SOLVING THE BROKEN LINK PROBLEM IN WALDEN'S PATHS

A Thesis

by

ZUBIN JAMSHED DALAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2003

Major Subject: Computer Science

SOLVING THE BROKEN LINK PROBLEM IN WALDEN'S PATHS

A Thesis

by

ZUBIN JAMSHED DALAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

August 2003

Major Subject: Computer Science

ABSTRACT

Solving the Broken Link Problem in Walden's Paths. (August 2003)

Zubin Jamshed Dalal, B.E., University of Bombay, India

Chair of Advisory Committee: Dr. Richard Furuta

With the extent of the web expanding at an increasing rate, the problems caused by broken links are reaching epidemic proportions. Studies have indicated that a substantial number of links on the Internet are broken. User surveys indicate broken links are considered the third biggest problem faced on the Internet.

Currently Walden's Paths Path Manager tool is capable of detecting the degree and type of change within a page in a path. Although it also has the ability to highlight missing pages or broken links, it has no method of correcting them thus leaving the broken link problem unsolved. This thesis proposes a solution to this problem in Walden's Paths.

The solution centers on the idea that "significant" keyphrases extracted from the original page can be used to accurately locate the document using a search engine. This thesis proposes an algorithm to extract representative keyphrases to locate exact copies of the original page. In the absence of an exact copy, a similar but separate algorithm is used to extract keyphrases that will help locating similar pages that can be substituted in place of the missing page. Both sets of keyphrases are stored as additions to the page signature in the Path Manager tool and can be used when the original page is removed from its current location on the Web.

To Mum and Dad,

For being my best friends and making me the man I am today.

To Ayesha,

For being the perfect baby sister and my confidant.

To Kavita,

For loving me and completing me.

To Nani, Grandma and Grandpa,

For the love and affection they've showered on me.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all those that have helped me in the course of this thesis.

Dr. Richard Furuta guided this thesis. I would like to thank him again for being my advisor and mentor. Working with him was the crowning glory of my Master's program at Texas A&M University. Dr. Frank Shipman and Dr. Furuta gave me this opportunity to prove myself for which I am grateful. Dr. John Leggett provided valuable advice on document frequencies and data mining. I would like to thank all my committee members for the immediate help they have provided on various issues.

Unmil Karadkar, my friend and office mate, helped me extensively and bore my numerous question-asking sessions. I wouldn't have been able to complete this project if it weren't for his invaluable insights. Reetal Pai helped me with her knowledge of part-of-speech taggers making an invaluable contribution to this thesis. Avital Arora and her constant urging and calling helped me write and complete this thesis.

I would also like to thank the rest of the Walden's Paths team, Luis-Francisco Revilla, Pratik Dave and Suvendu Dash for their help and support throughout my time at the CSDL. The coffee breaks and numerous "off-topic" discussions added to the Master's experience and I'm grateful for them.

I would like to thank my mum, dad, Ayesha and Kavita, for being my pillars of support. Every minute spent on this thesis was time away from them and I love them for being as understanding as they were. Their love and attention ensured the successful and timely completion of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

With the extent of the Web expanding at an increasing rate, the problems caused by broken links are reaching epidemic proportions. Studies have indicated [Pitkow 1998] that a substantial number of links on the Internet are broken. User surveys [Pitkow and Kehoe 1998] indicate broken links are considered the third biggest problem faced on the Internet. Web site credibility studies [Fogg et al. 2001] indicate that amateurism in web site design and maintenance is considered as the surest indicator of incredibility and that broken links in a page strongly indicate amateurism.

Walden's Paths [Furuta et al. 1997] applies the concept of "hypertextual paths" to the World-Wide Web by providing "a meta-structuring mechanism that allows the affiliation of elements from many different hypertexts". A path is considered to be a meta-document, one in many ways similar to a bookmark list, which contains links to various "related" resources on the Internet. Each link is augmented with information in the form of an annotation. The annotation can include comments about the page, important areas of the page to focus on, additional reference materials and anything else that the author of the path believes will assist in the assimilation of information contained on the page.

The Walden's Paths tool was originally built for use in the K-12 school environment by teachers, who desire to collect, organize and annotate information available on the Internet for presentation to their students [Shipman et al. 1997]. The paths are intended to be guidelines and are to be used along with the Walden's Paths tool, by the students,

---

This thesis follows the style and format of *ACM Transactions on Computer-Human Interaction*.

to browse through resources earmarked for them. However students can leave the path by visiting links on the pages and can return back to the path with a single mouse-click.

