Focused Web Searching with PDAs

Orkut Buyukkokten , Hector Garcia-Molina , Andreas Paepcke
Digital Libraries Lab (InfoLab), Stanford University, Stanford, CA, 94305
orkut@db.stanford.edu, hector@db.stanford.edu, paepcke@db.stanford.edu

Abstract

The Stanford Power Browser project addresses the problems of interacting with the World-Wide Web through wirelessly connected Personal Digital Assistants (PDAs). These problems include bandwidth limitations, screen real-estate shortage, battery capacity, and the time costs of pen-based search keyword input. As a way to address bandwidth and battery life limitations, we provide local site search facilities for all sites. We incrementally index Web sites in real time as the PDA user visits them. These indexes have narrow scope at first, and improve as the user dwells on the site, or as more users visit the site over time. We address the keyword input problem by providing site specific keyword completion, and indications of keyword selectivity within sites. The system is implemented on the Palm Pilot platform, using a Metricom radio link. We describe the user level experience, and then present the analyses that informed our technical decisions.

Keywords

Personal Digital Assistant, Handheld Computers, Mobile Computing, Palm Pilot

1. Introduction

The benefits of the World-Wide Web can be enhanced enormously if Web content can be made available on handheld Personal Digital Assistants (PDAs) by way of radio links. Frequently, information is needed most when a desktop machine is not available, or long boot times would be disruptive to the task at hand. Examples are information needs that arise while traveling, during business meetings, or in the course of conversations that are awkward to interrupt.

Unfortunately, the advantages in portability and instant availability of PDAs are compromised by the difficulties that arise from bandwidth and screen real-estate limitations. Radio links to PDAs carry nowhere near the volume that other WWW users enjoy on wired Internet facilities. Our link, for example, performs at an average observed throughput of 1.5 KB/sec. Screen sizes, like the Palm Pilot's 160x160 pixels on a 6x6cm surface, pose a tremendous challenge to fruitful exploration of Web resources.

Input facilities are another formidable obstacle in the way of PDA usage on the Web. Typical modern PDAs employ pen-based input with optical character recognition. Even for skilled operators, text entry is a time consuming and error prone activity.

Two basic approaches have been employed to tackle the problems of bandwidth limitations and screen real-estate for PDAs on the Web. The first approach prepares Web pages specifically for use on PDAs. Two closely related examples are the Wireless Markup Language (WML), and the Hand-held Device Markup Language (HDML) that are used to prepare content for mobile clients. Another example is the subset of HTML that is used with Palm VII PDAs [1].

The second approach is to convert regular HTML content automatically for use on PDAs. Systems that implement this approach attempt to present the pages as faithfully as possible. This may involve, for example, sophisticated transformations of images in preparation for display on PDA screens. Examples of systems using this approach are PalmScape [2], HandWeb [3] , Top Gun WingMan [4], ProxiWeb [5] and SnakeEyes [6].

The first approach has the drawback that it limits the amount of available content, and bears the danger of creating two parallel World-Wide Webs. Such duplicate effort could seriously tax human and machine resources.

The second approach does not sufficiently consider the fundamental differences in user interactions when small screens are involved. Every scrolling action becomes significant. Users on small screen do not see the overall information context that is available at a glance to users of large screens, where entire pages can be displayed. Small screens show only a tiny fraction of the information that HTML content designers intend users to absorb at once. In addition to associated costs in time, the consequently required scrolling activities within pages can be very disorienting.

Neither approach addresses the issue of expensive information input modalities. For example, the entry of keywords when accessing Web search engines consumes a significant portion of user-machine interactions on PDAs. In fact, input activity may end up dominating the interaction, and constituting the critical path to completing information intensive tasks. Masui [7] reports on using word completion techniques for accelerating word input on PDAs. But his technique relies on a locally stored dictionary, and is thus not a feasible approach for Web searching.

1.1. Power Browser

Our Power Browser provides an alternative set of techniques for interacting with the Web through PDAs. These techniques are of two categories. The first supports browsing. The second helps users search more effectively. Figure 1 shows how our techniques combine to shorten the time for finding relevant pages.

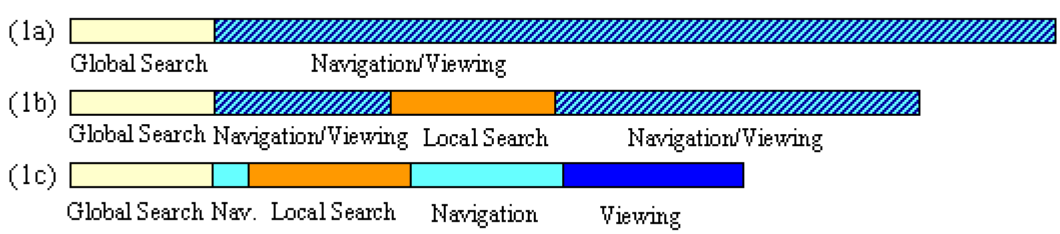


Fig. 1. Browsing Patterns.

Figure 1a shows the typical search pattern on the Web. Users consult a search engine to perform a global search of the Web. Once they receive result sets, they enter into a navigation and viewing phase, which continues until the proper page has been located. Of course, this process might require iterations, which are not represented in the figure. The horizontal axis represents time, although the chart is intended purely for qualitative illustration.

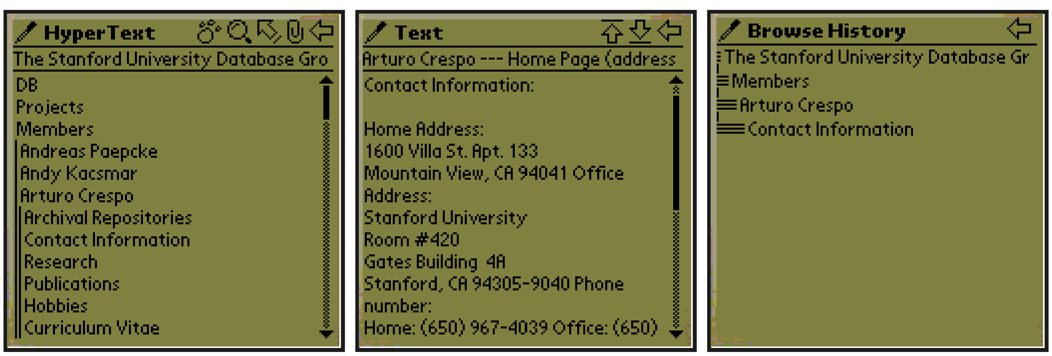
Figure 1b shows an improvement that is available when dealing with some Web sites. The process again begins with a global search, followed by some combined navigation and viewing activity. If the target site provides a site search option, the subsequent local search can shorten the 'final approach' to the correct page. The overall time required in scenario 1b is thus shortened.

Unfortunately, not all sites provide site search, which makes the scenario of Figure 1b a luxury users cannot always count on. Also, the local search option still does not address the cumbersome requirement for entering search keywords on the PDA. Figure 1c shows the pattern our Power Browser attempts to afford. The system shortens the overall interaction time by making the following three main contributions. (The main focus of this paper is contributions two and three, but we include a brief description of the first contribution here for completeness.)

Link Navigation : The first contribution is that the system provides facilities for navigating without having to view full pages. The navigation and viewing phases are therefore shown separately in Figure 1c. This separation is important because the transmission and examination of pages is so expensive. It is therefore wasteful to view content simply to find that one needs to navigate on to the next page.

We accomplish this separation by extracting informative link information from pages. Instead of displaying each page the user requests during the navigation phase, we first show a page summary that presents the links contained on the page. The links are arranged in a tree widget, similar to file folders in a file browser. Users navigate by expanding and contracting nodes. This approach frequently allows users to recognize the page they need to move to next on their way to a final destination. Once they are reasonably confident that they have found the desired page, they switch from the link view to a text view. The text view shows as much as possible of the actual page text.

Figure 2 shows example screen shots. Figure 2a is a link view of the *Stanford Database Group*. Each left justified entry is the description of a link on that root page. The Members link has been expanded to show the links on the Members sub-page. The link to Arturo Crespo's home page has been further expanded. Figure 2b is a text view, showing salient information on Arturo's page. Figure 2c finally, provides an overview of the navigation levels. This is useful when operating at deep nesting levels. The details of the Power Browser's navigation facilities are documented in [8].



