only

We can modify Back and Forward to move a pointer through the history list. This displays the page at that location but does not alter the list contents. Regardless of the position of the pointer within the list, when the user clicks on a link to a new page, the page is added to the end of the list and any elder duplicates are removed from the list. While similar to how a stack works, the fact that no pages are 'popped off' means that all previously seen pages are available via Back.

However, this scheme is not efficient for *hub and spoke* navigation. To explain, people often visit a central page (a hub), navigate one of the many links to a new page (the spoke), return to the hub, go to the next spoke and so on (Catledge and Pitkow 1995). Example hub pages are home pages, tables of contents, search result lists, and so on. The scheme above, while adequate for simple backtracking, is poor for hub and spoke navigation because the hub moves further and further away from the front of the list as spokes are visited.

Consider, for example, the user actions over the pages in Figure 1 where they move sequentially from *a* to the hub *d*, and then hub-and-spoke browse from page *d* to a series of spoke pages *e, f, g* and *h*. After arriving at *e*, the history list is $\{a,b,c,d,e\}$. The person returns to *d* by clicking Back and then selects the next spoke link to *f*: the history list is now $\{a,b,c,d,e,f\}$. To visit spoke *g*, the Back button must now be clicked twice to return to the hub *d* and link *g* selected, giving a history list $\{a,b,c,d,e,f,g\}$. If the person then wants to go to *h*, returning to hub *d* now requires three clicks. Thus the user's actions to visit all four siblings after first arriving on page *d* are $d \rightarrow e \Leftarrow d \rightarrow f \Leftarrow e \Leftarrow d \rightarrow g \Leftarrow f \Leftarrow e \Leftarrow d \rightarrow h$, making a total of ten user actions. In contrast, the stack-based technique requires only seven actions to visit the same set of pages, as previous siblings were popped off: $d \rightarrow e \Leftarrow d \rightarrow f \Leftarrow d \rightarrow g \Leftarrow d \rightarrow h$. In general, to visit n spoke links from a page, the recency with spokes behavior requires $\sum_1^n i$ user actions while the stack-based mechanism requires only $2n - 1$.

In summary, this scheme introduces one significant advantage and one significant disadvantage not found in stack-based navigation or pure recency.

- ✔ The list of previously visited pages is complete because no pages are popped off the list. Therefore users are guaranteed to be able to revisit pages already encountered during their browsing session by using the Back button.
- ✗ The order of page entries on the history list can be severely different from the order of page visitation. This effect will greatly reduce browsing efficiency in certain navigation tasks, such as hub and spoke.

4.3 Recency with Hub-and-Spoke Enhancement

The next version modifies the previous one to accommodate simple hub and spoke navigation. As before, the Back and Forward buttons just move a pointer through the list. Link selection, however, has different semantics. When the user selects a link on a page, both the current page and the new page are added to the end of the history list. The result is that the nearest hub page is always accessible with one click on the Back button.

Reconsider again the user's actions in navigating from page *a* to *d*, and then to *d*'s four sibling links *e*, *f*, *g*, and *h*. As before, the user navigates from *a* to *e* by link selection giving a history list of $\{a,b,c,d,e\}$. She then returns to hub *d* by clicking Back button once and then selects the link to page *f*. This moves *both* the current page *d* and the new page *f* to the end of history list, giving a history list $\{a,b,c,e,d,f\}$. Similarly, going to pages *g* and *h* only require a single click back to the hub. Thus, the full navigational sequence from *d* to its four sibling links is identical to that of the stack-based paradigm: $d \rightarrow e \Leftarrow d \rightarrow f \Leftarrow d \rightarrow g \Leftarrow d \rightarrow h$. In general, to visit *n* sibling links from a page requires $2n - 1$ user actions. Unlike the stack, however, the person can use

Back to reach other visited spokes by going beyond the hub page.

The additional advantages and disadvantages of this hub-and-spoke enhanced technique in comparison to those of the previous technique are as follows.

✔ Improved efficiency in hub-and-spoke browsing, matching the existing stack-based mechanism at 2*n*-1 user actions to visit *n* links emanating from a page.

✗ True temporal ordering is not maintained. Consider the navigation actions $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \Leftarrow d \Leftarrow c \rightarrow i$. The order of pages seen in the browser display (with duplicates removed) is $\{a,b,e,d,c,i\}$. However, the order of pages on the hub and spoke history list is $\{a,b,d,e,c,i\}$. Longer navigational paths can result in even larger inconsistencies between the contents of the history list and the order in which pages were presented to the user. Thus pressing Back after this sequence does not reflect the true time-based sequence of pages seen.

✗ Navigating back up the tree may mean extra navigation through intervening children. The stack-based approach moved people directly up the tree (i.e., through only parent hub pages), while hub and spoke recency may insert some spoke pages between the hubs.

4.4 Recency with Temporal Ordering Enhancement

As just mentioned, the recency schemes described so far introduce an inconsistency between the order of page revisitation using the Back and Forward buttons and the order in which the pages were most recently presented to the user. We can, however, introduce a new temporal ordering scheme that ensures that the order of revisitation matches the order in which pages have been presented to the user. The idea is that Back and Forward actions move through the history list as before, but that selecting a new link reorders the list to the true temporal sequence. We implemented this technique by maintaining a second pure recency list that traces the order of pages seen when a person navigates the primary history list using Back and Forward. As soon as the user displays a new page (presumably by selecting a page link), the contents of the secondary list are added in order to the main history list. For example, $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ produces the main list $\{a,b,c,d,e\}$. Going from $e \Leftarrow d \Leftarrow c$ creates a second list {d,c}. As soon as the person selects the new link $c \rightarrow i$, *d* and then *c* are added to the main list giving $\{a,b,e,d,c,i\}$ which is the correct temporal sequence. This scheme works over any number of Back and Forward actions.

