Google as a Bookmarking Tool

by

Ryan Jazayeri

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by .....
Robert C. Miller
Assistant Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Google as a Bookmarking Tool

by

Ryan Jazayeri

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

In this thesis, a scheme is presented that harnesses the power of Google to locate previously visited webpages. Instead of bookmarking webpages by their URL, this process is based on addressing pages by their *GoogleURL*, which is a short sequence of words that, when queried in Google, locates the desired webpage as the top search result. Good GoogleURLs are memorable and easily communicable, even by speech, and avoid the portability and scalability problems of traditional bookmarks. The GoogleURL Generator—a tool to automatically generate GoogleURLs—is presented, and is complemented with a Google Reordering tool that automatically reorders Google search results based on user browsing history. The goal of this thesis is to employ GoogleURLs and Google Reordering to make Google's "I'm Feeling Lucky" button more lucky.

Thesis Supervisor: Robert C. Miller
Title: Assistant Professor

Acknowledgments

I would like to thank Rob Miller for his excellent feedback and frequent availability for help. I would also like to thank all the other members of the LAPIS group, especially Michael Bolin and Brian Stube, for their insightful comments and ideas throughout the process.

I am especially thankful of the constant support of my parents, Mary and Mehdi Jazayeri.

Contents

1 Introduction **11**

2 Related Work **13**

2.1 Robust Hyperlinks 13

2.2 Personalized Web Search 14

3 GoogleURLs **15**

3.1 Google and PageRank 15

3.2 GoogleURLs versus Bookmarking of Traditional URLs 16

3.3 Characteristics of Good GoogleURLs 17

3.3.1 Short 17

3.3.2 Memorable 18

3.3.3 Robust 19

4 GoogleURL Generation **21**

4.1 Generating candidate GoogleURLs 21

4.1.1 Keyword Ranking 22

4.1.2 Guess Generation 23

4.2 Verifying GoogleURLs 24

4.2.1 Webpage equivalence 25

4.2.2 Google index inconsistency 27

4.2.3 Cost versus effectiveness 28

5 GoogleURL Generator System **29**

5.1 User Interface 29

5.1.1 Guess Table 29

5.1.2 Keywords Field . . . 31
5.1.3 Shorten and Expand . . . 32
5.1.4 Show Results . . . 33
5.1.5 Help . . . 33
5.1.6 Applet security considerations . . . 33
5.2 Design Process . . . 33
5.2.1 Paper Prototype . . . 34
5.2.2 Computer Prototype . . . 36
5.2.3 Standalone Java application . . . 37

6 GoogleURL Generator Evaluation 41
6.1 Pilot User Study . . . 41
6.2 Programmatic Evaluation . . . 44

7 Google Reordering 49
7.1 Algorithm . . . 52
7.1.1 Result Weight . . . 52
7.1.2 Result Ordering . . . 53
7.2 Implementation . . . 53
7.3 Design and Implementation Issues . . . 54

8 Conclusion 59
8.1 Future Work . . . 59
8.1.1 GoogleURL Generator Algorithm . . . 59
8.1.2 GoogleURL Generator Interface . . . 62
8.1.3 Google Reordering . . . 63
8.1.4 Google Bookmarking System . . . 64

List of Figures

5-1 GoogleURL Generator Interface. 30

5-2 Guess Table. 30

5-3 Autocompletion as the user types. 32

5-4 Paper Prototype of GoogleURL Generator. 34

5-5 Demo of GoogleURL Generator Prototype. 39

5-6 GoogleURL Generator standalone application. 40

5-7 GoogleURL Generator standalone application Browser. 40

5-8 GoogleURL Generator standalone application Guess Area. 40

5-9 GoogleURL Generator standalone application Guess Table. 40

6-1 Effect of number of guesses on percentage of GoogleURLs calculated. 45

6-2 Expected relationship between individual Scaling Factor and Yield. 46

6-3 Contribution to yield from each individual scaling factor, using five guesses per webpage. 47

7-1 Google Search results for “news” for Bob. 50

7-2 Google search results for “news” for Alice. 51

7-3 Google Reordering Search Toolbar. 54

7-4 HTML document and corresponding graphical DOM tree representation . . 55

Chapter 1

Introduction

With an ever expanding universe of webpages on the Internet, it is increasingly challenging to locate any particular page. Computer users often wish to return to a page that they have visited before, but have trouble finding it. Bookmarks are one solution to this problem, but have certain usability drawbacks, particularly scalability and portability. An average web surfer cannot be expected to file a bookmark for every page to which he might conceivably return; not only would that require enormous foresight, he would also expend an inordinate amount of time and effort in bookmark maintenance. Consequently, a user will often revisit a page without having bookmarked it. Web search thus inevitably becomes an integral part of locating documents on the web. It is here that this project comes in: making it easier to locate webpages again once they have already been found.

People frequently want to communicate to others where to find a certain webpage. This is commonly done via email, but is also often done by speaking the URL of the page. Traditional URLs, however, are not conducive to being verbally communicated—they are often hard to recall with their slashes and tildes intact. Longer URLs are quite impractical to communicate by speech.

This thesis aims both to facilitate the means by which users communicate webpage locations, and to make it easier for users to find pages they have visited before. The approach to this problem has two parts: the GoogleURL Generator and Google Reordering.

A *GoogleURL*[10] for a webpage is a sequence of words that, when queried in Google, returns that page as the top hit. As an example, "yahoo maps" would be a good GoogleURL for "http://maps.yahoo.com". A good GoogleURL is short and memorable, and is particu-

larly useful as a memorable alias by which to communicate a webpage's location to another person. Remembering a page's GoogleURL will also improve efficiency in finding that page. This thesis presents a tool to automatically generate GoogleURLs, called the *GoogleURL Generator*.

Google Reordering incorporates user information, such as the user's history and preferences, to make search smarter. Based on information known about the user, Google Reordering reorders the results generated by a Google query, bringing to the top the results that are most likely what he is searching for.

On the Google[2] website, the "I'm Feeling Lucky" button, instead of displaying results from the search query, brings the user immediately to the first result of the query. In short, the goal of this thesis is to make the "I'm Feeling Lucky" button more lucky. Both Google Reordering and the GoogleURL Generator contribute toward this goal. When a user wants to go to a specific page and knows its GoogleURL, he can be confident that typing its GoogleURL into Google and pressing "I'm Feeling Lucky" will bring him to the page he wants. Google Reordering can further improve the luckiness of search by making his GoogleURLs more robust.

The rest of this dissertation presents a system for harnessing Google as a bookmarking tool. Chapter 2 describes related work with similar objectives. Chapter 3 explains Google's algorithm, to the extent that it is publicly known. Properties of GoogleURLs are explored in detail, and compared to traditional bookmarking schemes. Chapter 4 illustates the algorithm by which GoogleURLs are generated. Chapter 5 describes the GoogleURL Generator user interface, and discusses issues that were uncovered during its design. Evaluation of the current GoogleURL Generator, by user tests and a programmatic evaluation, is described in Chapter 6. A Google Reordering design is presented in Chapter 7. Finally, Chapter 8 describes future work by which the current Google bookmarking system could be improved.

Chapter 2

Related Work

A variety of related work exists that seeks to help users locate webpages once they have already been found. Robust Hyperlinks[8] creates signatures by which a webpage can be uniquely located, similar to GoogleURLs. Scaling Personalized Web Search[5], an approach similar to Google Reordering, constructs a personalized PageRank vector based on a list of preferred pages.

2.1 Robust Hyperlinks

Robust Hyperlinks[8], by Phelps and Wilensky, proposes to design links so that when a webpage moves or mutates, it can still be located with high probability. The approach of Robust Hyperlinks is to create a lexical signature for each page, which is a string of identifying words in the document, that when fed into a search engine, extracts the desired document and only that document. A *robust hyperlink* for a page is a composition of its URL and lexical signature, such as "http://www.foo.com&lex-signature=w1+w2+w3+w4+w5", where wi is some representative word in the page. The resulting lexical signatures are robust such that they still find pages that are slightly modified but whose keywords in the lexical signature are still representative of the page.

The system includes the lexical signature of a target page as an attribute in a hyperlink. The hope was for browsers to support *robust URLs*, e.g. "http://www.foo.com&lex-signature=w1+w2+w3+w4+w5". Browsers would first strip the lexical signatures from the robust URLs, and the browser would attempt to go to the URL. If the server returned a 404 error, the browser would then search for the lexical signature in a search engine, and

go to the first resulting page.

The lexical signatures generated are similar to GoogleURLs. Both uniquely locate the target page. The main difference is that Robust Hyperlinks are long, for robustness, whereas GoogleURLs are short, for memorability. Differences also include that lexical signatures are hidden from the end user, whereas GoogleURLs are actively remembered and communicated.

2.2 Personalized Web Search

Much work has been done towards personalizing web search. Major search engines are exploring the problem, but none seems have found a successful solution that stands out. At the time of this writing, Google had recently released a Personalized Web Search[3], as a beta. Users must first create a profile by selecting a list of categories in which they are interested. Their search results would then be reordered based on these specified preferences. The user would be able to adjust a bar, controlling how much personalization they want, which then dynamically reordered the search results. Google Reordering fundamentally differs from this approach, because it is based on user browsing history rather than preferences explicitly solicited from the user.

One approach to personalized search[5], by Jeh and Widom, employs a strategy similar to the PageRank calculation, but instead computes a personalized PageRank. This *personalized PageRank vector* is constructed from a set of preferred pages. An importance level is propagated to the pages linked to by the preferred pages. Importance is iteratively propagated to the pages linked to by the previous level. To personalize a search, the set of preferred pages can be chosen specific to the user's interests. In this approach, the set of preferred pages are all set up at the outset, whereas the preferred webpages and domains in Google Reordering adjust dynamically as the user browses. Additionally, in Scaling Personalized Web Search, the set of preferred pages can be from a wider variety of domains. Even pages in domains that the user has never visited will be boosted in value if they are linked to by a preferred page. To contrast, in Google Reordering the set of boosted pages are those in the frequently visited domains, even if pages that are not linked to by a preferred page.

Chapter 3

GoogleURLs

A key idea in this thesis is that GoogleURLs can be used in place of bookmarks, or at least to supplement them, to improve the accuracy and efficiency by which users find webpages that either they, or the person informing them about the page, have visited before.

3.1 Google and PageRank

Google[2] is a very popular search engine, and is generally regarded as the most effective web search engine. Google continually crawls the web and maintains a huge database of the Internet, called the *Google cache*.

Google's algorithm[1] for delivering good search results is mostly proprietary, and is only partially known to the public. One metric that Google uses is *PageRank*[7]. PageRank is a measure of a page's importance on the web, and is proportional to the probability that a random surfer would arrive at that page. Links from other pages are counted as votes for the importance the page.

Given a search query, Google attempts to return results for which the query terms are a good identifier for the cached copy of the webpage. The order of these search results returned to the user is based on a combination of each result's PageRank, and on a measure of how strongly each result matches the query terms.

The exact algorithm by which Google determines how strongly each result matches the query keywords is not public knowledge. For each query, Google calculates a ranked list of search results. In calculating the rank for each result, Google takes into account the following:

- PageRank
- The number of times each word in the query occurs on the page
- Presentation details, such as whether the words in the query are in the title, in bold face, large or small font, whether the words are capitalized, etc.
- Anchor text associated with backlinks to the page, on the assumption that text in links will likely be descriptive of the page linked to.
- Proximity between words on the page.

Google also attempts to recognize when different URLs refer to the same page, and to index a reference to any particular page only once. This issue is discussed further in Section 4.2.1.

3.2 GoogleURLs versus Bookmarking of Traditional URLs

GoogleURLs have several advantages over bookmarks of traditional URLs. A huge advantage of GoogleURLs in general is that they are universal and profile independent. In contrast with bookmarks, a user can walk up to any computer and find a site by its GoogleURL. Any bookmarks in his usual profile are unavailable to him on the new machine, but GoogleURLs are always available. Another strong advantage is that GoogleURLs are easy to communicate to others, especially verbally. URLs can be difficult and inconvenient to speak out loud, because they have to be spelled out and remembered exactly. A good GoogleURL is more memorable and easier to communicate than the URL it refers to. Additionally, a recalled phrase may still be useful even if it is similar to, but not quite exactly, the GoogleURL the user had intended to remember. With a traditional URL, if the user does not get the URL exactly right, then it may be very difficult to find the page. If it is a GoogleURL that the user does not remember precisely, however, he still has a decent chance of finding the page he is searching for. Since many lexically similar queries are GoogleURLs for the same page, the user may potentially even stumble upon a new GoogleURL for the desired page. It is also possible that the page he is searching for is not the first hit, but is within the top ten results, and that the user will then be able to identify the desired page among the displayed results.