(2a) (2b) (2c)

Fig. 2. Power Browser Screen Shots.

Local Site Search : The Power Browser's second contribution is that we provide automated, focused site search even for sites that do not offer such local search facilities. Our site search support is created in real time, and improves with the amount of time users spend on a site. If a user dwells on a site for a long time, or if enough users visit the site over time, the site search facility will eventually be complete.

Local search is an important improvement to the overall search activity, because users often already know which site the desired page resides on, and can thereby eliminate the global search phase altogether, if local search is available. For instance, if users wish to find the latest sports scores, they are likely to start at the Web site of some mass audience newspaper. Similarly, when looking for the academic calendar of MIT, that university's root page is the place to start.

Traditional search engines (TSEs) which specialize on covering a wide selection of sites are not a replacement for local search facilities. Their breadth of coverage prevents them from indexing deeply into most sites. Desired pages located within sites may therefore not be found at all when using a TSE. Even when a deeply nested page is indexed, TSE ranking algorithms often place them far down on their hit list, because major root pages are ranked higher. The consequent scrolling and transmission time requirements are too costly on PDAs. Finally, TSEs cannot keep indexes very fresh. Instead of building voluminous, broad-coverage indexes, we partition indices by site. While this would, of course, make broad-area searches inefficient, this approach allows us to often refresh indexes of frequently visited sites.

Keyword Entry Support : As its third contribution, the Power Browser provides site specific keyword completion to save pen entry time, and approximate gauges for keyword selectivity. This allows users to enter even long keywords by writing just two or three characters on their PDA. As users enter successive keywords, the system informs them in real time of how many matching pages to expect. Users submit their searches for execution only when the keywords they entered have improved the selectivity enough.

Figure 3 shows the top level architecture of the system. PDAs communicate through a wireless network with a proxy server. That server provides the interface to the Web. All compute-intensive activities occur in that server.

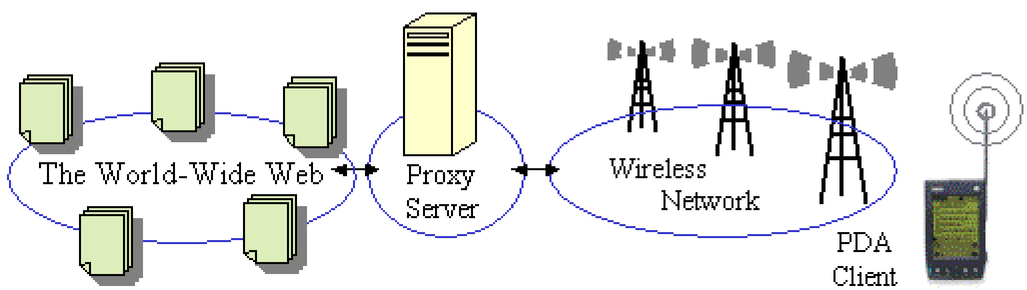


Fig. 3. Browsing Through a Proxy Server.

As mentioned earlier, in this paper we focus on contributions two and three: *local site search*, and *keyword entry support*. Section 2 walks through an example interaction. Section 3 details the technical challenges posed by our approach, and explains the tradeoffs we made. Section 4 summarizes how the components introduced in the paper fit into the overall Power Browser architecture. Section 5 concludes the paper and presents some of the perspectives for the future work on the Power Browser.

2. Example Interaction

2.1. Initiating the Browsing Process

For explanatory purposes, we present an example here to illustrate how the Power Browser works. Suppose our task is to find the academic calendar for the Spring Quarter at Stanford

University. Intuitively, we expect to find this information on the Stanford Web site. Three keywords related to this task are *academic*, *calendar* and *spring*.

The browsing process is initiated through one of three facilities. The user may manually enter a URL by writing on the screen with the pen. Alternatively, the user may select a bookmark (from the box in the bottom half of the screen) or use a search engine to find the URL. In Figure 4, the user wrote 'Stanford University' and then tapped on the Search button to activate a global search engine. The search engine can be selected from the *With* pull-down list. In this example, the user chose *Google* [9]. The Power Browser displays the search engine results as a set of link descriptions (Figure 5a). The user can tap on any of these link descriptions to retrieve the corresponding page. The link *Stanford Home Page: Welcome to ...* is the Web site we are looking for. So we tap on the site search button (magnifying glass icon at the top) and then tap on this link to activate the *Site Search* command.

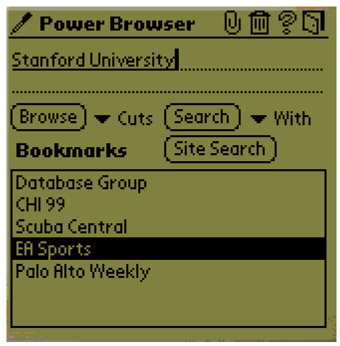


Fig. 4. Starting Browsing.

2.2. Word Completion

The utility of any site search is limited by the expense of two factors. The first one is the time required to enter the keywords on the PDA. The second factor is the time of evaluating the selectivity of chosen keywords without the expense of retrieving and examining the result lists.

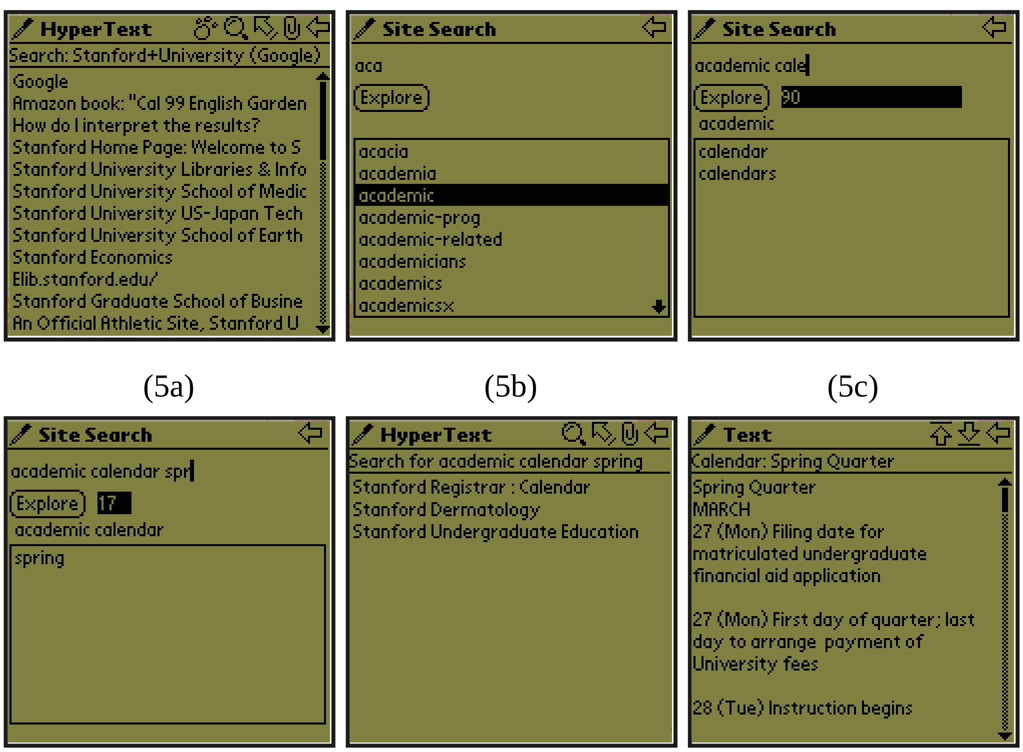
Once the user selects the *Site Search* command, a message is sent to the proxy server to get ready for searching. If the Web site was previously indexed, the proxy server loads the index. Then it starts to update this index by visiting pages that were not visited before. This way, the sites that are visited frequently by the Power Browser users get a more and more complete index. If the index is not available (i.e., the site was not visited before), it is created from scratch.

The user can enter the search words on the text field provided at the top of the display (Figure 5b, 5c, 5d). Once the user starts entering the characters, the Power Browser offers the option of completing the word immediately with *word completion*. Word completion is done in three phases:

Retrieval : After the user enters the first characters of the word, a request is sent to the proxy server to retrieve all the words in the Web site that start with those characters. The result is then displayed in a list widget. In this example, after the user entered *aca* in the search field, all the words that start with *aca* on the Stanford site are displayed in the list at the bottom of the screen. If the list is too long to display on a single screen, a scroll button is provided (Figure 5b).