In summary, the additional advantages and disadvantages of recency with temporal ordering in comparison to hub-and-spoke recency are as follows:

✔ After a new link is selected, Back and Forward will navigate through a temporally correct history list of previously viewed pages.

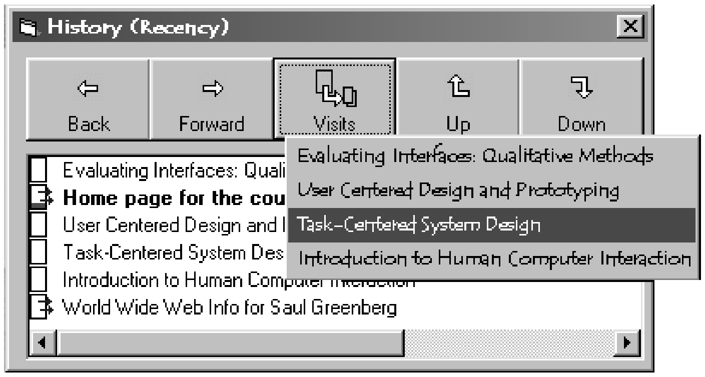
History (Recency)
Back
Forward
Visits
Up
Down
Evaluating Interfaces: Quali
Home page for the cou
User Centered Design and I
Task-Centered System Des
Introduction to Human Computer Interaction
World Wide Web Info for Saul Greenberg
Evaluating Interfaces: Qualitative Methods
User Centered Design and Prototyping
Task-Centered System Design
Introduction to Human Computer Interaction

Figure 3. A Hybrid history system.

✗ This method still combines movement through a list (via Back and Forward) with temporal ordering (via new link selection). There is a risk that users will find the re-ordering of pages on the history list unpredictable. This may not be a big problem, because we suspect that most users will use the Back and Forward buttons in a mechanical fashion, repeatedly clicking the buttons until they recognize the desired page.

✗ As with other recency methods, navigating back up the tree may require extra navigation through intervening children.

5. SUMMARY AND FUTURE WORK

We have articulated the reasons why the Back button is a good interface for revisiting pages. However, we raised questions about the current stack-based behavior underlying Back: while generally effective, problems arise when people use Back only to find that the desired page has 'disappeared'. Given the high use of Back by millions of people, research resulting in even a small improvement in its use is warranted.

Next, we presented several recency-based Back behaviors as alternatives to the stack. Each has both merits and problems, and we do not yet know which behavior (including the stack) is best overall. The next step is to evaluate the effectiveness of each behavior. We need to know how usable they are in various navigation scenarios, and how people react to them.

Of course, we can combine behaviors to produce a hybrid system. For example, we have implemented and are testing a hybrid system that combines stack and recency behavior along with an optional history list (Figure 3). The system controls and responds to the Microsoft Internet Explorer web browser (although shown as an add-on window, the final system would work within the Explorer by replacing its current buttons and by adding a frame). The system optionally displays the recency-ordered history list without duplicates (as shown), allowing a person to select an item directly from the list. The icons on the left indicate whether or not the person followed any links from a particular page (arrows vs. no arrows) and how often a person has returned to that page (icons for frequently visited pages gradually fill with red). The Back and Forward buttons implement recency with temporal ordering (Section 4.4), where the pointer movement is shown by bolding the current item (e.g., the user has gone back to the now-bolded second item in the list "Home page for…"). The Up and Down buttons implement the old stack-based behaviors, with the name change reflecting the fact that the stack moves people up and down the navigational hierarchy. Finally, the Visits button displays a localized history list as a menu. That is, the menu displays in time order all the spokes that have been accessed from the current hub page.

While there is much left to do, we believe research on Web navigation requires further steps Backwards (pun intended).

ACKNOWLEDGMENTS

The work of Shaun Kaasten and Linda Tauscher prodded us to develop the ideas underlying this paper: we owe them our thanks. The National Science and Engineering Research Council (NSERC) of Canada funded this research.

REFERENCES

Ayers, E. and Stasko, J. (1995) Using graphic history in browsing the World Wide Web. *Proceedings of the 4th International World Wide Web Conference. www.w3.org/pub/Conferences/WWW4/Papers2/270/, December 11-14, Boston.*

Catledge, L. Pitkow, J. (1995) Characterizing browsing strategies in the World Wide Web. *Computer Systems and ISDN Systems: Proceedings of the 3rd International World Wide Web Conference.* April 10-14, Darmstadt, Germany, Vol. 27, pp. 1065-1073.

Cockburn, A. and Jones, S. (1996) Which way now? Analysing and easing inadequacies in WWW navigation. *International Journal of Human-Computer Studies* **45**.(1), pp. 105-129.

Cockburn, A. and Jones, S. (1997) Design issues for World Wide Web navigation visualisation tools. *Proceedings of RIAO'97: The Fifth Conference on Computer-Assisted Research of Information*. McGill University, Montreal, Quebec, Canada, June 25-27, pp. 55-74.

Forsythe, C., Grose, E. and Ratner, J. (Eds.) (1997) *Human Factors and Web Development*. LEA Press.

Nielsen, J. (1995) *Multimedia and Hypertext: The Internet and Beyond.* Academic Press.

Rosenfeld, L. and Morville, P. (1998) *Information Architecture for the World Wide Web.* O'Reilly & Associates

Sano, D. (1996) *Designing large-scale web sites: A visual design methodology*. Wiley Computer Publishing.

Tauscher, L. and Greenberg, S. (1997) How people revisit Web pages: Empirical findings and implications for the design of history systems. *International Journal of Human Computer Studies*, Special issue on World Wide Web Usability **47**(1), pp. 97-138.