Another problems with traditional bookmarks is their lack of scalability. As bookmarks are added, keeping them organized becomes a huge effort. Bookmarks can be archived into folders, but each subfolder can only contain a bounded number of entries and still have each accessed easily. As the number of bookmarks grows and the subfolders become more filled, there may be no trivial way to split folders and still maintain a logical organization.

A disadvantage of GoogleURLs over traditional URLs is that GoogleURLs tend not to be as robust, but this advantage is contestable. In some cases, GoogleURLs can be *more* stable than traditional URLs. When a webpage is taken down and moved to a new location, the old URL may no longer be valid, but, after the new page is indexed by Google, the old GoogleURL might then point to the new page, and thus will still be valid. A bookmark pointing to the old page would no longer work, but the GoogleURL would still be functional. Another disadvantage of GoogleURLs is that pages absent from the Google database have, by definition, no GoogleURL. Some pages that might be desirable to bookmark, such as company internal pages, cannot be bookmarked by a GoogleURL.

3.3 Characteristics of Good GoogleURLs

So that they can be used efficiently and correctly in place of bookmarks, good GoogleURLs have three important properties in common: they are *short*, *memorable*, and *robust*.

Short GoogleURLs can be typed quickly, which is essential for any frequently used shortcut. GoogleURLs should be memorable so that, learning a GoogleURL, a person can recall that phrase some time later when attempting to locate the webpage. Finally, GoogleURLs should be robust, such that they do not become outdated when Google's index is recalculated. The world wide web is a very dynamic entity; webpages go down, new pages spring up, and some webpages even have different content each time they are visited. As Google reindexes its database, PageRanks fluctuate, and any arbitrary page's keyword weightings may shuffle up and down. As a result, GoogleURLs may become outdated without warning, but robust GoogleURLs will suffer from this only rarely.

3.3.1 Short

It is elementary to objectively decide what it means for a GoogleURL to be short. Memorability and robustness, however, are more elusive and difficult to define. Shortness will

generally improve memorability and decrease robustness, but, as described below, not always.

3.3.2 Memorable

It is beyond the scope of this paper to define exactly what a human being finds to be memorable. Human memory has been the subject of much cognitive science research, and aspects of it certainly vary from person to person. In general, however, it can be expected that a group of words that make sense together is more memorable than a group of unrelated words. Human memory works in *chunks*[9], so a group of words that can be remembered as a single chunk will be more memorable than multiple chunks. Similarly, a GoogleURL made up of a group of words that make sense as a group will be more memorable than unrelated words.

Furthermore, with GoogleURLs, the memorability depends partially on the circumstances in which the user learned the GoogleURL. For example, let us say that Alice has in her mind that she is attending a lecture on nuclear fission. The lecturer wants the audience to be able to visit his research website later on. He fears that no one will remember the URL, so the lecturer tells the audience a GoogleURL by which they can find his website. For Alice, the GoogleURL would be most memorable if it contained the phrase "nuclear fission". But let us say someone else at the lecture, Bob, only knew that he was attending a lecture of the "Atomic Lecture Series." For Bob, a GoogleURL containing the term "nuclear fission" might not be memorable at all! Furthermore, memorability has no clear relationship with how short the GoogleURL is. Shorter GoogleURLs tend to be more memorable than very long GoogleURLs, but the shortest GoogleURL possible is not necessarily the most memorable. For example, the GoogleURL "boston public library", for the website of the Boston Public Library, http://www.bpl.org, tends to be more memorable than the shorter GoogleURL "bpl". But often shorter GoogleURLs are indeed more memorable. The GoogleURL "amazon", for www.amazon.com, is indisputably more memorable than the GoogleURL "amazon music boosktore shopping site".

Another factor affecting memorability of GoogleURLs is how memorable each individual term within the GoogleURL is. Ideally, the whole GoogleURL can be distilled into a single memorable chunk, but otherwise, each word in the GoogleURL needs to be memorable as its own separate chunk.

Single terms composed of multiple words (without spaces) come up often in GoogleURLs, because they occur in the URL itself. For example, for the URL "www.thenews.com", "thenews" is a GoogleURL but "the news" is not. Users therefore have to remember whether the term is one word or two words, which detracts from memorability and communicability.

3.3.3 Robust

Robustness of a GoogleURL is difficult to define because it has dependencies on the entire contents of the Google database. Call a *static word* a word on a webpage that will be continually present on that page, and call a *dynamic word* a word that is present at some point but will not always be present. On first thought, one might assume that the longer the GoogleURL, the more robust it will be, but this is not the case. The difficulty in evaluating robustness reduces to determining which keywords on a webpage will be static and which will be dynamic. In general, adding static words increases GoogleURL robustness, whereas adding dynamic words decreases GoogleURL robustness. At the time of this writing, MIT's "Tech" newspaper, www-tech.mit.edu, had a GoogleURL "mit tech", although few months earlier, "mit tech" had been a GoogleURL for MIT's Technology Review magazine, located at www.techreview.com. Adding an additional term to the GoogleURL "mit tech" may or may not make the GoogleURL more robust. Extending the GoogleURL to "mit tech newspaper", which is also a GoogleURL, achieves a marked improvement in the robustness. The improvement comes about because "newspaper", the keyword added, is a static word on the MIT Tech's website. If, during the month of April, in hopes of increased robustness we further expand the GoogleURL to be "mit tech newspaper april", this will hurt the robustness. Once the month is no longer April, and the page is reindexed in Google, "mit tech newspaper april" may cease to be a GoogleURL!

Chapter 4

GoogleURL Generation

With access to Google's database and PageRank information, it would be possible to develop an efficient and reliable scheme for determining good GoogleURLs. Without such direct access to the database, however, the only way to generate GoogleURLs is to make informed guesses. Even simulating Google's algorithm to calculate PageRanks, and building a database of one's own, is still not sufficient to produce a completely reliable algorithm for GoogleURL generation. Certain aspects of Google's process are nondeterministic, and depend, for example, on the order in which the Internet is archived. Regardless, Google's algorithm is proprietary, so this algorithm seeks to reverse-engineer it.

Define the *target page* as the webpage for which a GoogleURL is to be generated. The process by which the GoogleURL Generator generates GoogleURLs for a target page consists of two stages. The first stage is to generate plausible guesses for what might be GoogleURLs for the target page. The second stage is to verify which of those guesses indeed are GoogleURLs. These are described in more detail in the following sections.

4.1 Generating candidate GoogleURLs

The algorithm generates an ordered list of potential GoogleURLs. To do so, the algorithm first examines the target page and its backlinks, and calculates a ranked list of present keywords. Each keyword is given a rank according to its prominence on the page. The list of candidate GoogleURLs are created based on those keyword ranks and on repeated phrases in the page.

The GoogleURL Generator currently examines the live copy of the webpage in the

Google cache to create a GoogleURL that works well at the present point in time. There remains the possibility that dynamic keywords had been used in the GoogleURL, resulting in poor robustness. Future improvements to the GoogleURL Generator can remedy this problem by also examining the Google cache, as described in Section 8.1.1.

4.1.1 Keyword Ranking

The target page is parsed to determine a list of the words on the page. Call any word present on the page a *keyword*. HTML parsing is performed using LAPIS[6], a tool for structural text processing.

For each keyword, a corresponding *keyword weight* is calculated. Intuitively, a keyword's weight measures how well that keyword works as an identifier for the target page. Ideally, the calculated keyword weight corresponds to how well the keyword distinguishes the target page from the other pages in Google's database. The system therefore attempts to emulate Google's ranking algorithm in calculating the keyword weights.

For each keyword on the target page, a record is kept of how many times it occurs in each *context*: namely in boldface, in headings, in capitalization, in the title, as a token in the URL, in simple plain text, in outgoing links on the target page, and in incoming links from backlinks.

The top *backlinks* to the target page are also examined, keeping track of each word that occurs in the links pointing to the target page. The term *backlink* refers to a webpage that has hyperlinks pointing to the target page. It is possible to query Google to obtain a list of backlinks to a particular website.

For each possible context c there is a corresponding *scaling factor* S_c, which remains constant over all webpages. For example, there is a *boldface scaling factor*, and a *capitalization scaling factor*. The boldface scaling factor determines how much extra weight to assign each time a word occurs in boldface.

For each keyword, the number of occurrences in each context has been recorded. For each keyword-context pair, a dampening function is applied to the number of occurrences to obtain an *occurrence factor*. The dampening function currently used to generate the occurrence factor of keyword k in context c is simply $min(5, n_{k,c})$, where $n_{k,c}$ is the number of occurrences of k in context c. The rationale behind applying a dampening function, instead of just using the number of occurrences, is so that a keyword that occurs repeatedly

in the same context has a bound on how much it can contribute to the weight. More importantly, Google employs a similar practice. In order for this algorithm to effectively reverse engineer Google's algorithm, it is important to use a dampening function.

The keyword's weight w_k, for keyword k, is determined by the equation below:

$$w_k = \sum_{c \in C} S_c min(5, n_{k,c}) = \sum_{c \in C} S_c F_{k,c}$$

where C is the set of all contexts, S_c is the scaling factor for context c, and $F_{k,c}$ is the occurrence factor for keyword k in context c.

4.1.2 Guess Generation

A *guess* is a sequence of keywords that can serve as a Google query string. Each possible guess is evaluated to produce its *guess value*, which predicts how good a GoogleURL it is for the target page. What exactly it means for a guess to be good is explained in more detail at the end of this section. A guess value is dependent on the weights of the keywords throughout the page, and on the order of the keywords in the guess.

A stop list of extremely common words, such as "the" and "and", are filtered out and not used in any guesses. Google ignores these words present in its queries, so there is no benefit of including them in guesses.

The system attempts to recognize phrases on the target page. The importance of phrases is twofold: for one, Google includes proximity information in its database, and attempts to return pages in which the words in the query occur near each other. Additionally, phrases tend to be memorable as a single chunk, so phrase detection provides a means by which a machine can generate GoogleURLs that are likely to be memorable.

To harness this phrasing information, for each keyword the algorithm keeps track of the number of times each other keyword occurs directly after it. Call $NF(k_i, k_j)$ the *neighbor factor* for two keywords k_i and k_j. Define $NF(k_i, k_j) = \min(5, Follow(k_i, k_j))$, where $Follow(k_i, k_j)$ is the number of times that keyword k_j occurs immediately after keyword k_i in the target page or its backlinks. The neighbor factor is therefore a measure of the number of times a particular sequence of two keywords occurs after each other in the target page or its backlinks. There is an additional constant *neighbor scaling factor* that measures how much the neighbor factor contributes toward the guess value. The neighbor scaling factor

is, essentially, how important phrases are in generating GoogleURLs.

Google considers keywords placed at the front of the search query to be more important. When calculating guess value, the algorithm biases it towards the weight of the keywords earlier in the guess. Consider a guess g composed of a sequence of n keywords. The guess g can be represented as $[k_1\ k_2\ ...\ k_n]$, where each k_i is a word within the guess. The value v_g of guess g is calculated as follows:

$$v_g = \frac{2}{n(n+1)} \sum_{i=1}^{n} (n+1-i) w_{k_i} + S_{neighbor} \sum_{i=1}^{n-1} NF(k_i, k_{i+1})$$

where w_{k_i} is the weight of the i^{th} keyword in g, and $S_{neighbor}$ is the neighbor scaling factor.

As an example, consider the guess "boston public library". Here, $g = [boston\ public\ library]$, whereby k_1 is *boston*, k_2 is *public*, and k_3 is *library*. Assume that in the target page, "boston" precedes "public" exactly 4 times, and "public" precedes "library" exactly 7 times. In this case, the guess value for "boston public library" will be calculated as follows:

$$\begin{aligned} v_g &= \frac{3w_{boston} + 2w_{public} + w_{library}}{6} + S_{neighbor}(NF(boston, public) + NF(public, library)) \\ &= \frac{3w_{boston} + 2w_{public} + w_{library}}{6} + S_{neighbor}(4+5) \\ &= \frac{3w_{boston} + 2w_{public} + w_{library}}{6} + 9S_{neighbor} \end{aligned}$$

Note that each guess value is *not* a measure of how likely it is that the guess will be a GoogleURL for the target page. If that were the case, then the guesses with the most keywords would be the highest ranked guesses. As described above, a guess's value also encapsulates how short and memorable the guess is predicted to be. If they are successfully validated, the guesses generated are expected to be short and memorable. In the absence of phrases, the shortest potential guesses will be the highest ranked, and should be memorable because of their shortness. Longer guesses are only tried if they are in frequently occurring phrases in the document, which would likely be memorable identifiers of the page.