Pruning : Once the list is retrieved, the user can write more characters and decrease the number of words. Each additional character entered will decrease the number of words that begin with those characters. Therefore the list will start shrinking. This process is done on the PDA since the new list is a subset of the words we obtained in the first phase. After each additional character is entered, all the words in the list that do not start with the given input are removed. The user can use one of the words as a search keyword simply by tapping on that word on the screen. This causes the input to be completed. For example, after the user wrote *aca*, s/he can simply tap on the item *academic* to complete the word in the search field.

Refinement : The user also has the option to enter multiple keywords. This is useful for improving the selectivity of the query. In order for the user to evaluate whether additional keywords are needed, the PDA displays a count of the number of Web pages that contain all the keywords entered so far. This display appears as soon as a keyword is completed (Figure 5c, top). If the last word is not yet entered completely (e.g., *cale* in Figure 5c), the page count includes pages that contain all the previous words (the partially entered characters are ignored). In our example, when the user tapped on *academic*, the number 90 appeared on the screen showing that there are 90 words on the Web site (indexed so far) that have the word *academic* in it. Then the user started entering the second word (*cale*). In Figure 5c, the list consists of all the words that start with *cale* and also occur together with the word *academic* on a page. As the user writes more words, the number of result pages decreases. In this case the user entered the second word *calendar* (either by writing the whole word or tapping on *calendar* on the list). The number 90 drops to 17 (Figure 5d). Similarly, it goes down to 3 when the last keyword *spring* was entered (This screenshot is not shown since it is similar to the previous ones).



(5d) (5e) (5f)

Fig. 5. Example Interaction.

2.3. Search Results

The user can decide to start looking at the search results when s/he thinks that the result set will be small enough. In our example the user decided that 3 results were reasonably few. Notice that until this point no pages were transmitted to the PDA. The retrieval of the titles of all matching pages is initiated by tapping on the *Explore* button. This action activates the Web browser. The browser displays the titles of the pages. While indexing the Web site, the title information is stored together with the index. For example, in Figure 5e we see the titles of the pages that contain all the given keywords. The links are ordered according to the number of keyword occurrences on the page. The user views or navigates through the promising pages. This process is the same as browsing the results of a global search engine on the Power Browser (Figure 5a) with the exception that all the links on the root page are guaranteed to be search results. The user can drill down into pages by expanding/collapsing the tree nodes and viewing the pages by selecting the titles/link descriptions. Figure 5f shows the result of tapping on ’Stanford Registrar: Calendar’ in Figure 5e. The title at the top of the display informs us whether we are looking at the link descriptions (*HyperText*) or are viewing the text itself (*Text*).

3. Challenges and Solutions

Figure 6 shows a summary of the messages that pass between the PDA and the proxy server to realize the site search interactions shown above. There are, however, many details to consider in the design of the overall system. In this section we will go through the obstacles we faced, our solution approaches and the evaluation of our system.

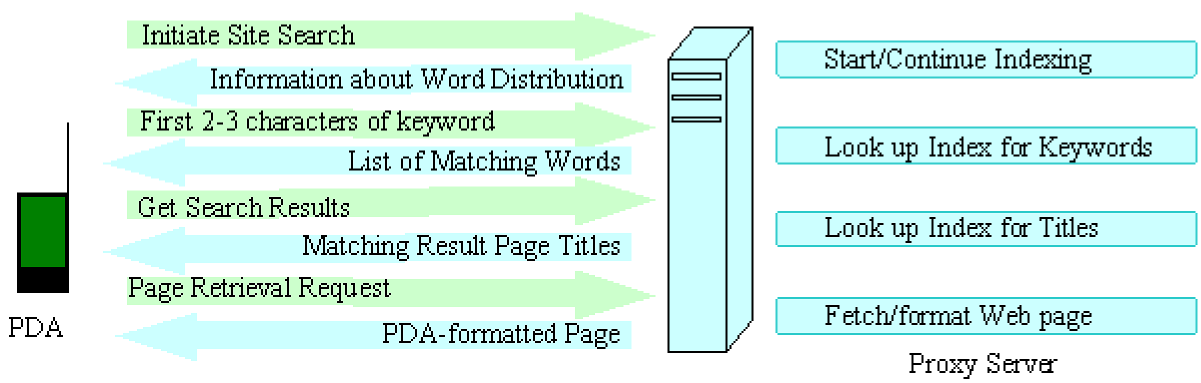


Fig. 6. Messages between PDA and Proxy Server.

3.1. What is a Site?

The quality of a site search depends on the semantic or organizational coherence of the site. We expect the pages that are considered to be in the same site to have a common topic or to belong to the same entity (i.e. organization, company). However it might be impossible to externally determine what makes up a Web site. Multiple groups may share a single Web server, but they might be partitioned into separate directories and use different access control schemes. In other cases, a group may maintain a single logical site that happens to span multiple Web servers. We also cannot determine what Web pages are produced by a particular entity.

There are several different approaches that can be used to approximate the semantic or organizational coherence of a Web site. In all of the following approaches, the perception of a site is a logical notion since the files might be accessed over a series of IP addresses.

- *Same Domain* : All the pages reachable from a given root Web page at the same domain are viewed as a

Web site. The Web site is identified by the domain name of the root page. For instance, if we take *http://www.stanford.edu/* as the root page, all the pages that are reachable from this page and are in the *www.stanford.edu* domain will be considered to be within the same site. Sites like *www-db.stanford.edu* or *cs.stanford.edu* are considered to be in different domains.

- *Same Neighborhood* : Another approach is to start with an initial page and to consider all the pages that are in close proximity. For instance, we can regard all the pages that are at most three links away to be within qualifying close neighborhood. The associated indexing process is complicated since indices of Web-sites may overlap. Solutions to this problem are not discussed here.
- *Same sub-domain* : In defining a site, we can also take advantage of the hierarchy of domain names. For instance *www.stanford.edu* and *cs.stanford.edu* both belong to the *stanford.edu* sub-domain. In this approach, we consider pages that are close in the domain hierarchy (e.g., cs.stanford.edu, www.stanford.edu, www-db.stanford.edu) to be in the same site. That is, we would regard all the pages in the domains *cs.stanford.edu* and *www-db.stanford.edu* to be in the *stanford.edu* site.
- *Hybrid* : Finally, we can apply a hybrid approach where we assign weights to domains. When we initiate our search by *www.stanford.edu* we can give less priority to the pages in *www-cs.stanford.edu* while indexing. This would cause more pages in the *www.stanford.edu* domain to be indexed compared to other pages that are considered to be in the same sub-domain.

In order to understand the implications of the various logical site notions, we examined the content of several sites. Figure 7 shows the data for three of these sites: Stanford, Yahoo, and New York Times Magazine. Starting at the root page of each site, we classified the URLs found into ones pointing to the same domain, into a sub-domain (but not same domain), and outside the site. We then visited the URLs pointing to the same domain, in a Breadth-first fashion, continuing to examine and classify links. The process stopped when 1000 links had been examined at a site.

Looking at Figure 7, we see that 41% of the Stanford links identify pages in the *www.stanford.edu* domain. About 18% of the links point to pages in Stanford sub-domains, such as *cs.stanford.edu*. The remaining 40% go outside Stanford. This is very different from, say the NY Times Magazine, where over 72% of the sampled links go outside the site.

	Stanford	**Yahoo**	**NY Times Magazine**
Pages in same domain	41.1 %	24.8 %	7.8 %
Pages in sub-domain	18.5 %	27.6 %	20.4 %
Remaining pages	40.4 %	47.6 %	71.8 %

Fig. 7. Statistics of Pages Indexed.

The "same domain" site definition may be appropriate for sites like Stanford, where a substantial percentage of the content is local. However, such a definition may be inadequate for a site like Yahoo, where only 25% of the links are in the same domain. Here a "same sub-domain" definition may work better, since it would cover about 52% of the links. For a site like the NY Times Magazine, a "same neighborhood" definition may be more useful, so that more content could be indexed. However, there is a cost associated with the broader definitions. When we crawl pages outside a domain, the number of links to be visited increases tremendously. This results in fewer levels of pages being indexed within the same domain and causes pages that might be completely irrelevant to be indexed. Because of these concerns, in our prototype we initially implemented the *Same Domain* approach, although we plan to experiment with other approaches in the near future.