Currently the Walden's Paths project encompasses two tools created and utilized for the purpose of authoring and maintaining paths, the Walden's Paths Path Author [Karadkar et al. 2000] and Path Manager [Francisco-Revilla et al. 2001] respectively. As their names indicate, the Path Author tool was built to assist authors with the creation of paths and the ability to annotate pages in the path.

Once the path is built and made public using the Path Server, the Path Manager tool is used by the authors to perform various maintenance tasks like updating annotation information and adding or removing pages from paths. The Path Manager tool is also responsible for detecting changes in paths and the pages contained in them. The Path Manager achieves this by storing document signatures, which consist of paragraph checksums, links and keywords, computed at various stages in the page's life. Path Manager stores three signatures for each page: the signature of the original page when it was added to the path, the last verified signature and the last computed signature. Path Manager then computes the signature of the current page on the fly and compares it to the three stored signatures in order to determine the degree and type of change in the page.

However there are numerous instances in which a page is moved from its original location or is no longer hosted on the Internet. These are referred to as broken links in the path and result in the well known 404 errors. Currently Path Manager can detect broken links and but has no mechanism to deal with them.

Hence as an additional feature to the path management tool in Walden's Paths, it is desired that the problem of missing pages in a path be resolved. This can be done by locating copies of the original page to replace the missing page or to offer the author a set of similar pages that might be substituted for the original page. This thesis proposes a solution to the problem of broken links in Walden's Paths.

The remainder of the thesis is organized as follows: chapter II discusses related research work on paths, Walden's Paths and solutions to the broken link problem. Chapter III discusses the need for the broken link prevention mechanism and its proposed integration with the Path Manager tool. Chapter IV describes the proposed approach. Chapter V presents the architecture, features and design decisions involved in the development of the keyphrase extraction mechanism and the subsequent exact and similar result generation mechanisms. It also contains a description of the available input arguments and the desired output requirements. Chapter VI presents an discussion based on the results of the implemented keyphrase extraction algorithm as well as the result generation mechanisms. Chapter VII concludes the thesis and proposes further work and possible applications.

CHAPTER II

RELATED WORK

This section of the thesis is broadly divided into four categories. The first describes the concept of hypertextual paths as the basis for the Walden's Paths tool. The second describes the architecture as well as the interface of the Walden's Paths tool. The third includes a detailed description of the Path Manager tool while the fourth lists previously proposed solutions to the broken link problem.

Hypertextual Paths

Hypertextual paths were conceptualized by Bush [1945] in his landmark article, "As We May Think". In the following excerpt from the article, Bush describes the creation of a path or as he calls it a trail:

> *First he runs through an encyclopedia, finds an interesting but sketchy article, leaves it projected. Next, in a history, he finds another pertinent item, and ties the two together. Thus he goes, building a trail of many items. Occasionally he inserts a comment of his own, either linking it into the main trail or joining it by a side trail to a particular item. When it becomes evident that the elastic properties of available materials had a great deal to do with the bow, he branches off on a side trail which takes him through textbooks on elasticity and tables of physical constants. He inserts a page of longhand analysis of his own. Thus he builds a trail of his interest through the maze of materials available to him.*

Numerous systems have made use of the concept of paths as proposed by Bush. Textnet proposed in [Trigg and Weiser 1986] constructed trails from an ordered list of nodes that the user followed. Systems such as NoteCards's [Halasz et al. 1987] Guided tours [Trigg