4.2 Verifying GoogleURLs

To test if a guess string is a GoogleURL for a target webpage, the guess is queried in Google to check whether the first result is the same webpage as the target page. This would all

be simple and straight-forward, but is complicated by URL aliasing and the problem of determining when two different URLs address the same webpage. This issue of webpage equivalence is described below.

4.2.1 Webpage equivalence

Some webpages can be addressed by multiple URLs. Each Google search result, however, contains only one URL. Define the *canonical name* of a webpage as the URL having the highest PageRank by which the page is indexed in Google. When multiple addresses of a webpage are among the search results, then the vast majority of the time the canonical name will be the highest ranked of those addresses. Rare instances when an *alias* can be ranked above its canonical name are explained later in this section. Note that the term *canonical name* is not necessarily the same as the DNS Name for the webpage. Define *synonyms* as all the URLs by which a certain webpage can be addressed.

If the user loads a page aliased by some synonym other than the canonical name, then the GoogleURL Generator runs into trouble validating guesses. Consider, for example, if the user enters the target page "www.americanairlines.com" into the GoogleURL Generator. At the time of this writing, a Google search for "american airlines" yielded the synonyms "www.aa.com", "www.americanair.com", and "www.americanairlines.com", all within the top ten search results. In this case, the URL "www.aa.com" had the highest PageRank, so is the canonical URL for the American Airlines website. The GoogleURL Generator will generate a set of guesses, such as "american airlines", that yield "www.aa.com" as the top hit, instead of the URL "www.americanairlines.com" that the user had loaded. At this point, the GoogleURL Generator cannot recognize that "american airlines" is in fact a GoogleURL, because it cannot figure out that "www.aa.com" and "www.americanairlines.com" are synonyms.

Synonyms can take several different forms:

- *DNS Alias*: The DNS server stores a DNS name for a URL, as well as any number of DNS aliases for that URL. When going to any of the DNS Aliases in a browser, the webpage is displayed but the browser's URL does not refresh to the DNS Canonical Name. It is possible to query the DNS server to determine what DNS name that a URL is an alias for. It is not possible, however, given a URL, to query the DNS for all aliases for that URL.

- *Permanent Redirect Alias*: Permanent redirects are issued by the webserver, and the browser's URL changes correspondingly when redirected.

- *Server-level Alias*: The redirect occurs at the server level, and the browser's URL field is not updated. The redirect does not need to be registered with the DNS, so one cannot easily determine the DNS Name of the page that is actually displayed. Consider a web server hosting the two URLs X and Y. At some point in time, the server may return the same content for accesses to both X and Y. It would be logical to conclude that X and Y are synonyms, because they both apparently cause the browser to display the same content. Nevertheless, the web server may at any point decide to handle the two URLs differently, and serve up an entirely different page when Y is accessed. Server-level aliases are therefore less reliable than permanent redirect aliases and DNS aliases, but are still considered synonyms.

Other complications involving URL synonyms include the following:

- URLs are sometimes case sensitive. The hostname portion of the URL is always case-insensitive, but the rest of the URL may be case-sensitive. As an example, "web.mit.edu/jaz/www/googleURLHelp/" works, but "web.mit.edu/jaz/www/googleurlhelp/" does not. The two could, in fact, be different webpages entirely.

- A "www" preceding one of two otherwise identical strings, does not necessarily imply that the two strings are synonyms. For example, "www.student.mit.edu" and "student.mit.edu" are not synonyms, but "yahoo.com" and "www.yahoo.com" are.

- index.html: "http://www.google.com/index.html" and "http://www.google.com" are synonyms, but "http://www.amazon.com" and "http://www.amazon.com/index.html" are not. This can occur, for example, in instances when the URL is a redirect, so appending "index.html" onto the end does not create a synonym.

- Relative URLs: links to a page within the same domain are often specified relative to the referencing page. For example, a link from the source of the page "web.mit.edu" can refer to "web.mit.edu/site/aboutsite.html" either as the absolute URI, "web.mit.edu/site/aboutsite.html" or as a relative URI, "site/aboutsite.html". Furthermore, relative URLs can point to a location that issues a permanent redirect, which cannot be determined just by lexically comparing the URLs.

Determination of webpage equivalence—determining whether URL X and URL Y are synonyms—is a difficult problem. While it may seem intuitively easy for a human to decide that two URLs are synonyms, this equivalence is difficult to quantify reliably. Webpages on the Internet can have dynamic content, even if, on any particular access, the webserver serves the page to the web browser as a static page. When a surfer visits a page, the content displayed in his browser is entirely dependent on the webserver. The display can systematically depend on factors such as the time of access or personalized state like cookies.

Google attempts to recognize when URLs are synonyms and to index these aliases together under one name. Two URLs that are correctly associated together in Google's index are "graphics.lcs.mit.edu" and "graphics.csail.mit.edu". In this case, "graphics.lcs.mit.edu" happens to be a DNS alias for "graphics.csail.mit.edu", but "graphics.lcs.mit.edu" is the URL by which the two are indexed.

Google's determination of webpage equivalence is far from perfect, however, and there are many occurrences where multiple synonyms appear in the Google index. For example, the American Airlines website mentioned earlier has multiple distinct records in Google. Since these URLs have extremely similar content in Google's cached database, the page with the highest PageRank is nearly always the highest ranked search result of these three pages. Complications could arise, however, if in a page with dynamic content the cached versions of the synonyms differed significantly from each other. Some search query might potentially yield a non-canonical name ranked above its corresponding canonical name. Furthermore, PageRanks fluctuate up and down as Google recalculates its index. Therefore, the canonical name is not absolute, and can be employed only as a useful heuristic.

The current version of the GoogleURL Generator has very limited detection of webpage aliasing. The simplest lexical aliases, differing just by "www", "http://", "index.html", or a trailing slash are assumed to be equivalent. Even though this equivalence may not always hold, the vast majority of the time the assumption is valid.

4.2.2 Google index inconsistency

Since the Internet is such a dynamic system, Google frequently crawls the web to try to keep its database up to date. Obviously, Google's database cannot reflect the real world wide web perfectly. Common examples are pages that are down or pages that are simply not in Google's index. If a webpage is composed of entirely static content, then there will

be no difference between the webpage and Google's cached copy of it. For pages with dynamic content, however, there will inevitably be discrepancies between the webpage and the indexed copy. It would be possible to examine Google's cache, and take advantage of this discrepancy to detect dynamic content, but currently the GoogleURL Generator does not do so.

4.2.3 Cost versus effectiveness

In generating GoogleURLs, three conflicting goals must be balanced: minimizing time taken, minimizing the number of Google queries, and maximizing the quality of GoogleURLs produced.

The user's time is valuable, and it is essential that the GoogleURL Generator is able to generate GoogleURLs fairly quickly. Each query to Google is expensive for two reasons. First, the number of queries to the Google Web APIs are limited to 1000 per day. Second, each query takes valuable time. It is necessary to put some bound on the number of guesses the system will be allowed to automatically make. The current system performs five guesses automatically each time a target page is loaded. Further guesses must be requested by the user explicitly.

Another parameter is the number of backlinks to be examined. Examining backlinks is valuable for generating a good GoogleURL for some pages, but was found to be only a marginally useful part of the GoogleURL Generation process. Examining backlinks was unreliable—at times, backlinks would be down, or would take a very long time to load. At some point, it makes sense to decide to stop waiting for each backlink to load. Since processing each backlink takes a significant amount of time for usually minimal benefit, the GoogleURL Generator limits the number of backlinks examined to two.

Chapter 5

GoogleURL Generator System

The GoogleURL Generator is a tool that generates GoogleURLs for webpages. The current interface, described further in Section 5.1, is an applet embedded in a side frame of a webpage. The process by which the GoogleURL Generator user interface was designed is illustrated in Section 5.2.

5.1 User Interface

The GoogleURL Generator attempts to automatically generate GoogleURLs of a target webpage. If the user is not satisfied with the GoogleURLs generated, or if the GoogleURL Generator was unable to generate any GoogleURLs, then the user can then interact with the system to generate better GoogleURLs. This user interface consists of several mechanisms to help the user create better GoogleURL guesses, and is shown in Figure 5.1.

After launching the GoogleURL Generator, the user enters the URL of the target page into the *URL field* combo box. After pressing "Enter" or "Go", the GoogleURL Generator will display the target webpage in the frame on the right. The system then creates and validates guesses for GoogleURLs for the target page, by examining the target page and the backlinks as described in Chapter 4.

5.1.1 Guess Table

Once the automatically generated guesses have all been queried, the results are displayed in a *Guess table*, as shown in Figure 5-2. The Guess table consists of two columns, a *Keywords* column and a *Search Rank* column. Each guess is shown in the Keywords column, and the

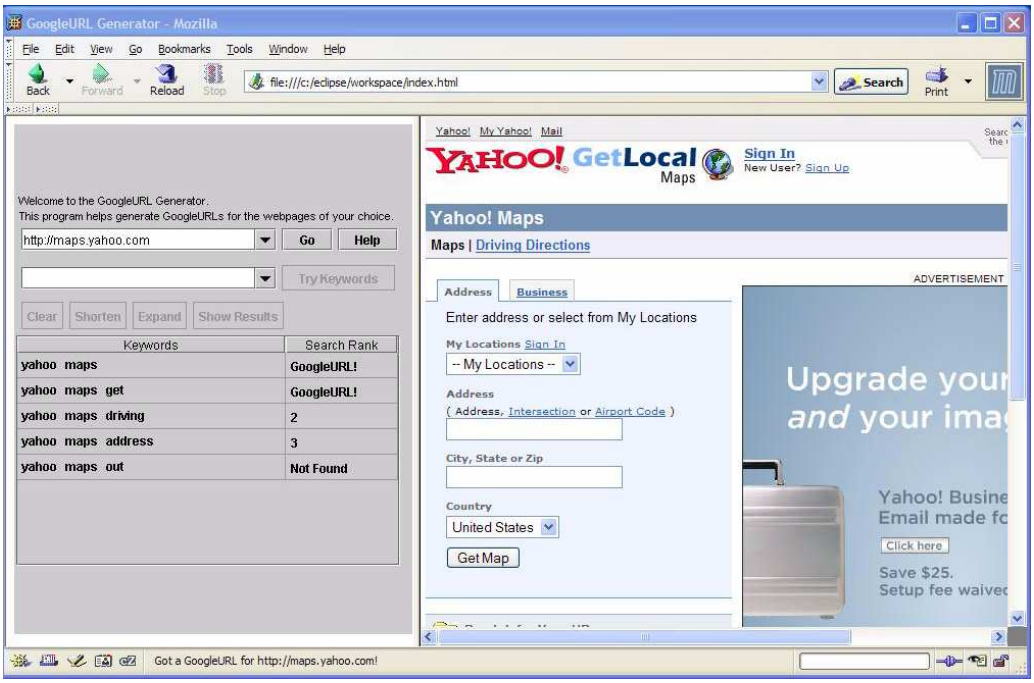

Figure 5-1: GoogleURL Generator Interface.

Keywords	Search Rank
yahoo maps	GoogleURL!
yahoo maps get	GoogleURL!
yahoo maps driving	2
yahoo maps address	3
yahoo maps out	Not Found

Figure 5-2: Guess Table.

search rank for the target page is shown in the Search Rank column. Any guess for which the target page was the first search result is labeled "GoogleURL". Only the top ten results are examined, so a guess is labeled "Not Found" if the target page is not among the top ten Google search results. The entries in the table are automatically sorted by search rank, so that any successful GoogleURLs for the target page are shown at the top. Among guesses that have the same search rank, the guess consisting of the fewest number of keywords is

sorted above the longer guesses.

5.1.2 Keywords Field

There is an editable combo box, called the *Keywords field*. It has a drop-down menu and autocomplete functionality, and is used to construct new guesses to query. The drop-down menu is automatically populated with any words that are found on the target page or backlinks. The user can click on any entry in the drop-down box to fill in that entry into the Keywords field. The combo box works in such a way that the user can construct a multi-word guess by using the combo box drop-down menu several times in succession. The user can also type a guess directly into the Keywords field. As the user types, the drop-down menu will automatically become visible, filled in with any word from the page that matches what the user has typed so far, as shown in Figure 5-3. Unlike the standard autocomplete implementation, where one can only autocomplete the entire contents of the field, the Keywords field autocompletes by each individual keyword of the guess. The advantage of the autocomplete and the drop-down menu is, first of all, as a shortcut so that the user does not have to type in the whole word. Additionally, this functionality is designed to prevent the user from making spelling errors, or from accidently making a guess using words that are not on the target page. It is easy for the user to recognize when he has inadvertently typed a word that is absent from the target page, since the drop down menu showing possible completions will suddenly disappear as he types.