3.2. Implementing Site Search

As mentioned in the introduction, traditional search engines (TSEs) like AltaVista and Google, have shortcomings when it comes to helping PDA users find pages within a site of interest. In particular, they may not provide thorough enough and up-to-date enough indices. Furthermore, since they index the entire Web, they often return too many hits that are not relevant to a site search.

We have therefore built our own search facility specifically targeted to the needs of PDAs. This facility has three important features: *site-specific indices*, *on-demand crawling* and *incremental index updates*.

Site-specific indices: For each Web site visited, a separate index is created (using the "same domain" site definition). Keeping separate, partitioned indices has many advantages in our context. First, since each index is site-specific, built using the appropriate logical definition of a site, all results returned match the search context. In addition, the entire site index can be read into the proxy’s memory, can be accessed efficiently, and can easily be updated incrementally, as we will see below. Finally, all the vocabulary associated with each site (needed for word completion) can be accessed efficiently since it is in the same index.

On-demand crawling: Our crawler visits pages at a site only when a user is accessing that site. The more users examine the site, or the longer they stay at the site, the more pages are crawled. This means that the resulting index will be more thorough for popular sites, and will dynamically adjust as popularities shift. When a site is visited for the first time, the crawler rapidly visits pages at that site in order to build up the index. Of course, search results will not be comprehensive unless the user waits, but the longer s/he waits, the more comprehensive the index will become. From the point of view of Web sites, they only get visited when real users are examining the site, so it is to their advantage to provide information quickly to meet the demands of a "potential customer." This is in contrast to traditional crawlers, which may place high loads on Web sites, requesting pages that might never be searched.

Incremental indexing: In our system indices are incrementally updated, as new pages are crawled. Users can concurrently access the index, as more data is collected for the visited site. When the user closes his/her session, or starts searching another site, the index is automatically saved for future retrieval. Next time, when a user makes a search on the same Web site, the pre-stored index is used. After loading an index, the crawling process is resumed and the index is again updated incrementally.

Associated with each Web site index, there are three pieces of information. The first one is the *Found Set*. This is a set of links that have been encountered so far in the site. The second one is the *Pending List*, which contains the Web pages that are waiting to be visited and indexed. Finally, there is the *Inverted Index* that contains the vocabulary used in the Web site. It maps words to pages, and records the number of occurrences of each word on each page. The Found Set and the Inverted Index are implemented using hash tables. This data structure provides access in constant time. The Found Set contains the URL of each page and its title. The Inverted Index contains all the distinct words occurring in the Web site crawled so far.

One of the major challenges of our approach is providing good response time to users. (With traditional crawling and indexing, all pages are collected and indexed off-line, before the user makes a request.) For a stored index, we require that it can be loaded into memory in about 3 to 5 seconds. This is the fastest any user will be able to react to the initial search form and enter a few characters for the search. Our system has no problem meeting this lower bound.

If the index does not exist, the crawling speed becomes a key factor. Choosing the order in which to crawl is an important parameter for this process. The *Incremental Crawler* uses a Breadth-first search approach to index Web pages. That is, pages that are reachable through fewer links are indexed first. Fortunately, many sites make important pages reachable from the root via few links, so our Breadth-first strategy works well. With our "same domain" site definition, remote links are ignored by the crawler, speeding up the local traversal.

Multithreaded programming can increase the crawling speed tremendously since it allows the indexer to parse and index multiple pages at the same time. However, in order to keep the database consistent, mutual exclusion must be enforced while updating the indices. This puts a limitation on crawling since more threads will cause more collisions to occur on the indices. To determine the best number of concurrent threads, we tested different strategies, summarized in Figure 8. The figure shows the time it takes to index 100 pages as a function of the number of threads for the Stanford Web site. Based on these results, our system uses 3 concurrent crawling threads. Note that this number is only for a single Web site. There can be multiple crawlers running for each user that accesses different Web sites.

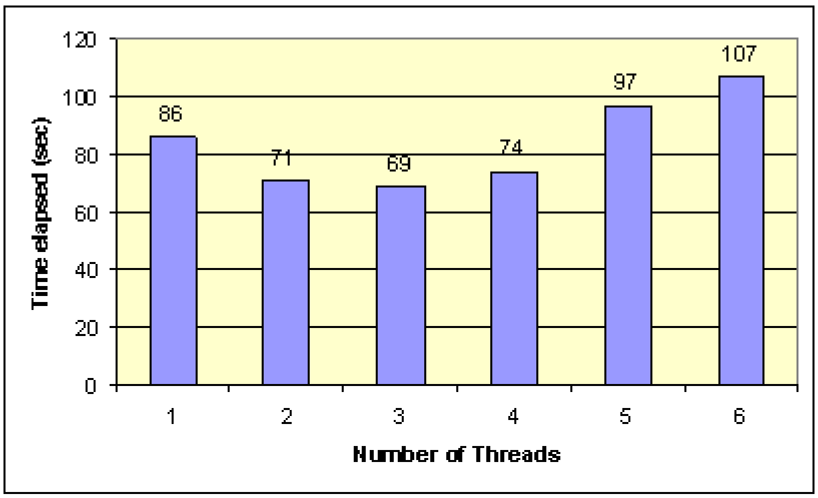


Fig. 8. Indexing Speed.

Notice that the maximum crawling speed should only be used for popular Web sites. The small request interval caused by fast crawling can be very resource consuming for small Web sites. The proxy server adjusts the request interval by looking at the popularity of the Web site's root page, using page rank (Section 4). For small sites it uses a 5 second interval between requests.

Since indexes for the various sites are constructed and maintained as disjoint data sets, multiple browsing sessions are supported easily. This helps with issues of scalability. In order to take advantage of locality, we plan to investigate optimal routing of user browsing requests to proxies that have already started indexing the respective target site.

3.3. Implementing Word Completion

The Power Browser provides keyword completion to reduce pen entry time and effort. After the user writes the first characters of a search term, all the words at that site that start with the characters are displayed. We call the set of words displayed the *matching word list*, and an entry in this list is a *matching word*.

As the user enters characters, the PDA needs to decide at what point (i.e., after how many characters) to request the matching word list from the proxy server. On the one hand, the sooner the PDA requests the matching word list, the sooner it will be able to assist the user. On the other hand, if the PDA requests the list too early, there can be two problems: First, the list of words may be too long. Suppose that the list widget can display 8 items at a time. If 120 items are returned in the list, the user would need to scroll 7 times on the average. This would not be acceptable, and in many cases the scrolling could take longer than entering the word itself (if the word is short). Second, the delay waiting for the proxy to send the matching word list could be too long. If the list is too long, or the words have many characters, the transmission time could be unacceptably high. Thus, the PDA's decision procedure needs to balance these conflicting factors.

As a first step in determining a good decision procedure, we measured the transmission speed of our wireless connection. We observed that the time (in milliseconds) to transmit a message is approximately $500 + 0.65n$, where n is the number of bytes in the message. We use this formula below, in estimating response time given the number of words in a list and their lengths.

In the best case, entering a single character with the pen requires a single gesture. But the user needs to concentrate on the task and, depending on skill, several corrections must be made if the character recognition fails. In contrast, selecting a list item requires multiple pen-taps; scrolling the list and selecting the item. However, the error rate is much lower, and the actions are more mechanical and comfortable to carry out. In order to make word completion beneficial, the user must expend less effort to select the item from the list than entering an additional character. In our studies, we found that 4 taps (3 for scrolling, one for selection) is a reasonable bound on word list interaction. Since the list widget in our system contains 8 items, this translates into a word length limit of 56 words. (With a 56 word list, 8 words can be seen with 0 scrolling taps, 8 others with 1 tap, and so on, for an average of 3 scrolling taps.)

We initially tested word completion on the Stanford University Web site. We then extended our experiments and analyses to other sample sites. We started crawling the Stanford site at the root page, and crawled and indexed 1000 pages in Breadth-first order. These pages contained 32,962 distinct words. Figure 9 shows the average number of words that begin with character sequences of varying lengths.