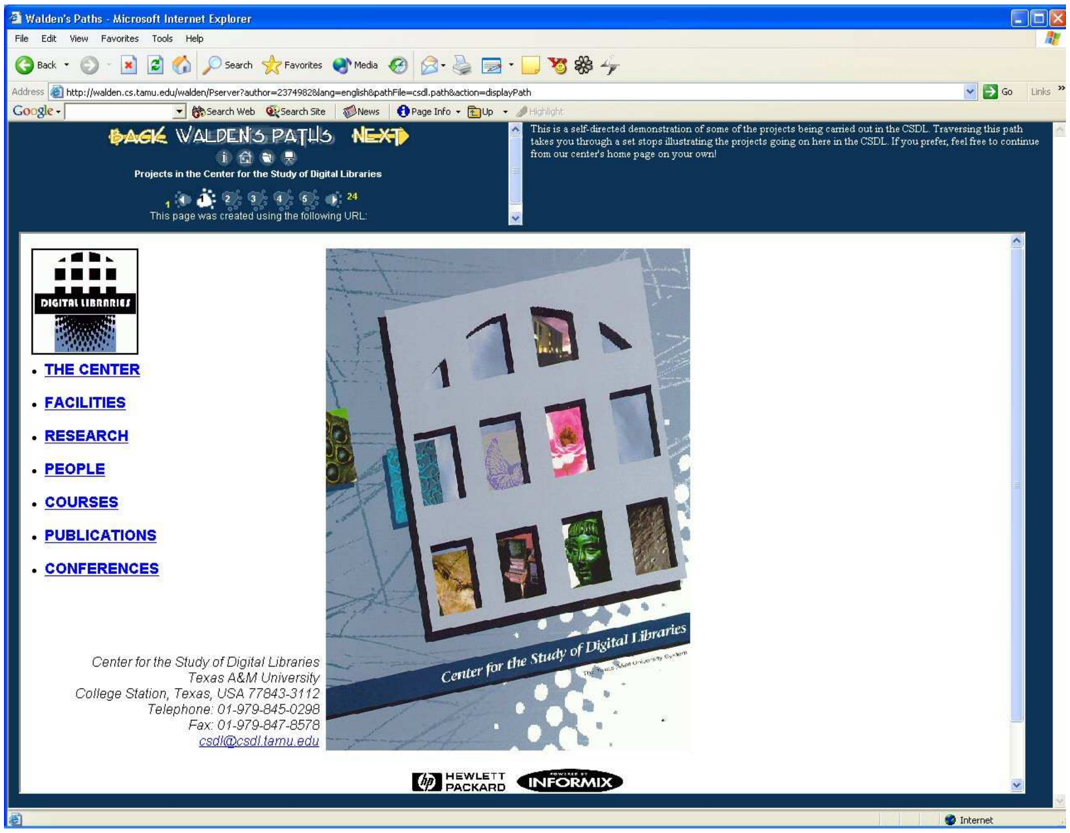
1988] allowed authors to associate a set of cards with each stop on the tour. Zellweger [1989] explored the basic issues surrounding paths such as categorizing different types of paths, their creation, edition and storage. She also introduced the concept of scripted paths allowing stops in documents and compliant applications.

Walden's Paths

Furuta, et al., applied the concept of "hypertextual paths" to the World-Wide Web by providing "a meta-structuring mechanism that allows the affiliation of elements from many different hypertexts" [Furuta et al. 1997]. Paths are very similar to extended bookmark lists but in addition each link is augmented with additional information. The annotation can include comments about the page, important areas of the page to focus on, additional reference materials and anything else that the author of the path believes will assist in the assimilation of information contained on the page.

The Walden's Paths tool was originally conceptualized for use in the K-12 school environment by teachers that desire to collect, organize and annotate information available on the Internet for presentation to their students [Shipman et al. 1997]. The paths provide a context in which to study the documents linked and are intended to be guidelines for students using them. Students can use them to browse through resources earmarked for them but can leave the path by visiting links on the pages and can return back to the path with a single mouse-click.

Figure 1 illustrates the Walden's Paths interface. The screen is divided into three sections, namely, the "Control Frame" on the top left, the "Annotation Frame" on the top right and the "Content Frame" at the bottom. The control frame contains a navigation toolbar that allows the user to navigate through the path either sequentially or by jumping to the desired page. If the user chooses to leave the path to explore other resources a "back to path" link is displayed in this frame. The annotation frame allows the author of the path to relate annotations to the page being showcased. This section is separated from the content frame so as to allow the clear differentiation between the contents of the original page and the comments of the path author.


Walden's Paths - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://walden.cs.tamu.edu/walden/Pserver?author=2374982&lang=english&pathFile=csdl.path&action=displayPath
Go Links
Google Search Web Search Site News Page Info Up Highlight
BACK WALDEN'S PATHS NEXT
Projects in the Center for the Study of Digital Libraries
1 2 3 4 5 24
This page was created using the following URL:
DIGITAL LIBRARIES
THE CENTER
FACILITIES
RESEARCH
PEOPLE
COURSES
PUBLICATIONS
CONFERENCES
Center for the Study of Digital Libraries
HEWLETT PACKARD
INFORMIX
Internet

This is a self-directed demonstration of some of the projects being carried out in the CSDL. Traversing this path takes you through a set stops illustrating the projects going on here in the CSDL. If you prefer, feel free to continue from our center's home page on your own!