The user can also construct guesses by using the Guess table. Each word in the Guess table that is not already in the Keywords field becomes highlighted when the user moves the mouse over that word. In Figure 5-2, the user has moved the mouse over "maps". When the user clicks on a keyword, the clicked word is appended onto the end of the Keywords field. The rationale for this functionality is that the words in the Guess table are the highest ranked keywords, and the user is likely to reuse some of these if he performs subsequent guesses. Initially, the Guess table is populated with highly ranked keywords that the system hypothesizes would make up the best guesses. Any guess that the user performs manually is placed into the Guess table, and the individual words of that guess will also be clickable. The fact that these keywords are clickable allows the user to easily construct a guess made up of permutations of the keywords in previous guesses. In addition to the individual keywords, the individual rows of the Guess table are clickable. Clicking on a row copies the

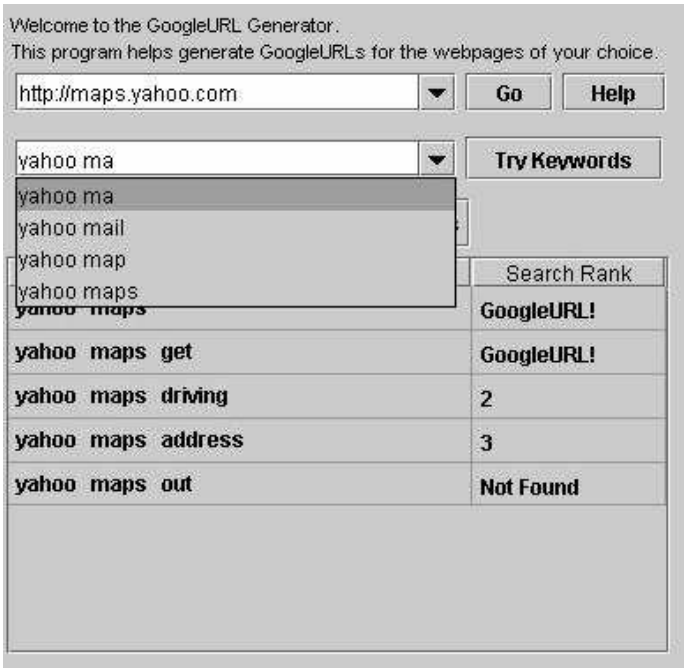

Figure 5-3: Autocompletion as the user types.

entire guess into the Keywords field.

5.1.3 Shorten and Expand

In order to construct additional guesses using a single guess as a foundation, the interface provides mechanisms to *shorten* and to *expand*, using the contents of the Keywords field as the guess to operate on. Call the *base guess* the contents of the Keywords field when the shorten or expand is performed. The shorten operation creates several guesses with fewer keywords than in the base guess. The current implementation preserves the order of the keywords and creates several new guesses, each of which has exactly one keyword removed from the base guess. For example, performing a shorten on the base guess "boston public library" creates the three resulting guesses "boston public", "boston library", and "public library", which are queried and added into the Guess table. Performing an expand also creates several guesses from the base guess, but instead adds additional keywords onto the guess. In the current implementation, guesses are created by taking the highest ranked keywords that are not already contained in the guess, and appending them onto the end of the base guess. As an example, consider a page where the highest ranked keywords, in

descending order, are “calendar”, “academic”, “mit”, “degree”, and “deadline”. If the user performs an expand on the guess “mit academic”, guesses that will be automatically generated are “mit academic calendar”, “mit academic degree”, and “mit academic deadline”.

5.1.4 Show Results

When the user is attempting to manually create GoogleURLs, it is sometimes helpful for him to be able to examine the Google search results. The button labeled “Show Results” causes the Google search results of the query currently in the Keywords field to be displayed in the frame on the right. The user can look at these and potentially gain insight into why the other search results had been ranked higher.

5.1.5 Help

A button labeled “Help”, when clicked, displays a help page in the frame on the right. The page explains what GoogleURLs are and what the GoogleURL Generator is, and how it can be used. There are also detailed instructions and walkthrough for how to use the system, with screenshots. A “Common Problems and Troubleshooting” section details possible issues that users may have with either with using the system or with GoogleURLs in general.

5.1.6 Applet security considerations

By default, security considerations do not allow applets to open a network connection to any host other than the one it came from. Furthermore, applets cannot run any scripts or read or write any files on the host that is executing it. For experimentation purposes, the GoogleURL Generator was installed on a computer with a lenient Java policy. Many of these limitations can be circumvented, as detailed in Section 8.1.2, by running a server on the host machine and passing all outside communication through the server.

5.2 Design Process

The user interface of the GoogleURL Generator has undergone several iterations of revision. The interface was originally developed as a term project for 6.893, an MIT course on user interface design and implementation. The *Spiral Model* of development was applied, in

which each successive stage consists of a period of design followed by an implementation of that design. The implementation is then evaluated, and the feedback is incorporated into a new design in the next stage of development. In the spiral model, early stages of implementation are low fidelity. To avoid having to throw away large amounts of work, just enough features and functionality are implemented to allow for useful feedback to be generated. In later stages, as the design becomes finalized, the implementation phase is the implementation of the whole system.

5.2.1 Paper Prototype

The original GoogleURL Generator interface design was found to be too complicated. In that system, the interface was split into four separate panels as appear in Figure 5-4. The

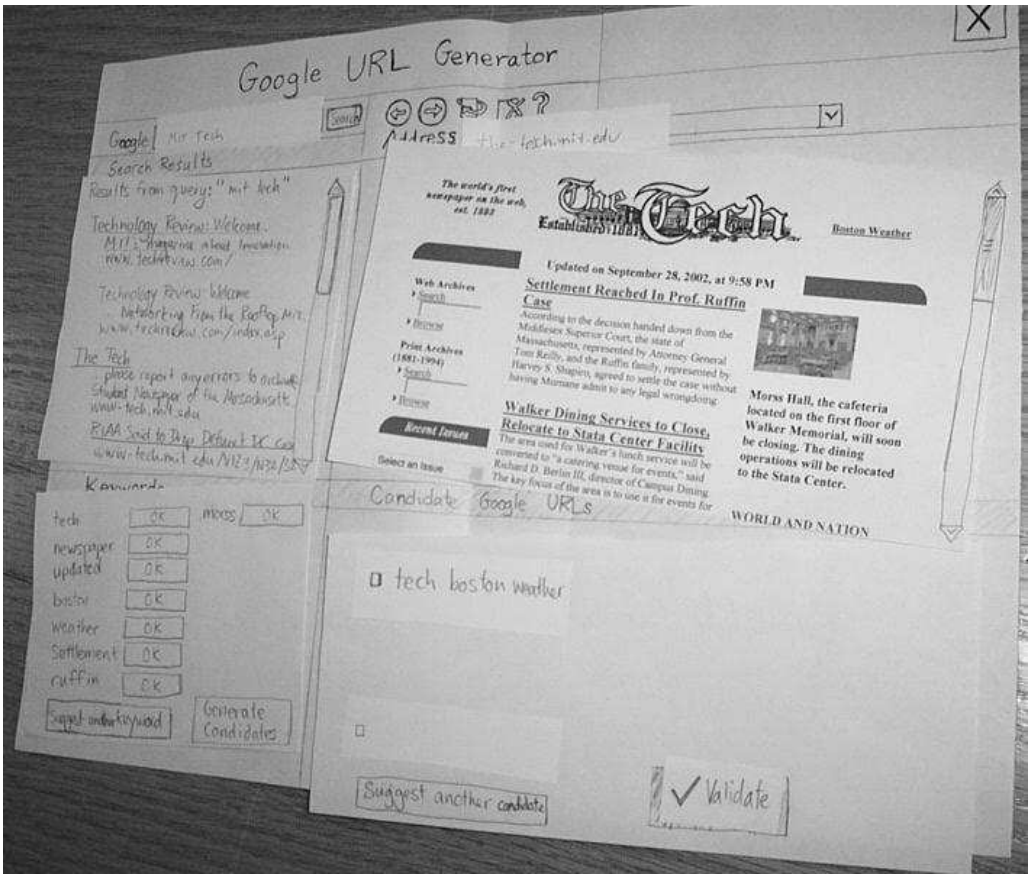

Figure 5-4: Paper Prototype of GoogleURL Generator.

upper-left panel was the *Search results panel*, which could be used to query a phrase in Google and load up any of the results as the target page. Alternatively, the user could

load a target page by entering its URL in the *address bar*, at the top. Once a target page was loaded, it would be displayed in the upper-right panel, the *main panel*. Upon loading, the system would automatically parse the target page and generate two lists: the *suggested keywords* and the *candidate GoogleURLs*. The suggested keywords would be displayed in the *keywords panel*, at the lower-left. The candidate GoogleURLs would be displayed in the *candidate GoogleURL pane*, at the lower-right. Upon loading, selected keywords found on the target page would be among the suggested keywords. Each suggested keyword would have a combo box next to it with which the user could designate whether that keyword was memorable or not. The user could also suggest a keyword of his own, that would be added to the list of suggested keywords. In the keywords panel, a button labeled "Generate Candidates" would create an updated list of GoogleURL candidates based on the the new information about keyword memorability. The user would also be able to explicitly add a guess of his own to the candidate GoogleURLs list. Each candidate GoogleURL would have a checkbox next to it. The user would be able to check a subset of these candidate GoogleURLs and click "Validate", at which point the system would query those candidates in Google and inform the user which were GoogleURLs.

This original design involved a large amount of user interaction to generate any GoogleURLs. The interface had been designed in this way for two reasons. First, it was desired to tap into user knowledge about keyword memorability. Having the user explicitly mark which keywords he considers memorable allows the system to glean more insight into memorability. Without such user input, it is more difficult for a computer to determine which GoogleURLs might be memorable. The other reason for extensive user interaction was to avoid excessive queries to Google, because of the expense of each Google query, explained in Section 4.2.3. By selecting which queries to validate, the user would filter out any guess with which he would not be satisfied, and therefore the system would not waste any queries on GoogleURLs that the user would not want.

To evaluate the interface's effectiveness and usability, this system was implemented as a paper prototype, shown in Figure 5-4, which several 6.893 students sampled as test users. Users were generally intimidated by the user interface, primarily because it had so many components and sub-panels. Having a combo box for every keyword and a checkbox for every candidate GoogleURL was too much for a user unfamiliar with the system to take in all at once. Furthermore, users were confused about needing to manipulate the list of

suggested keywords, since it does not seem immediately related to their goal of generating a GoogleURL.

5.2.2 Computer Prototype

In the next stage, a computer prototype was developed with a modified design. In this prototype, there was no Google search results panel or main panel. Instead, the demo was web based, and a simulated browser was embedded into the webpage, as shown in Figure 5-5. The user typed the target page URL into the address bar, and the embedded web browser would display that target page. In the actual prototype, all of the backend functionality was canned, so the only target webpage that worked was the Tech's website at "http://www-tech.mit.edu". Once the target page was loaded, an ordered list of keywords was displayed in the *Keywords list*. These keywords were ranked based on the system's processing of the target page, and expected usefulness of each keyword. The user was able to manipulate the ranking of those keywords by selecting a keyword and pressing the "Move Selected Up" or "Move Selected Down" buttons. The user could also add keywords to the keyword list by typing them and clicking "Add Keyword". The idea was for the user to rank the keywords in order of how desirable each keyword would be if used in a GoogleURL. After the user clicked "Generate Google URL", the system would attempt to generate GoogleURLs based on the user ranking of the keywords. A list labeled *Validated Google URLs* contained all successful GoogleURLs that had been found so far. Next to it, a list labeled *Guesses tried* showed a record of the search rank of every phrase that the system had queried in Google. The user was also able to query a GoogleURL guess of his own by entering the phrase and pressing "Try a Lucky Guess". The guess would then appear in the Guesses tried list, and, if it were indeed a GoogleURL, also in the Validated Google URLs list.

Again, the computer prototype was evaluated by 6.893 students. One of the most prevalent complaints about the interface was that it was unclear what effect the ordering of keywords had on the GoogleURLs generated by the system. As with the paper prototype, manipulating the keywords list had no obvious relation to generating GoogleURLs. When a user simply intends to find a GoogleURL, then having to rank a list of keywords forces him to deviate from his actual goal. The interface was still too complicated and intimidating, with an abundance of components. The interface also took up such a large portion of screen space that users could not see both the target webpage display and the controls for

generating GoogleURLs.