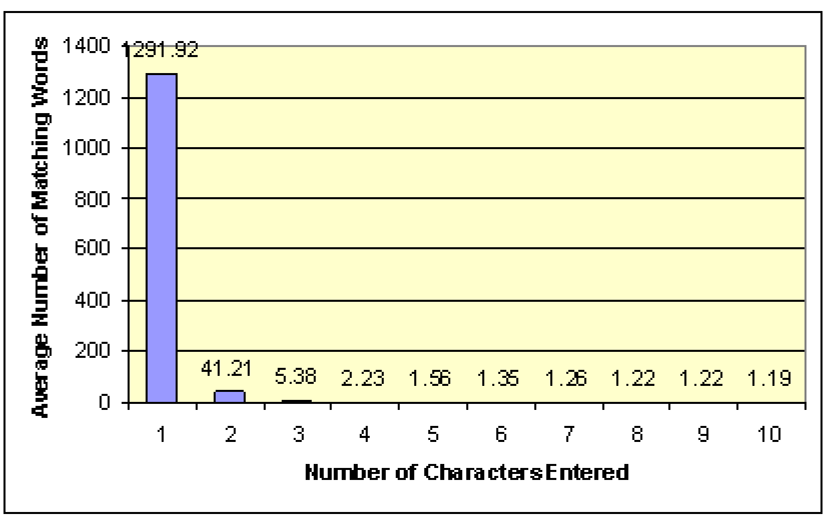


Fig. 9. Number of Words in Matching Words List.

As we can see from this graph, when given just one character, there are too many matching words (1291.92 on the average). Therefore, transmitting the results takes too long (over 7 seconds), and looking at the list to locate a word is too expensive (80 taps on the average). When the user enters the second character, the average number of matching words drops to 41.21. When the user enters three characters, the average size of the matching word list drops down sharply to 5.38.

On the average, therefore, entering two characters discriminates sufficiently to pare the resulting word list to practical proportions. However, in order to design the system to work in all cases, we must understand the distribution of words in more detail. Figure 10 shows the number of words that begin with each three character combination, again using the 1000 pages of the Stanford Web site. (The x-axis labels of this and several following graphs list just a small subset of the total set of prefixes). We see that for all cases, the number of words is acceptably below 56. However, Figure 11 shows a similar graph for the two character case. The variance is much more significant, with *co* producing 844 words, and *re* producing 715, although a large number of cases are still below 56.

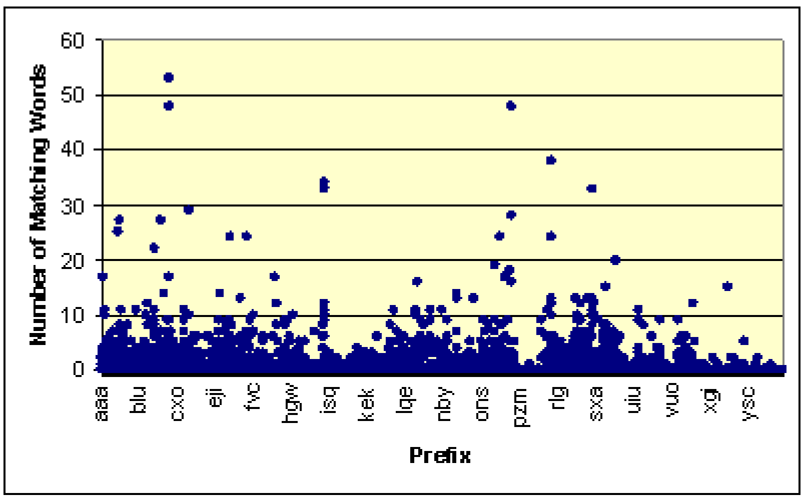


Fig. 10. Number of Matching Words for 3 Characters.

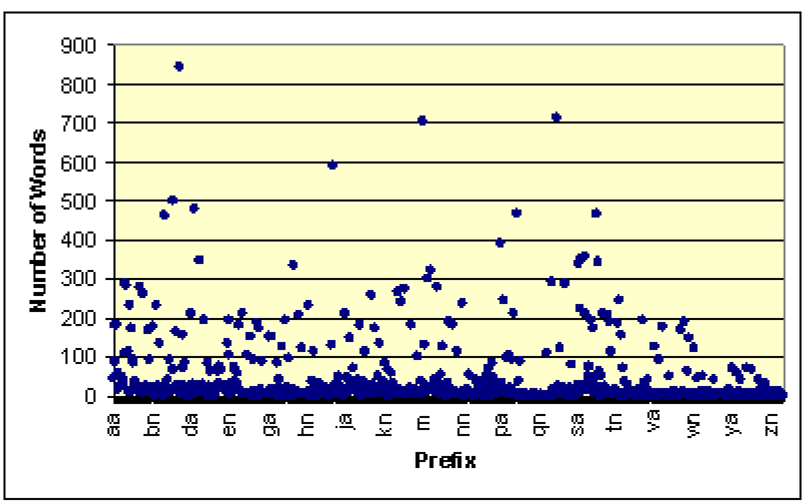


Fig. 11. Number of Matching Words for 2 Characters.

Of course, the number of words to be transmitted is not the only factor that must be considered when deciding on the lists to transmit. The average length and consequent transmission time is another factor. Figure 12 shows the time required to transmit the various word lists for prefixes of length two. Not surprisingly, the transmission time for *co* is more than five seconds, which is too long. We determined that the average word length of each word list never exceeded 12 characters. If we limit the word list length to 56 words, the transmission time would therefore not exceed 1 second (based on the transmission time formula introduced earlier).

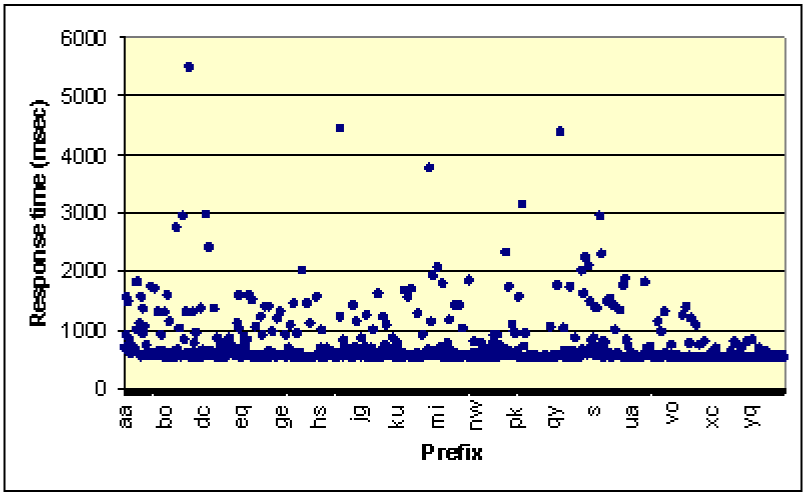


Fig. 12. Response Time for Transmitting Words Starting with Each Combination of 2 Characters.

In summary, for many cases it is sufficient if the user enters just two characters. The resulting word list is short enough to be transmitted quickly, and the scrolling effort for the user is acceptable. In some cases, however, the system should wait for three characters, before a matching word list is retrieved from the proxy server. There are three main approaches that can be used to decide when to wait for three characters. The decision in all these cases is based on the prefix distribution (i.e., the number of words that match the first two characters). In the remainder of this section we will go through these approaches and discuss the tradeoffs.

Resolution through Inquiry : One option is for the PDA to request the matching word list immediately after the user enters 2 characters. If the list is short, the proxy server sends it immediately. Otherwise, the server just sends a *negative acknowledgment* and the PDA requests the list again after the third character. There are two main problems with this approach. First, it will result in an unnecessary round trip delay when the list is too long. Second, the approach results in additional message traffic (which drains client side batteries due to radio transmissions).

Resolution through Feedback : Immediately after the PDA initiates a site search, it receives the word distribution from the proxy server. The proxy can either compute the distribution on the fly, or it can have a precomputed distribution that is incrementally updated as the index changes. On-the-fly computation may be too slow, since the full site index must be read from disk before the PDA is given the distribution. In either case, if an unindexed site is visited, there will be no distribution to rely on, which can be a problem. We also need to decide if the PDA should be notified of changes to the initial distribution it received. Incrementally updating the PDA's cached distribution increases its accuracy, but introduces more wireless traffic and complexity to the protocol.