5.2.3 Standalone Java application

In the next design, the GoogleURL generator was a standalone application, as displayed in Figure 5-6. The focus, in the prototype, was on the interface for actually generating GoogleURLs, instead of expending effort rendering webpages. Therefore, it was decided to separate the target page display from the interface for generating GoogleURLs. The system would simply display the loaded target webpage in the default web browser in another window. The new interface was split up into three parts: the *Browser*, the *Guess area*, and the *Guess table*.

Browser

The Browser, displayed in Figure 5-7 looks like the navigation control of a traditional web browser. There are *Back*, *Forward*, *Reload*, and *Home* buttons, which operate similarly to how one would expect of a web browser. There is a text field for entering the URL of the website to load, called the *URL field*. A *Go* button, when clicked, displays in the default web browser the URL in the URL field, and a *Help* button displays the GoogleURL Generator Help page.

Guess Area

The Guess area, shown in Figure 5-8, contains a field for inputting GoogleURLs to validate, called the *Keywords field*. To guess a GoogleURL, the user enters a GoogleURL into this Keywords field and presses the *Try Keywords* button. The user can shorten or expand the guess in the Keywords field by pressing the *Shorten* or *Expand* button. As with the current system, the Keywords field was equipped with autocompletion.

Guess Table

The Guess table, as shown in Figure 5-9, shows a record for all GoogleURL guesses that have been performed by either the system or the user. The Validated GoogleURLs section and Guesses tried areas, from the previous prototype, were fused together into this single Guess table. The table had three columns: *Keywords*, *Search Rank*, and *Option*. As in the current interface, each GoogleURL guess encompassed one row of the Guess table. The

Keywords column and Search Rank column worked the same as they do in the current interface. The additional Option cell, however, contained buttons corresponding to options the user could perform on the guess in that row. The *View Results* button would display the top ten search results for that guess. The Shorten and Expand buttons would operate on the guess corresponding to that particular row, instead of on the contents of the Keywords field.

Many similarities exist between this application and the current interface. The current system was in fact developed from feedback from evaluation of this application. Several significant, albeit premediated, flaws in the interface arose from the fact that the application and the rendering of the target page were in two windows in separate applications. Consequently, the user could not easily see both the interface and the target page at the same time. Furthermore, the system had no way to detect if the user had clicked on any links in the web browser or had otherwise navigated to a new page. The evaluation also made it clear that the interface was still too cramped and complicated. Users were particularly not inclined to try out the buttons in the Option column. To resolve these issues, it was decided to eliminate the Option column and to make the interface thinner, so that it would fit in a sidebar aside the displayed target page. In this way, the interface would be simplified and easier to use, and the user would be able to view both the target page and the GoogleURL Generator all at once. The current system was therefore made into an applet embedded in a side frame.

The overriding theme throughout the whole design process was that the simpler design was always preferable. Because of all the user interaction solicited, the initial design could have allowed for a more powerful engine for generating good, memorable GoogleURLs, in a manner that avoided excessive Google queries. These benefits, however, are useless if the user cannot figure out how to use the interface, or is too intimidated to use the system in the first place.

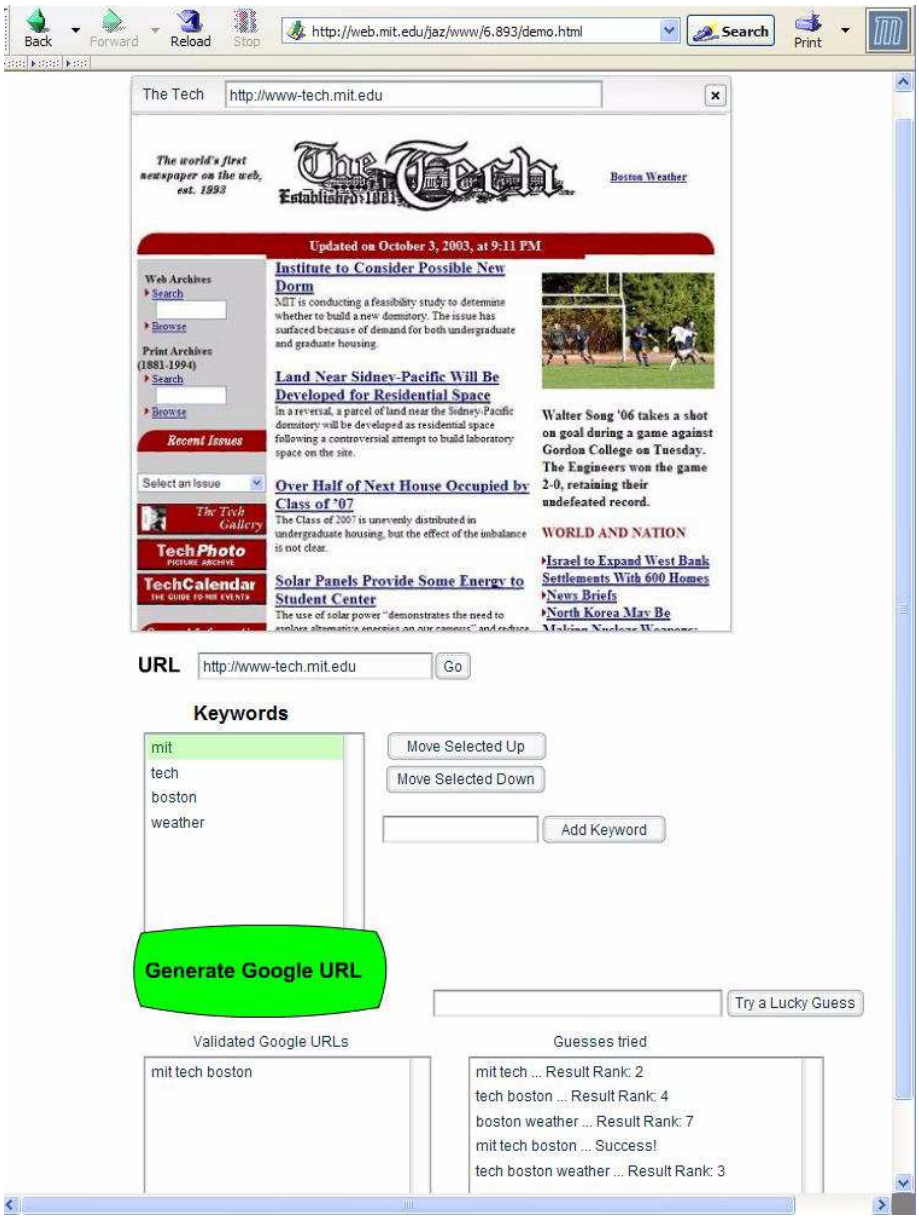

Figure 5-5: Demo of GoogleURL Generator Prototype.

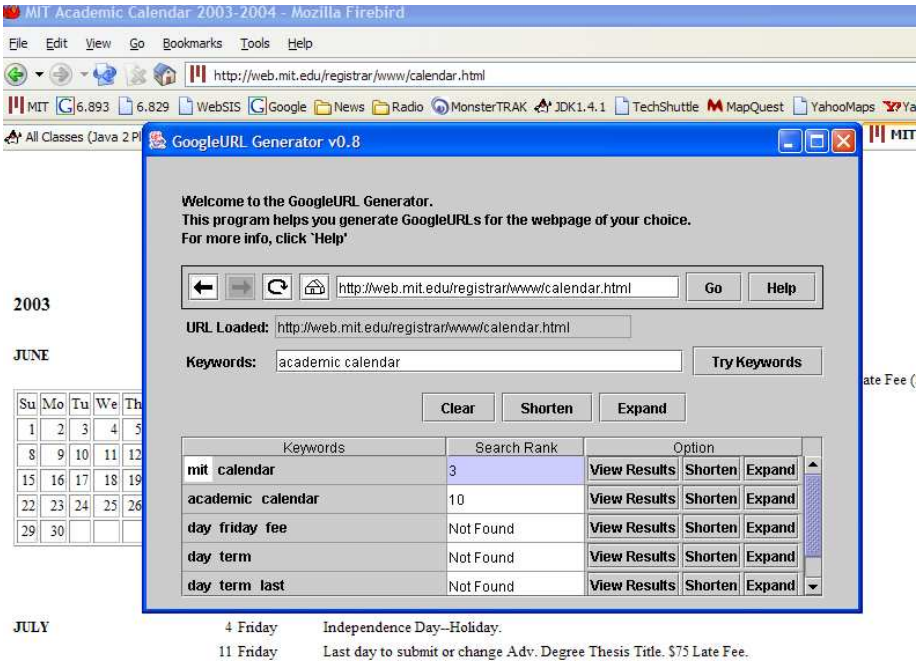

Figure 5-6: GoogleURL Generator standalone application.

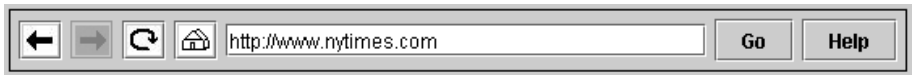

Figure 5-7: GoogleURL Generator standalone application Browser.

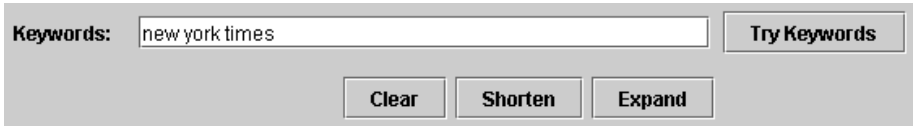

Figure 5-8: GoogleURL Generator standalone application Guess Area.

Figure 5-9: GoogleURL Generator standalone application Guess Table.

Chapter 6

GoogleURL Generator Evaluation

Evaluation of the GoogleURL Generator took place in iterations throughout the development, as described in Section 5.2. This chapter describes the evaluation of the current version of the GoogleURL Generator, which was tested in a pilot user study and through a programmatic evaluation.

6.1 Pilot User Study

The pilot user study was performed to achieve two separate goals. First of all, it was desired to evaluate the usabililty of the GoogleURL Generator system. The other intention was to explore the feasibility of using GoogleURLs as bookmarks.

Actual users in the study were five researchers at MIT Computer Science and Artificial Intelligence Lab. The users fit the desired profile of being competent computer users, and were able to quickly grasp the concept of GoogleURLs.

At the beginning of the study, each user was given a briefing introducing him to GoogleURLs. It was next explained to him for what reasons GoogleURLs might be useful. Then, he was told that the GoogleURL Generator was a tool for generating GoogleURLs for webpages of the user's choosing, and that in the study he would be trying out the GoogleURL Generator in action. He would be given a list of URLs and asked to generate, for each page, the single GoogleURL that would be:

1. most useful for him to find the website again, and

2. easiest for him to communicate to others so that they could find the website.

amazon	yahoo mail	calendar academic mit	boston public library
amazon	yahoo mail	mit academic	boston library
amazon bookstore	yahoo mailbox	calendar mit	public library boston
amazon	yahoo mail	mit calendar	bpl
amazon	yahoo mail	mit calendar	bpl

Table 6.1: GoogleURLs that users generated for http://www.amazon.com, http://mail.yahoo.com, http://web.mit.edu/registrar/www/calendar.html, and http://www.bpl.org

Any final questions about the background were taken. The users were then given the list of webpages and placed in front of the GoogleURL Generator.

The list of webpages began with pages for which the GoogleURL Generator generated good GoogleURLs. Next came pages for which the GoogleURLs generated were expected to be unsatisfactory. After these were descriptions of webpages for the user to find and generate a GoogleURL for, such as "your homepage", "the news or magazine site you read most often", and "your high school's home page". Each user was observed as he used the system, and the GoogleURL Generator system recorded a log of all user and system activity. The GoogleURLs that users generated for the predefined pages are shown in Table 6.1.

It generally took a small amount of time before the users became accustomed to the GoogleURL Generator interface. The critical event occurred when the user realized that he had to enter the URL of the target page into the URL field combo box. The problem is that the applet gives little feedback after the user has entered the target page URL into the URL field. Since the applet is single threaded, the guess table display is not updated as Google queries are performed, but only once all rows have been put into the guess table. Therefore, it takes some time for all GoogleURL generation to take place. During this time, system status was reported in the web browser status bar, but this is small and easy to overlook when users do not know to be watching for it. Consequently, users often did not realize that they were on the right track and were using the GoogleURL Generator correctly. After realizing that entering the target page address into the URL field was the first step, however, users tended to quickly figure out how to use the GoogleURL Generator.

An interesting finding was that the majority of users were unaware that the words in Google queries are not commutative. For example, the GoogleURL Generator generated the GoogleURL "public library boston" and a user wrote down "boston public library" without

verifying that it was, in fact, a GoogleURL. When questioned about it later, users were surprised that reordering the words in the search query would change the results. After understanding that Google tries to return pages containing phrases in the search query, it becomes clear that rearranging the search terms affects the search results. After all, one would expect an entirely different sort of results when searching for "dogs eating" than when searching for "eating dogs". To use GoogleURLs effectively as bookmarks, it is essential that users understand that GoogleURLs cannot just be arbitrarily reordered. Once learned, this is a simple concept to understand, but poses a minor barrier to using GoogleURLs as bookmarks.

Interestingly, users differed in the priorities they had when choosing good GoogleURLs. Before the study, I had expected that users would choose the shortest GoogleURL. Though this was often the true, it turned out to not always be the case. Sometimes users opted for predicted robustness over shortness, and at other times chose memorability over shortness. Even at times when a one-word GoogleURL had been generated, users did not always pick it. In one instance, the target page was "www.commondreams.org". The GoogleURL Generator had determined that "common", "dreams", and "common dreams" were all GoogleURLs. The user chose the two-word GoogleURL, "common dreams", as the best GoogleURL. When later asked why he preferred the longer GoogleURL, he responded that he would worry that either "common" or "dreams" might stop working as a GoogleURL. Robustness was clearly a top priority. In some other cases, users favored memorability over shortness. When determining a GoogleURL for "www.bpl.org", it was found that "boston library" and "boston public library" were both GoogleURLs. One user chose "boston library" and another chose "boston public library". The most satisfactory GoogleURL depended on the person and their particular priorities, be it ease of memorability or ease of typing.

During the pilot study, users were asked if there were any webpages for which they currently used GoogleURLs. Most users, it turns out, do not regularly use GoogleURLs for finding webpages.

Three users were contacted again a week after the study and asked to find the webpages that they had chosen themselves. They were given the same cues as they had been given during the pilot study, such as "your high school's webpage". The intention was to observe how they would find those pages, to draw insight into the memorability of GoogleURLs. All three users were able to find the pages again on the first try, although this result might

hello world
michael jordan
jobs
britney spears
brian stube
pants
strawberry fields forever
chicken teriyaki
dogs eating bananas
peak oil chicken
songs without love
yellow carpet spiral
international
international treaty

Table 6.2: Seed phrases used to generate sample set of 140 webpages

have been an artifact of users being advanced computer users from MIT.

6.2 Programmatic Evaluation

The system also underwent a programmatic evaluation, geared toward assessing the effectiveness of the algorithm and adjusting the scaling factors.

In the evaluation, it was desirable to only test the GoogleURL Generator on pages that are in the Google database. It is possible to use a random string as a *seed* that yields webpages that are guaranteed to be in Google's index. The testing algorithm does so by querying the seed in Google and using each of the top ten results, in turn, as the target page for which to generate GoogleURLs. To ensure that an approximately stable sample set of webpages, the algorithm can fix a set of seed phrases. The fraction of the pages for which the GoogleURL Generator was able to create a GoogleURL is calculated.

With this testing framework in place, it is possible to obtain an objective measure of the performance of the GoogleURL Generator. The scaling factors can be adjusted up and down, to see whether they have a positive or negative effect on the fraction of pages for which GoogleURLs generated are generated.

The evaluation used 14 fixed seed phrases, listed in table 6.2, each of which yielded 10 webpages. A total of 140 webpages were examined on each test iteration. The best overall yield that could be achieved over the sample set, using the top 15 guesses, was generating a

GoogleURL for 83% percent of the webpages. Using only the top five guesses, GoogleURLs were generated for 72% percent of the sample. The relationship between yield and number of guesses tried is shown in Figure 6.2.

Figure 6-1: Effect of number of guesses on percentage of GoogleURLs calculated.

First of all, it was desirable to know that the scaling factor for each context was of approximately the correct magnitude. To verify this, each scaling factor was first set to an initial value v_0, and the GoogleURL Generator was run on the sample set of 140 pages. In turn, each scaling factor was set to zero, and an iteration of examining the 140 pages was performed. The overall percentage of pages for which GoogleURLs were generated was compared when the scaling factor was zero and when the scaling factor was set to its initial value v_0. In this fashion, it is possible to determine not only whether each scaling factor indeed contributes to the yield, but also to learn how much each individual scaling factor contributes.

The results from these tests are valuable, but misleading about how useful each scaling factor is. The results depend on which initial values are used for each scaling factor. As it is increased from zero, each scaling factor can generally be expected to contribute an

increasing amount to the yield. At some value, the scaling factor would reach a local maximum, and then the contribution to the yield would begin decreasing as the scaling factor was increased. Eventually, the scaling factor's contribution would be negative. This relationship is depicted in Figure 6.2. If the initial value assigned to any scaling factor had

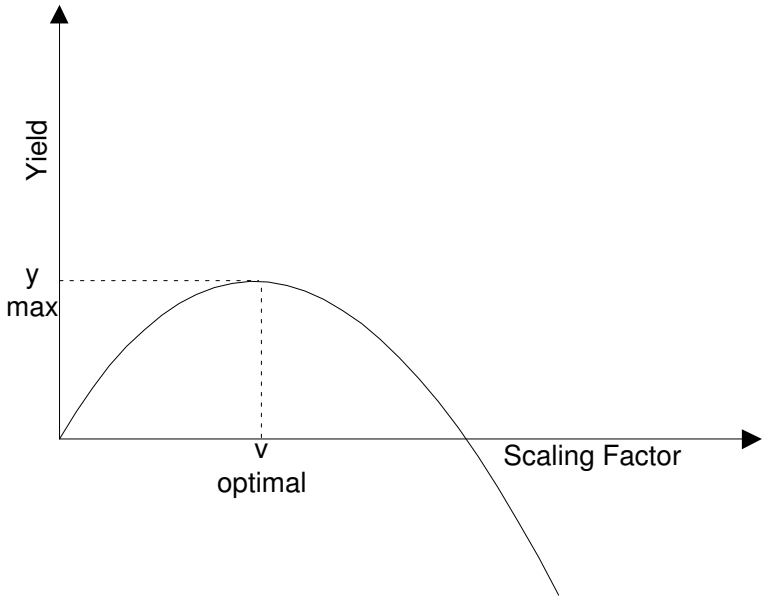

Figure 6-2: Expected relationship between individual Scaling Factor and Yield.

been too high or too low, a smaller change in yield would be achieved. The scaling factors for which an appropriate initial value had been chosen would seem to have a larger effect on the yield.

Therefore, another series of tests were performed to find the local maximum $v_{optimal}$ for each scaling factor. Each scaling factor was first assigned an initial value v_0. The GoogleURL Generator was run with that scaling factor as $v_0 + \delta$ and as $v_0 - \delta$, for some small value of δ. It was observed in which direction moving v would yield the higher fraction. The test was then reran with the scaling factor moved by δ in that direction, either to $v_0+2\delta$ or $v_0 - 2\delta$. The scaling factor would continue progressing in the same direction, changing by δ each time. Once a local maximum was found, that value would be kept as the refined scaling factor. The optimal scaling factors are as shown in Table 6.3.

After an optimal assignment of scaling factors had been determined, it was possible to determine each scaling factor's contribution to the overall yield by setting a single factor to zero and comparing that yield to the optimal yield. Only the top five guesses were

neighbor scaling factor	1.5
title scaling factor	3.6
URL token scaling factor	4.1
outgoing link scaling factor	-0.9
backlink scaling factor	2.1
bold scaling factor	1.8
cap scaling factor	0.5
heading scaling factor	2.5

Table 6.3: Optimal Value of Scaling Factor

queried, because five guesses is a feasible number of guesses for the GoogleURL Generator to automatically query. Furthermore, a sharper distinction is seen between different factor contributions when fewer guesses are used.

All scaling factors were found to contribute to the overall yield in varying degrees, as illustrated in Figure 6.2. By far, the single most important scaling factor was the *neighbor*

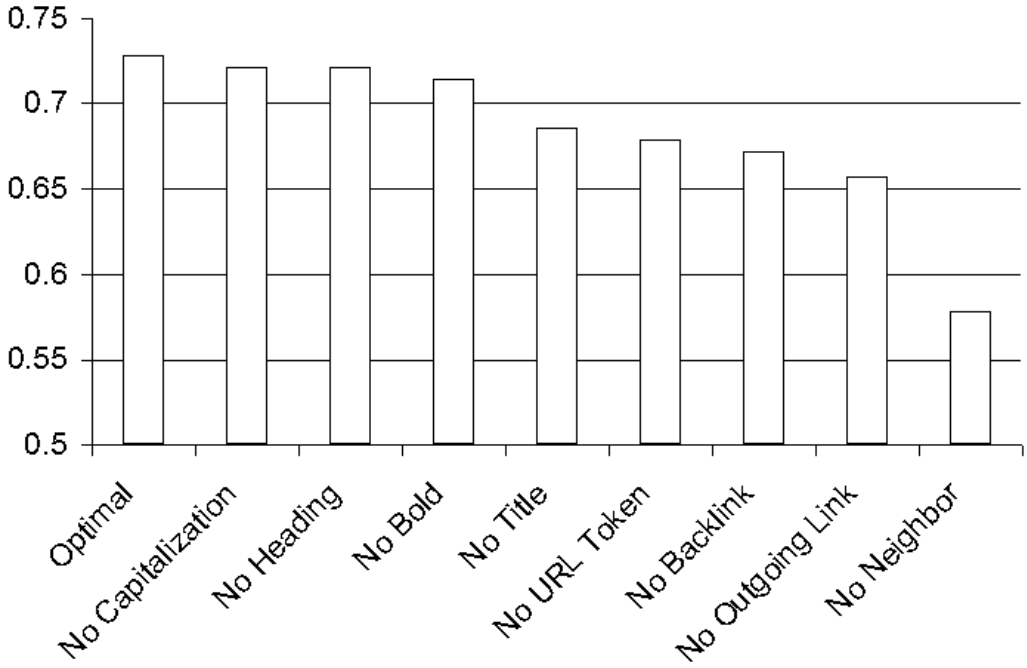

Figure 6-3: Contribution to yield from each individual scaling factor, using five guesses per webpage.

scaling factor. Taking phrasing into account, and using the top five guesses, the GoogleURL Generator generated a GoogleURL for 73% of the sample set, while setting the neighbor

factor to zero generated GoogleURLs for 58%. Another important factor was the *outgoing link scaling factor*, which was negatively weighted to counter the word's occurrence as plaintext. Removing the negative weighting for words in links, the yield dropped from 73% to 65%. Other factors that had significant contributions to the yield were the *backlink scaling factor*, *URL token scaling factor*, and *title scaling factor*. Scaling factors for capitalization, boldface, and headings contributed less.

A serious obstacle to carrying out this evaluation was the limit of 1000 queries per day. For each webpage, one Google query is performed to retrieve the list of backlinks. The algorithm generates and validates 5 GoogleURL guesses, making 6 Google queries per page. Obtaining the ten webpages from each seed phrase requires one Google query per seed. Using 14 seed phrases yielded 140 webpages to process, so the grand total of Google queries required per test was 854. The programmatic evaluation, therefore, had to be laboriously carried out over several weeks using multiple keys.

Chapter 7

Google Reordering

Google Reordering seeks to reorder the results returned by a Google search, and place at the top the hits that are most likely to be the user's desired results. Based on the user's browsing history, it is possible to infer that some results would be more likely to be desired results than others. The principle applies that, if a person has been to a page multiple times, it is likely that he will wish to return to that page. In particular, if a site he has visited many times is among the search results, he is more likely to return to the site he frequents than to click on one of the sites he has never been to. Consider the results of a web search in which there is some page, X, that the user has accessed frequently. Also among the results is a page, Y, that is ranked above X, but which the user has never accessed. It is more likely that the user will want to go to page X than to page Y, so it makes sense to place the result for X above the result for Y.

Google Reordering enhances the potential for using GoogleURLs as bookmarks. The reordering can have the advantageous effect of making GoogleURLs more robust. Even when a GoogleURL has become outdated, it can still work as a GoogleURL if the user has visited that target page frequently. Furthermore, Google Reordering allows for shorter GoogleURLs. As an example, consider a user Bob who wants a bookmark for the BBC News website. Bob knows that "bbc news" is a GoogleURL for the BBC News website, but that "news" yields it as the second hit, behind CNN.com. If Bob never visits CNN.com, but does frequently access the BBC News website, then Google Reordering will cause "news" to become a GoogleURL for BBC News, from Bob's computer, as shown in Figure 7-1.