Resolution through Estimation: In both of the previous approaches the exact distribution of words was used. Given the complexity and increased wireless traffic of those solutions, an alternative is to use a "canonical" distribution that approximates the actual distributions. If we can find such a distribution, then most of the decision making could shift to the client. (As we will see, this is the approach we have initially implemented in our prototype.)

We first study if we can use a single word distribution, that is not updated as the index grows, to make decisions for a given site. For example, say the client earlier has received the distribution for the Stanford site, when 1000 pages were indexed. Say the site index now has 2000 pages. Can the client still use the older distribution?

Figure 13 shows the relative error induced if the old distribution (1000 pages indexed) is used to "guess" how many words have a given prefix when 2000 Stanford pages are indexed. For example, consider the prefix *il*. When 1000 pages are indexed, there are 20 words that start with that prefix, or 0.061% of the total 32962 words. When 2000 pages are indexed, there are a total of 44518 words, so using the old distribution we guess that prefix *il* will have 0.061% of 44518 words, or 27 words. However, in reality there are 41 *il* words, so the relative error is 0.341 (3.41%) (41-27 / 41). In Figure 13 positive numbers mean that the actual word count was larger than the estimate. (A value of 0.5 means that the actual list was twice the size of the estimate.) Negative values mean that the actual list was smaller than expected. (Note that there are many prefixes where the error is -0.351. These correspond to cases where both distributions have exactly the same number of words. Thus, we overestimate by the ratio of the total vocabularies, (32962 - 44518)/32962.)

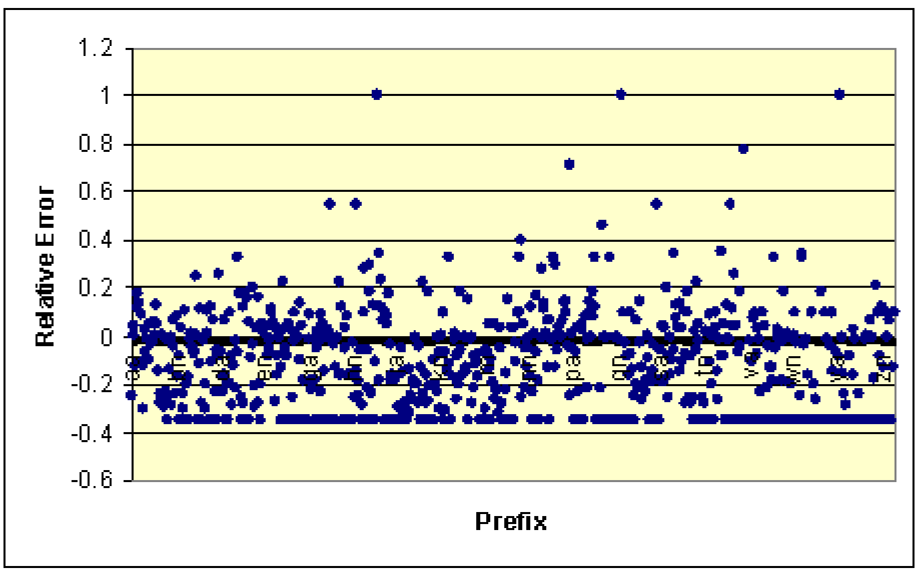


Fig. 13. Relative errors when using 1000 page distribution for 2000 page index (Stanford site).

Figure 13 shows that the errors can be quite large. However, surprisingly we can still make relatively good decisions using the out-of-date distribution. Recall that our goal is not to guess exactly how many words have a given prefix, but simply to decide if the list has more than our threshold 56 entries. Figure 14 shows the actual penalty we pay for underestimating or overestimating the word list, when we use the 1000 page distribution to make decisions when 2000 pages are indexed. For example, as discussed earlier, for prefix *il* we guessed 27, but the actual list has 41 entries. However, in this case both values are less than our threshold of 56, so we did not make a mistake! There is no penalty since even if we knew the correct length, we would have still requested the list after the user entered 2 characters. Thus the figure shows no penalty in this case.

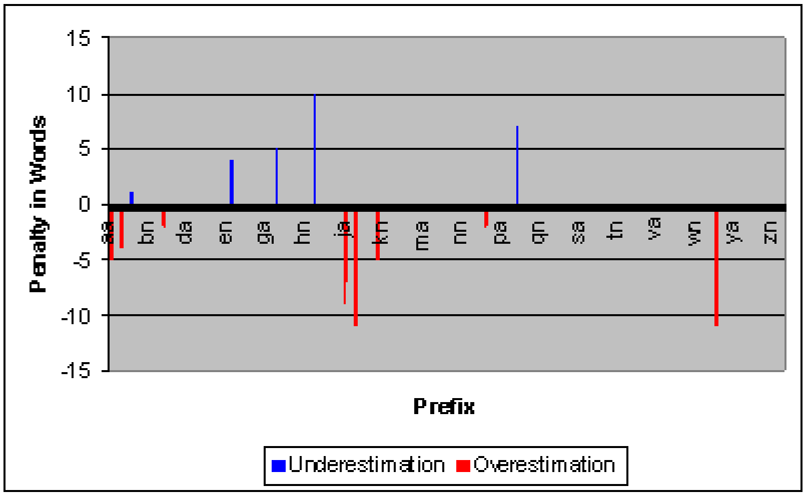


Fig. 14. Penalty for Using 1000 Page Distribution with 2000 Page Index (Stanford Site).

For prefix *gl*, however, our estimate is 55, while the actual list has 61 entries, above the threshold. Thus, we make a mistake and request a list that has 5 (61-56) entries in excess of our limit. For the negative values shown in Figure 14 we overestimate the list length, and do not request a list after 2 characters, even though the list was actually below the threshold. The actual magnitude of the error is not significant in this case, since no list is transmitted.

Figure 14 shows that the actual penalty paid for using an obsolete word distribution is quite low. A penalty is paid (a list is requested too early or too late) for only 2.07% of the prefixes (14 out of 676 prefixes). Figure 15 shows that penalties are still low when the 1000 page distribution is used for a 4000 page index. For 4000 pages, there were 21 errors (3.11% of the cases), compared to the 14 errors in the earlier case.

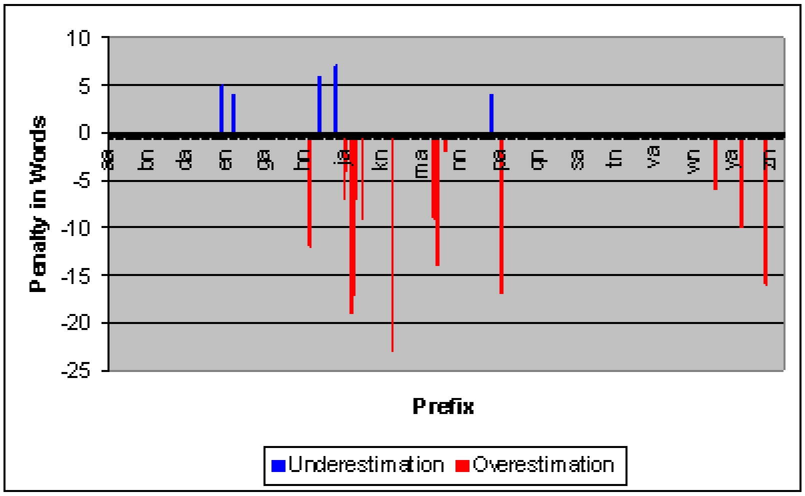


Fig. 15. Penalty for Using 1000 Page Distribution with 4000 Page Index (Stanford Site).

We conclude that a single distribution can be used for relatively good decision making at a single site, for wide ranges of index sizes. Could we go a step further and use a single distribution, say from the Stanford site, to make decisions at other sites? If words are distributed across prefixes in a similar fashion across sites, then a single "canonical" distribution could indeed be used.