A drawback of Google Reordering is lack of portability, which is a shortcoming shared

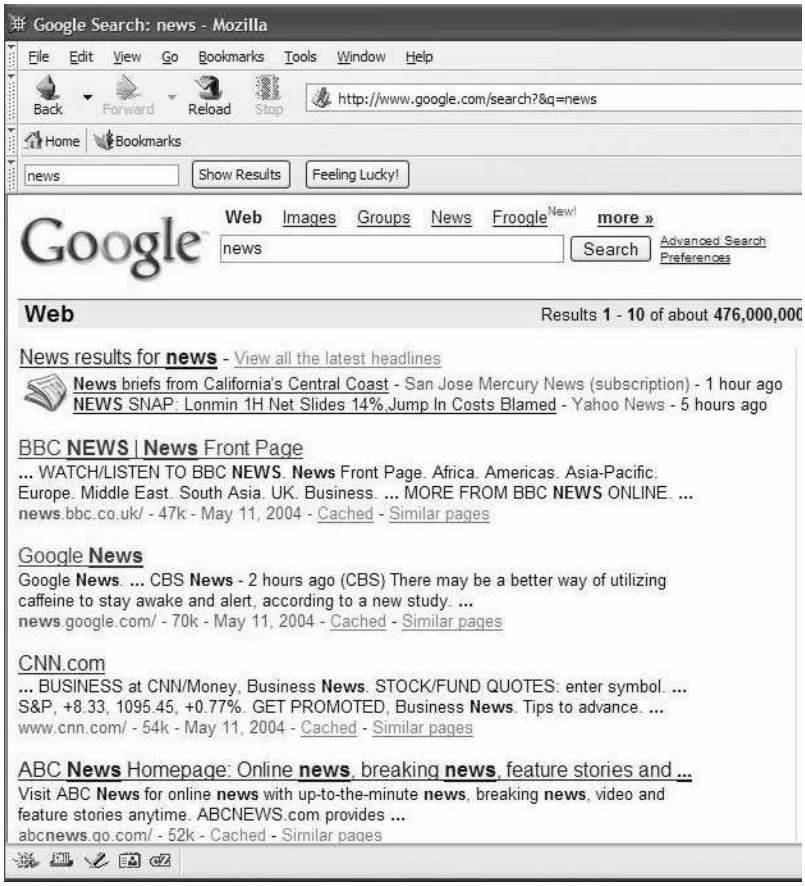

Figure 7-1: Google Search results for "news" for Bob.

with the traditional means of bookmarking URLs in a web browser. The engine performing the reordering needs access to the user's browsing history, or at least to some information in the profile. Bob from the preceding example would not be able to walk up to any computer and use "news" as a GoogleURL for BBC News.

Another potential drawback of Google Reordering is that it might render existing GoogleURLs nonfunctional. Google Reordering destroys the platform independence of GoogleURLs by placing dependencies on the user profile, so what is a GoogleURL for one user might not be for everyone else. Consider another user, Alice, who uses "news" as a GoogleURL for CNN.com, the top hit without any reordering as shown in Figure 7. If Alice uses Bob's computer, then the query "news" will bring her to the BBC News website instead of CNN.com as she had intended. Furthmore, if Alice uses Bob's computer and visits CNN.com frequently, then Bob's GoogleURL of "news" for the BBC News may be disrupted.

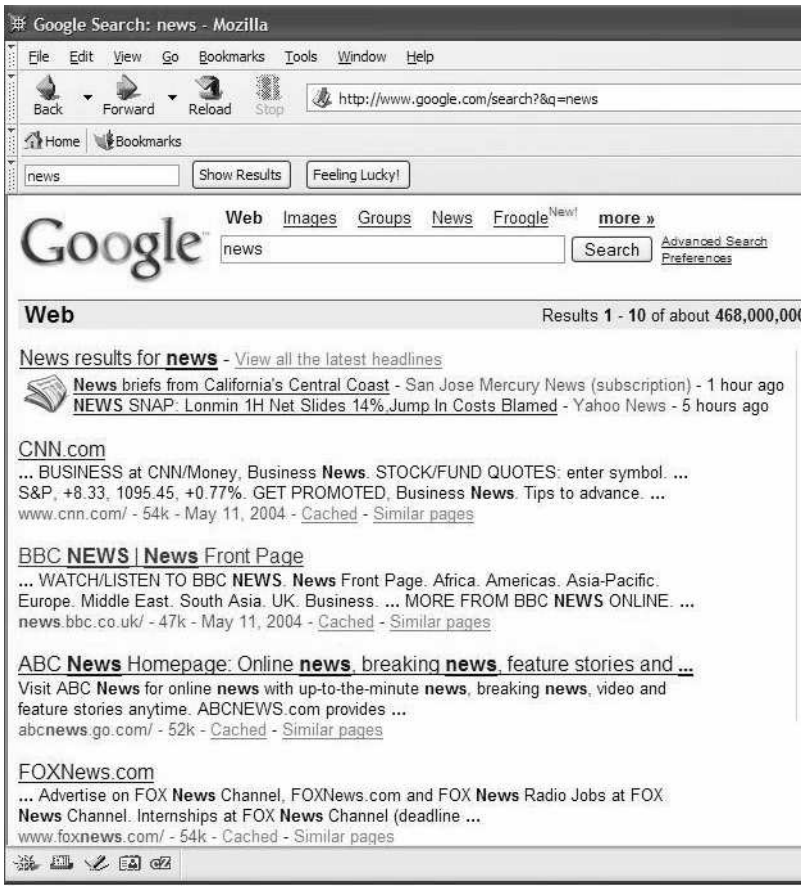

Figure 7-2: Google search results for "news" for Alice.

Privacy issues come into play when considering how powerful to make Google Reordering. All web browsers keep a browsing history, so using this history for Google Reordering is, in principle, no more intrusive than the act of browsing. If the user wishes to remove all traces of all sites he has visited, he can simply clear his browsing history. All Google Reordering state will be automatically cleared. On the other hand, a significant drawback of Google Reordering is that browsing history records of pages that have not been accessed for a certain amount of time will be automatically cleared. If the Google Reordering system maintained browsing records, without user consent, after they had been cleared from the browsing history, that would violate user privacy. Google Reordering is therefore stuck using records that fade out over time.

7.1 Algorithm

The algorithm for reordering Google results is based on the idea that users tend to revisit the same types of pages. In the current system, a very simplified version of the algorithm is implemented. In this section, however, the proposed algorithm is presented in its entirety.

7.1.1 Result Weight

When a Google search is performed, a *result weight* is calculated for each search result. The results are then placed in order of the result weight, and presented to the user.

The algorithm gives weight to a search result based on the frequency of access, recency of access, its search rank in the original Google search results, and frequency of access of other pages in that domain. The result weight is the sum of four factors: the *rank factor*, the *recency factor*, the *frequency factor*, and the *domain factor*. The result weight w is calculated as $w = f_{rank} + f_{recency} + f_{frequency} + f_{domain}$, where f_{rank}, $f_{recency}$, $f_{frequency}$, and f_{domain} are the respective factors.

Rank Factor

The rank factor f_{rank} is a function that increases with search rank. A possible implementation could be a linearly increasing rank factor. In that case, $f_{rank} = c_{rank}(rank_{MAX} - rank)$, where c_{rank} is a constant, $rank_{MAX}$ is the number of search results, and $rank$ is the search rank among the Google results.

Recency Factor

The recency factor $f_{recency}$ measures how recently the search result has been accessed. A simple implementation could be $f_{recency} = \frac{c_{recency}}{\delta_{access}}$, where $c_{recency}$ is a constant and δ_{access} is the time since the page was last accessed.

Frequency Factor

The frequency factor $f_{frequency}$ varies with how many times the page has been visited in the known history. An straight-forward implementation would be $f_{frequency} = c_{frequency} * frequency$, where $c_{frequency}$ is constant and $frequency$ is the number of accesses to the page.

Domain Factor

The domain factor f_{domain} is determined by the number of accesses in recent history to pages in the same domain. It would be cumbersome to explore the entire browsing history in real time upon any web search. To avoid this, the system need only explore the entire history once when the web browser is loaded. Upon loading, the system can parse the browsing history and determine the domain for each entry, keeping a record of the number of accesses to each domain. Multiple entries in the same domain will be counted toward the same domain, so
"http://news.bbc.co.uk/2/hi/middle_east/3690669.stm" and
"http://news.bbc.co.uk/2/hi/americas/3689267.stm"
would both be counted as accesses to the domain "news.bbc.co.uk". To calculate the domain factor for a search result, the algorithm would first parse the URL to determine the domain of the search result, and look up the number of times that domain had been accessed. An implementation could calculate the domain factor as a constant multiplied by the number of accesses to that domain.

7.1.2 Result Ordering

Finally, the search results would be placed in order of the calculated result weight. Note that, in the absence of any history information, the only nonzero factor will be the rank factor, and the results will not be changed from the original Google search results.

It is important that the system also recognize when the "I'm Feeling Lucky" button is pressed, and immediately direct the browser to the highest weighted result.

7.2 Implementation

The current Google Reordering system is implemented as a plugin for Mozilla, an open-source web browser. Upon installation, the plugin is granted privileged access to the user's system, and therefore has access to user data such as browsing history.

Search Bar

The plugin creates a bar at the top of the web browser containing a text box and two buttons, as shown in Figure 7.2. The text box is called the *Search box*, and there is a *Show Results*

Figure 7-3: Google Reordering Search Toolbar.

button and a *Feeling Lucky* button. When the Show Results button is pressed, it displays the Google search results to the user, reordered in a very simple manner. Only the top ten results are examined. Among these, any search results in the browsing history are brought above the unvisited results. If multiple results have been visited, then the order from the original Google search results is preserved. When the Feeling Lucky button is pressed, all steps proceed like with the Show Results button, except that after the reordering takes place the web browser is immediately redirected to the top result.

The reordering is implemented by first performing a Google search for the contents of the Search box. The resulting page containing the search results, is then displayed in the web browser. Once the page has finished loading, the search results are reordered by examining and manipulating the DOM tree (Document Object Model tree). To display a document, Mozilla converts the HTML into a DOM tree. The DOM is an API for well-formed HTML and XML documents, and has a tree structure based closely on the structure of the HTML document modeled. For example, Figure 7-4 is a graphical representation of the DOM tree and the corresponding HTML source document. There is an HTML document containing a table, and a node in the DOM tree corresponding to the table body (<tbody>) HTML element. The children of this node in the DOM tree are nodes representing the table rows in the HTML document (<tr> elements).

Using RDF data sources, it is possible to query whether each search result has been visited before. Walking through each search result, the system directly manipulates the DOM tree that is displayed to the user.

The Google Reordering search text box and buttons are implemented as XUL (XML User-interface Language) components, whose event handlers are written in JavaScript.

7.3 Design and Implementation Issues

The current Mozilla plugin performs reordering by working directly off the DOM tree. Before any reordering can take place, the page must be fully loaded and the DOM completely

```
<html>
<head>
    <title>Hello World Page</title>
</head>
<body>
    <table>
    <tbody>
        <tr><td>Hello World</td></tr>
        <tr><td>Ciao Mondo</td></tr>
        <tr><td>Hallo Welt</td></tr>
    </tbody>
</table>
</body>
</html>
```

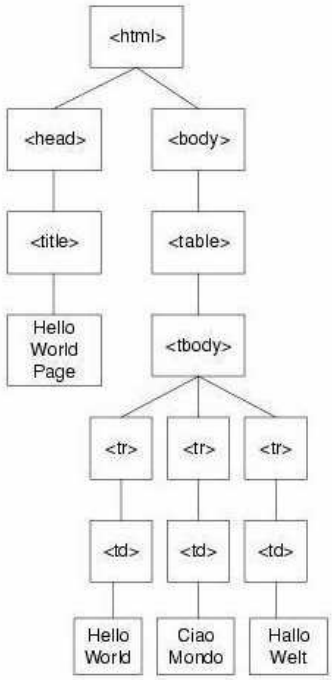

Figure 7-4: HTML document and corresponding graphical DOM tree representation

constructed. As the page is being loaded, however, it is incrementally displayed to the user. As a result, waiting until the search results page is fully loaded to reorder causes the user to see the reordering taking place. It would be more desirable to have the search results page seamlessly reordered behind the scenes before being displayed. It would be possible to hide the page until it is fully loaded, but this has the adverse effect that the user has to wait longer to see any of the page.

It might be feasible for the Mozilla plugin to use Google's Web APIs service, but the 1000 queries per day limit would limit Google Reordering's usefulness.

Since the XUL buttons communicate with the web browser via JavaScript, it is necessary for the user to have JavaScript enabled for the plugin to work. This creates a minor incompatibility with the current GoogleURL Generator, which functions better with JavaScript disabled. There are certain web pages that do not allow themselves to be displayed in the side frame of the GoogleURL Generator. Those pages, "www.nytimes.com" for example, use JavaScript to redirect the browser to their site. It is possible to prevent this usurping of control by disabling JavaScript, but unfortunately that would also disable Google Reordering.