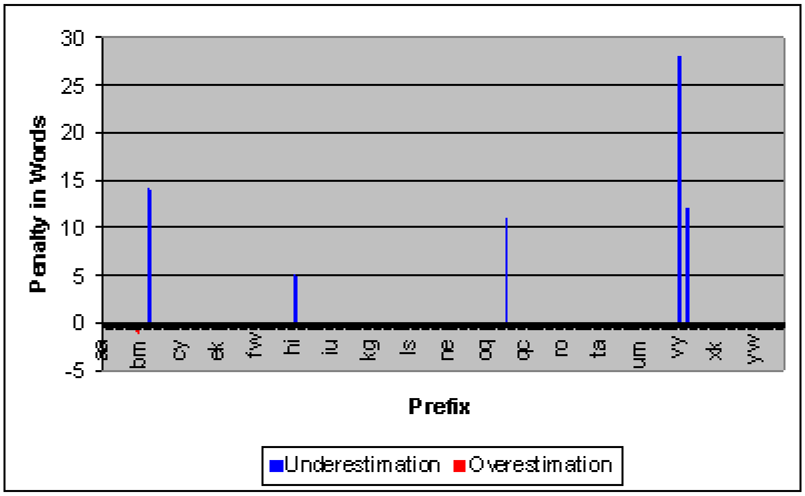


Fig. 16. Penalty for using Stanford distribution for NY Times Magazine.

Our experiments show that word distributions vary considerably across sites (analogous to what Figure 13 shows for changing index sizes). However, a distribution from a foreign site can still be used to make decisions effectively. To illustrate, Figure 16 shows the penalty for using the Stanford distribution at the NY Times Magazine site (both indexes have 1000 pages). An error is made in only 6 cases (0.89%). The worst penalty was for the prefix *wa*, where we estimated 48 words, but in reality there were 84 words (a penalty of 28 words above threshold).

We also studied if the word distribution from the Oxford English Dictionary (OED) could be used for decision making. Figure 17 shows the penalty of using the 112,334 word distribution from the OED to make decisions at the Stanford site. For example, for the prefix *jo*, we estimated 47 words but received 185 words; the penalty is 129 words. Similarly, for the prefix *os* we estimated 59 words, the actual list was 30 words; the penalty is 26 words. We see that the OED is adequate for making decisions. Overall, an error was made in only 39 cases out of 676 (5.77%).

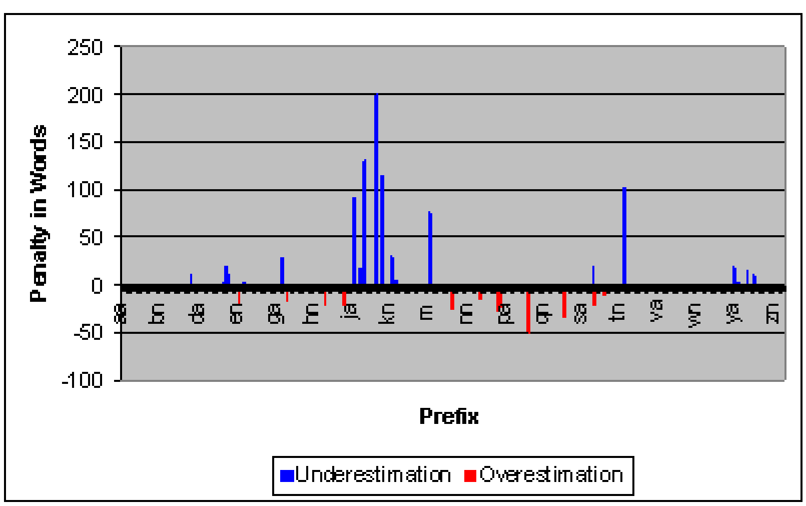


Fig. 17. Penalty for using OED distribution for Stanford site.

In summary, our results show that a canonical distribution can be used to decide when to request a word completion list from the proxy, with acceptable errors. The overall procedure, implemented on our prototype, works as follows. The PDA stores the normalized distribution of the Stanford dictionary (our canonical distribution). After the PDA initiates the site-search, it receives the number of words at the site. When the user types in two characters, the PDA estimates the number of matching words using the canonical distribution. If this number is greater than 56, the PDA waits for a third input character. Otherwise, it requests the matching word list from the proxy. If desired, when the number of indexed words changes, the proxy can send the new number, so that the PDA can have more accurate information. When the server notices that the size of the matching word list requested by the PDA is too long, it can send the new number of indexed words together with the word list and thus avoid an extra message.

4. Implementation

The architecture of the system is summarized in Figure 18. The user browses the Web through an HTTP Proxy server (large dashed box). The proxy server fetches Web pages on the PDA's behalf, dynamically generates summary views of Web pages, and manages the site search facility. The connection between the PDA and the Power Browser Proxy Server is established through a wireless modem.

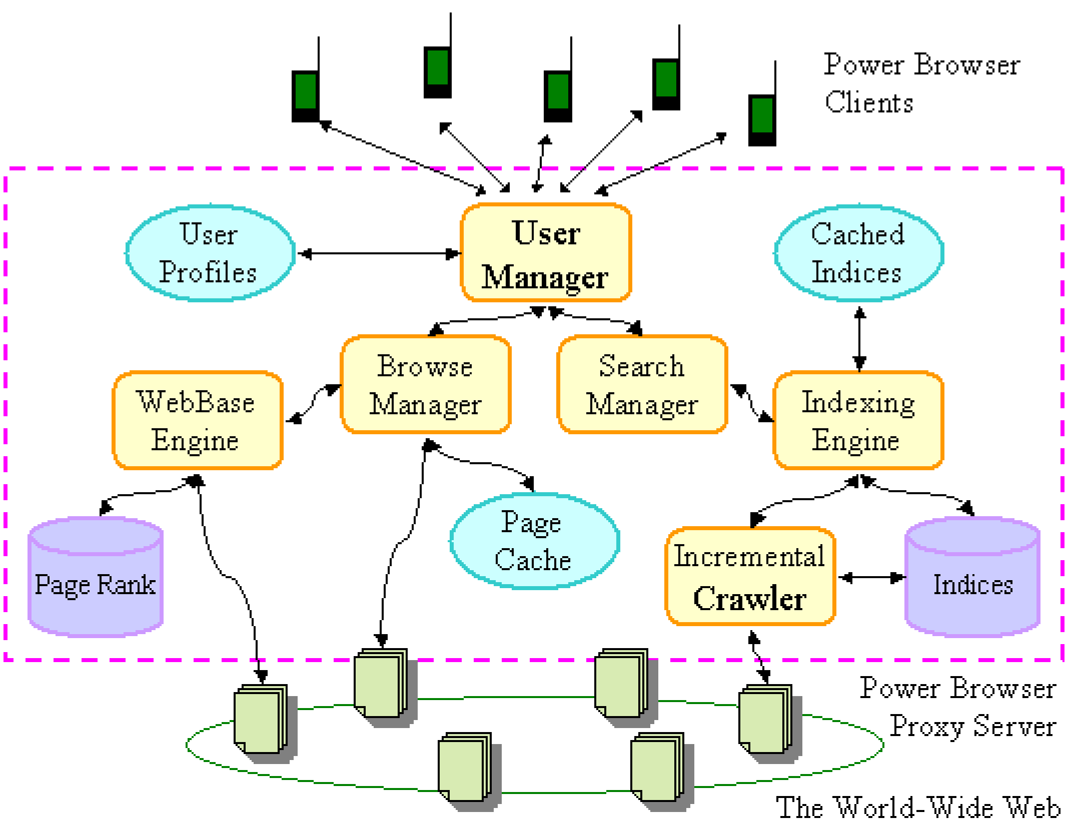


Fig. 18. The Power Browser Architecture.

The components of the proxy server are as follows:

- *User Manager:* The User Manager is responsible for keeping track of the connected users, and for answering user requests. The Manager opens a new session for each PDA and maintains browsing information about the user's activities for the duration of the session (*User Profiles*). The session terminates when the connection between the PDA and the proxy is closed. If the user is browsing or viewing pages, the User Manager initiates the Browse Manager to retrieve the Web page formatted explicitly for the PDA. If the request involves a site search, the results are retrieved through the Search Manager. For each user, a separate Browse Manager and Search Manager process is invoked.
- *Browse Manager:* This component fetches and formats Web pages explicitly for the PDA.
  Fetching : When any module needs to fetch a Web page, it sends a request to the Browse Manager. If the page was already retrieved during the session, the Browse Manager returns the cached version of the result (*Page Cache*). Otherwise it downloads and parses the requested Web document.
  Formatting Pages : In order to reduce the size of the Web pages on the PDA display, white space is avoided as much as possible. The *Browse Manager* collapses sequences of paragraphs or line-breaks. Lists and tables are re-formatted into simple text blocks. Many attributes within the pages, like color, size, font, alignment are ignored.
  Formatting Links : The choice of good link descriptions for each link is made heuristically as follows. If the link has "meaningful" anchor text, we use that text. (Anchor text is what a browser underlines to indicate the presence of a link.) We consider the anchor text to be meaningless (for our purposes) if it is one of the popular content-free expressions such as "click here" or "next." If there was no meaningful anchor text, we instead consider the URL associated with the link. If it points to a directory, we use only the right-most element of the URL. If the URL ends in a file name, we remove the extension, and use the file name.
  Additional Services : The Browse Manager also makes use of our *WebBase* facility. WebBase can provide the *page rank* of a Web page. The page rank of page *p* takes into account the number of Web pages that point to *p*, and in a sense reflects the "importance" of the page on the Web [10]. The *Power Browser* gives the user the option to order the links of a given page using this ranking. This way, links to "important" pages will appear first, potentially reducing the amount of scrolling needed to find links of interest. The list of links can be further improved using different algorithms besides Page Rank ([11]).
- *Search Manager:* The Search Manager is in charge of answering site queries. The Indexing Engine and Incremental Crawler work independently. This allows the system to answer queries even if the index is still being updated.
- *Incremental Crawler:* The crawler fetches Web pages needed by the Indexing Engine. The crawler uses and manages the *Found Set* and the *Pending List*, as described in Section 3.2.
- *Indexing Engine:* The *Indexing Engine* is responsible for answering site queries and providing word completion using the indices. When a site search is initiated through the Search Manager, the indexing engine loads the index if it is available. Then it sends a request to the *Incremental Crawler* to update the index. The Indexing Engine uses the *Inverted Index* to return search results.

5. Conclusion

Web access from radio linked PDAs is an exciting prospect. Innovation in several areas is, however, needed if the use of the Web on PDAs is to be effective. The problems of bandwidth limitations, screen real-estate, and the slow speed of pen-based input stand out among the challenges that must be addressed.

Our Power Browser supports both browsing and focused search activities on PDAs. In this paper we focused on the browser's support for one particular phase of Web-based information retrieval: the exploration of single sites. We described the following related features:

- *A search facility over individual Web sites*. As PDA users visit sites, on-demand crawlers build indices over those sites. The indices grow, as users dwell on the corresponding sites, or as more users come to visit. We explained tradeoffs between different crawling strategies, and presented experimental results to illustrate these tradeoffs.
- *Search word completion*. In addition to supporting search, the site indices are used to help users avoid time consuming pen-based input of search keywords. Once users have entered two or three characters of a promising search word, the Power Browser transmits a list of the keywords that occur on the site to be searched, and that begin with the given characters. We showed through experiments why two, or at most three characters constitute sufficient user input to make the resulting keyword list short enough to transmit and display on the PDA.

- *Selectivity estimates*. As users enter keywords, the Power Browser displays the number of hits that would be retrieved, if the search were to be launched with these keywords. Users can thereby gauge whether more keywords are needed to make the result set manageable, without having to incur the cost of actually launching the search, and examining the result.

This combination of user support mechanisms adds up to significant improvements in Web access through PDAs, especially if combined with the related browsing facilities that were described in [8]. Informal, preliminary user testing has highlighted several scenarios where our facilities are particularly useful.

One example arises when users know the beginnings of a keyword, but do not know the precise spelling. They can just enter the first two or three characters, and can then examine the list of matching keywords to identify the keyword they had in mind.

In a second scenario the user benefits from the selectivity measure. When using relatively generic keywords, such as ’color printer’ on a site like Hewlett-Packard, the selectivity measures can quickly help users decide whether more terms, such as ’inkjet’, might be needed to pare down the result set.

A third scenario occurs when Web sites suddenly spring up. For example, Web sites are now often established to disseminate information about highly publicized, sudden events, such as plane crashes or political scandals. Search engines that focus on covering large portions of the Web will not have indexed such ’flash sites’ when the demand arises. The on-demand crawling that underlies the Power Browser offers much quicker response time.

Of course, there are many improvements we plan to introduce in future versions of the browser. One area to pursue is our definition of ’Web site’. Recall that we currently crawl pages within the same domain as the root page chosen by the user. This approach favors organizational coherence: most pages are likely to be managed by the same institution or company. We plan to explore whether other approaches, like the same neighborhood strategy, are better for some information tasks. In particular, it would be very useful to identify crawling strategies that provide better semantic coherence than our same domain approach sometimes affords.

One problem that occurs with neighborhood crawling approaches is that as crawls from different root pages are launched, their resulting indexes are no longer disjoint as they are in our current system. This can, of course, be wasteful in storage. We will need to explore new storage strategies to manage this issue.

Another enhancement to our system would distribute our proxy server across multiple machines to avoid bottleneck problems as usage grows. A consequence of such distribution would be that indexes for different sites would be stored on different servers. Clients would therefore need to be routed to the correct place. We could use a hash based approach, in which a hash of the root page’s partial URL would identify the proper server. However, this approach would not guarantee effective load balancing, as the sizes of logical Web sites vary. We will need to explore appropriate distribution strategies that satisfy these requirements.

Existing techniques used to improve general web navigation could also be applied to the wireless scenario ([12, 13]).

Searching and browsing the Web from PDAs bears a potential for important productivity gains in how mobile users perform information intensive tasks. The Power Browser introduced in this paper takes one step towards realizing this potential.

References

[1] Palm Inc. Web Clipping Development. http://www.palm.com/devzone/webclipping/.

[2] Kazhuo Oku. Palmscape. http://palmscape.ilinx.co.jp/.

[3] Smartcode Software. HandWeb. http://www.smartcodesoft.com/.

[4] Armando Fox, Ian Goldberg, Steven D. Gribble, and David C. Lee. Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot. In *Proceedings of Middleware '98*, 1998.

[5] ProxiNet. ProxiWeb. http://www.proxinet.com/.

[6] Snakefeet. SnakeEyes. http://www.snakefeet.com/.

[7] Toshiyuki Masui. An Efficient Text Input Method for Pen-based Computers. In *Proceedings of the Conference on Human Factors in Computing Systems*, pp. 328-335, 1998.

[8] Orkut Buyukkokten, Hector Garcia Molina, Andreas Paepcke, and Terry Winograd. Power Browser: Efficient Web Browsing for PDAs. In *Proceedings of the Conference on Human Factors in Computing Systems*, 2000.

[9] Google Inc. Google. http://www.google.com/.

[10] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh World-Wide Web Conference*, 1998.

[11] Massimo Marchiori. The Quest for Correct Information on the Web: Hyper Search Engines. In *Proceedings of the Sixth World-Wide Web Conference*, pp. 265-276, 1997.

[12] Henry Lieberman. Letizia: An Agent that Assists Web Browsing. In C.S. Mellish, editor, *Proceedings of 14th International Joint Conference on Artificial Intelligence*, 1995.

[13] Alexa Internet. Alexa. http://www.alexa.com/.

Vitae

Orkut Buyukkokten is a Ph.D. student in the Department of Computer Science at Stanford University, Stanford, California. He is currently working on the Digital Library project and is doing research on Web Browsing and Searching for personal digital assistants.

Hector Garcia-Molina is a professor in the Departments of Computer Science and Electrical Engineering at Stanford University, Stanford, California. From August 1994 to December 1997 he was the Director of the Computer Systems Laboratory at Stanford. From 1979 to 1991 he was on the faculty of the Computer Science Department at Princeton University, Princeton, New Jersey. His research interests include distributed computing systems and database systems. He received a BS in electrical engineering from the Instituto Tecnologico de Monterrey, Mexico, in 1974. From Stanford University, Stanford, California, he received in 1975 a MS in electrical engineering and a PhD in computer science in 1979.

Andreas Paepcke is a senior research scientist and director of the Digital Library project at Stanford University. For several years he has been using object-oriented technology to address interoperability problems, most recently in the context of distributed digital library services. His second interest is the exploration of user interface and systems technologies for accessing digital libraries from small, handheld devices (PDAs). Dr. Paepcke received BS and MS degrees in applied mathematics from Harvard University, and a Ph.D. in computer science from the University of Karlsruhe, Germany. Previously, he worked as a researcher at Hewlett-Packard Laboratory, and as a research consultant at Xerox PARC.