An initial attempt at Google Reordering had been to textually parse the history database on each search. The one advantage of this approach is that it is possible to parse multiple web browser history files. If a user frequently uses multiple web browsers, then it would be desirable to base the reordering on all his browsing history. The big disadvantage is that web browser history files' formats are not all documented, and so parsing them involves reverse-engineering the file format. As a consequence, whenever the file changes, it must be completely reparsed in order to learn what changed. History files can become quite large, so parsing them linearly can take a long time. An approach that needs to do so repeatedly, such as on every search, is infeasible. The plugin approach is efficient because it has fast, easy access to browsing history.

The problem of determining webpage equivalence, discussed in Section 4.2.1, arises also in Google Reordering. Ideally, the system would be able to recognize not only if the the user had visited a certain URL but also if he had visited any synonym of that URL. The web browser history does not recognize synonyms. For example, the two URLs "dictionary.reference.com" and "www.dictionary.com" are synonyms, but are both indexed in Google as "www.dictionary.com". If a user frequently accesses "dictionary.reference.com",

this information should ideally be reflected in the frequency factor, recency factor, and domain factor. When performing a Google search for "dictionary", however, the top hit is "www.dictionary.com". We would like to associate the frequent accesses to "dictionary.reference.com" with "www.dictionary.com", but it is difficult to determine that those two URLs are synonymns. Consequently, a lower ranked search result that the user has visited just once, such as "www.merriam-webster.com", might be reordered to the highest rank.

Chapter 8

Conclusion

In this dissertation, a framework has been presented to supplement present mechanisms for finding webpages and communicating their locations. This process is based on users remembering GoogleURLs in place of URLs. Compared with URLs, GoogleURLs have advantages and disadvantages, and can certainly be useful as portable and easily communicable bookmarks. The GoogleURL Generator was introduced, which is a tool for automatically generating GoogleURLs. Google Reordering can additionally serve to make GoogleURLs more stable and more feasible to use as bookmarks.

8.1 Future Work

The current implementations of the GoogleURL Generator and Google Reordering are not the final step toward using Google as a bookmarking tool. The rest of this chapter discusses possible future work that could improve these tools.

8.1.1 GoogleURL Generator Algorithm

The GoogleURL Generator algorithm leaves out several pieces that could have significant impacts on effectiveness. Incorporating *inverse document frequency* would allow the GoogleURL Generator to distinguish between common and uncommon words. To improve robustness of generated GoogleURLs, the GoogleURL Generator could better detect dynamic content, and avoid using that in GoogleURLs. Finally, beyond the initial guess generation, the mechanisms for automatically generating GoogleURLs are not intelligent. The following sections explain these possible improvements in more detail.

Inverse Document Frequency

A remaining problem with the guess generation algorithm is that a keyword weight is not dependent on the number of occurrences of that keyword across the Internet. Call the number of occurrences of a word on the Internet the *inverse document frequency*. The algorithm does filter out a stop list of extremely common words, such as "the" and "a", but there is no distinction made between a word that occurs one thousand times on the web and another word that occurs several million times.

Inverse document frequency could be valuable information in ranking keywords and generating good GoogleURLs. In general, the infrequent words are expected to be better identifiers for the page. Intuitively, the reason is that uncommon words distinguish the target webpage better than common words. One would also expect that the infrequent words would be more distinctive and thus more memorable. Incorporating inverse document frequency information would also allow the algorithm to reverse-engineer Google's algorithm more effectively.

One way by which inverse document frequency could be determined would be by querying each word in Google and observing how many results there were. The results would be the approximate number of documents in Google's index containing the word. This would be a very effective means of approximate inverse document frequency, but would be prohibitively expensive in terms of Google queries.

One argument against using inverse document frequency is that misspellings of words, which would presumably have low inverse document frequency, would suddenly become highly ranked. This effect would be difficult to mitigate. One might consider constructing GoogleURLs using only keywords that are in the dictionary, but that would seriously limit the GoogleURLs we could come up with. Many surnames, for example, occur in good GoogleURLs but are not in the dictionary. Alternatively, an algorithm could only incorporate inverse document frequency for words that were in the dictionary, but then the distinctive, identifying words that had low inverse document frequency but were not in the dictionary would not get boosted.

Dynamic Content Detection

Robustness is an important property for a good GoogleURL to have. It is essential that every word in the GoogleURL continues to appear on the website. The GoogleURL generation algorithm could be improved to attempt to determine which content on the webpage is static and which is dynamic. As described in Section 3.3.3, it is difficult to determine exactly which words are static and which are dynamic. It is still possible to make good guesses about what content will be safe to use in a GoogleURL.

Dynamic content can be discovered by examining snapshots of the target webpage taken at multiple periods of time. When a user a user loads a webpage, the system can compare the current page with the copy in the Google cache. Any words not present on both pages is dynamic content, and should definitely not be used in any GoogleURL. There is still no guarantee that the other content is static, but at least the content eliminated is certainly dynamic.

This approach will be more productive the longer ago Google has indexed the page. If the page had just been indexed in Google, then the number of dynamic words removed might not be so substantial. If there was content that differed on every access, advertisements for example, at least that would be factored out. An additional possibility would be to examine the Internet Archive [4], and compare snapshots of the target webpage at various points of time.

Intelligent Expand and Shorten

The current mechanisms for shortening and expanding guesses are not particularly powerful. Ideally, the system would record user actions all along, and incorporate this knowledge when creating guesses. For example, if the user typed in several guesses all containing the same keyword, then the system would infer that the user would want a GoogleURL using that keyword.

Another front in which shorten and expand are lacking is that they rely heavily on the originally calculated keyword weights. If those weights are not representative of the page, or if there is a very highly ranked keyword that should not be highly ranked, then the system will have a difficult time creating GoogleURLs. The GoogleURL Generator will persistently try different configurations containing that faulty keyword. One way to

enable the GoogleURL Generator to branch away from faulty keywords would be to dynamically refactor the keyword weights based on observed search results. For example, if no GoogleURLs had been generated and one word was common to all those failed queries, then that word could be downgraded.

It might also be beneficial to have shorten or expand be automatically queried under certain conditions. Suppose, for example, that upon loading a target page the GoogleURL Generator generated and validated several guesses, all of which contained three or more words. If a GoogleURL was found, a shorten could be performed on it, since no guesses of two words had even been attempted. Similarly, upon load, if no GoogleURLs were created and all guesses contained two or fewer words, then an expand could be automatically triggered.

These possible changes could improve the GoogleURL Generator's effectiveness, but the additional Google queries would be more expensive.

8.1.2 GoogleURL Generator Interface

The current version of the GoogleURL Generator can certainly be improved. A single-threaded applet that is not web-accessible is not the most productive means with which users could access the GoogleURL Generator.

Making the GoogleURL Generator multithreaded would be a clear improvement. The user would then see the rows in the guess table be filled in one at a time as each guess is performed. It would be more comfortable for the user, because he would see what is the system is doing, instead of just having to wait without any feedback.

It would be desirable to make the current GoogleURL Generator applet web-based. There are several reasons why the applet is not web-based. For one, as described in Section 5.1.6, applets do not have permission to access the user's system or to communicate with URLs other than the serving website. To grant permission, each user would have to edit his Java security policy. Another reason is that the tools the GoogleURL Generator makes use of are rather large. It would be cumbersome to load all of LAPIS and Google Web APIs as libraries. Furthermore, LAPIS does not currently support being initialized over HTTP.

One solution by which it would be possible to access the GoogleURL Generator over the web would be to set up a server on the machine hosting the applet. The applet would be allowed unrestricted communication with the server. The server would handle all webpage

processing and Google querying. The server would thus remove the inconvenience of users needing to explicitly grant permission. The applet also would not need to load LAPIS and Google Web APIs jar files; all that would be taken care of by the server. The applet and server would not require heavy communication. Whenever the user loaded a target webpage, the applet would pass that URL to the server. The server would generate GoogleURL guesses, query them in Google, and then inform the applet of the results. Unfortunately, a solution like this, wherein all Google queries were performed by a central server, could quickly exhaust the 1000 alloted queries per day. The server could require that each user pass in his key, but doing so would add an inconvenience of configuration.

Another improvement to the GoogleURL Generator would, instead of being a web-accessible applet, be a side bar installed as a Mozilla plugin. The system could use a very similar user interface to the current system. As an installed plugin, the system would avoid the applet security issues. This approach would also allow each installation to use its own key.

8.1.3 Google Reordering

As a Mozilla plugin, Google Reordering currently only takes the Mozilla browsing history into account. Sometimes people use multiple web browsers, though, so it would be desirable to examine the history from other web browsers as well. This could be particularly advantageous because of knowledge gleaned about preferred domains. A difficulty that might arise is needing to determine which history files should be loaded. If multiple users share a computer using distinct profiles, their browsing histories will be kept separately. It is unclear with whom the default profile should be associated.

In the current Google Reordering, the system only considers the top ten search results when deciding what to reorder. The reason is simply that each Google query yields one page of at most ten search results. Google Reordering's ability to stabilize GoogleURLs would be improved, since even if pages dropped below the top ten results, they could still remain GoogleURLs. The issue is that multiple pages would need to be examined, and a more complicated scheme for obtaining the resulting pages would be necessary.

A disadvantage of the current Google Reordering interface is that it is overly obtrusive, and a more seamless interface would be desirable. Users can currently see the reordering taking place, whereas they should not even need to be aware that the results had been

reordered. The current version also only reorders searches that are typed into the search bar in the Google Reordering toolbar. The whole Google Reordering toolbar would be unnecessary in a more seamless interface. The system could detect whenever a Google page had been accessed, and then automatically reorder the results. Still, for bookmarking purposes, it might still be useful to have a search box incorporated into the browser that simulated running a reordered "Feeling Lucky" search on the contents of the search box.

8.1.4 Google Bookmarking System

As it stands, Google Reordering and the GoogleURL Generator attempt to accomplish several common goals. Nevertheless, these two parts are still distinct components that are not integrated with each other. Improved versions of both Google Reordering and the GoogleURL Generator could be fused together into a Google Bookmarking System. This Google Bookmarking System would contain functionality from both the GoogleURL Generator and Google Reordering, but they would both be incorporated with simpler, less interactive interfaces.

The GoogleURL Generator can be useful, but it requires the user to load it up whenever a GoogleURL is to be calculated. To complete its functionality should not require the user's full concentration. Similarly with Google Reordering, the bookmarking system could be more effective as a Mozilla plugin that operated in the background. Whenever the user visited a page, the GoogleURL Generator would begin searching for a GoogleURL in the background. Any GoogleURLs found would be displayed to the user. The system would be able to cache the GoogleURLs that had been generated. Upon revisiting a page for which a GoogleURL had previously been calculated, the system could just periodically verify the old GoogleURL, instead of having to calculate it from scratch.

In this system, the user would not have to go out of his way or expend energy trying to figure out GoogleURLs himself. They would be automatically generated, and accessible to him at a glance. As a consequence, users would regularly see the GoogleURLs for the pages they frequently used. Whenever he explicitly wanted a GoogleURL for a page, it would be within his view. Furthermore, GoogleURLs combined with Google Reordering, will stabilize themselves for frequently accessed pages. When a page's GoogleURL changes, that will be reflected in a new visit to the page. Google Reordering may continue to allow the old GoogleURL work, but a new GoogleURL would be visibly displayed for the user.

A beneficial side-effect is that users would sometimes subconsciously remember a webpage's GoogleURL from having seen it displayed in their browser, even without having intentionally created it. Readily knowing GoogleURLs off the top of their heads would allow people to more easily communicate webpage locations to others on the fly, without taking the time to learn the URL. Coupled with Google Reordering in their own browser, users would be better equipped for finding pages they had previously been accessed, and harnessing Google search for personal bookmarking.

Bibliography

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[2] Google. http://www.google.com.

[3] Google personalized search. http://labs.google.com/personalized.

[4] Internet archive. http://www.archive.org.

[5] G. Jeh and J. Widom. Scaling personalized web search, 2002.

[6] Robert C. Miller. *Lightweight Structure in Text.* PhD dissertation, Carnegie Mellon University, Computer Science Department, School of Computer Science, May 2002.

[7] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[8] Thomas A. Phelps and Robert Wilensky. Robust hyperlinks and locations. Technical report, Division of Computer Science, 2000.

[9] A. Newell S. K. Card, T.P. Moran. *Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.

[10] David Weinberger. Google URL. JOHO the Blog, October 2002. Proposed neologism; http://www.hyperorg.com/blogger/archive/2002_10_01_archive.html.