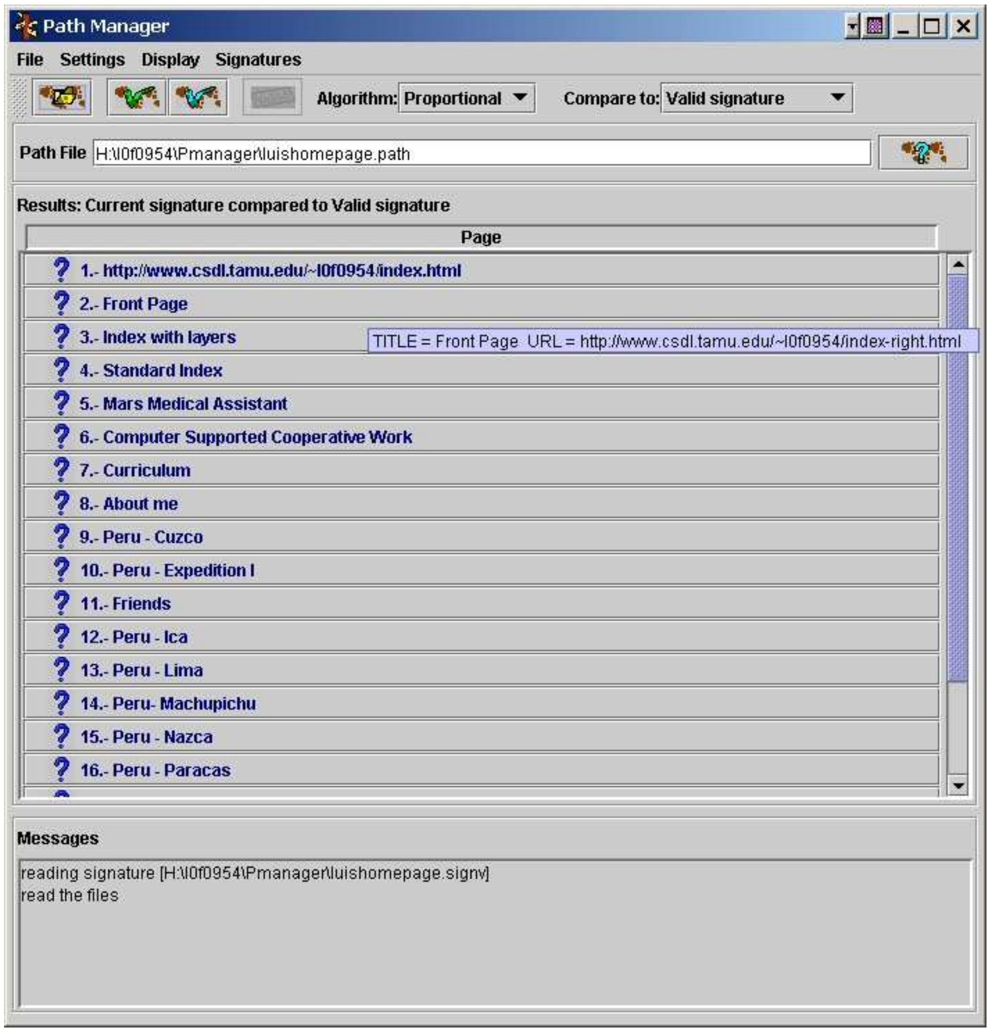
Center for the Study of Digital Libraries
Texas A&M University
College Station, Texas, USA 77843-3112
Telephone: 01-979-845-0298
Fax: 01-979-847-8578
csdl@csdl.tamu.edu

Figure 1. The Walden's Paths Interface.

In addition to the above described presentation tool, the Walden's Paths project encompasses tools created and utilized for the purpose of authoring, maintaining and publishing paths. The Walden's Paths Path Author [Karadkar et al. 2000] and Path Manager [Francisco-Revilla 2001] are two such tools. As their names indicate, the Path Author tool was built to assist authors with the creation of paths and the ability to annotate pages in the path while the Path Manager was built to enable the authors to perform various maintenance tasks like updating annotation information and adding or removing pages from paths. Since the algorithms developed as a part of this thesis are to be integrated with the Path Manager tool, we continue by describing it in greater detail.

Path Manager

The Path Manager tool of Walden's Paths was implemented in Java and is described in [Francisco-Revilla 2001]. The system takes a path file as input and checks all the pages in the path for changes. The tool can also take an HTML page like a bookmark list as input. Figure 2 shows the Path Manager interface at an initial state with a path loaded.



H:\l0f0954\Pmanager\luishomepage.path

Results: Current signature compared to Valid signature

Page
1.- http://www.csdl.tamu.edu/~l0f0954/index.html
2.- Front Page
3.- Index with layers
4.- Standard Index
5.- Mars Medical Assistant
6.- Computer Supported Cooperative Work
7.- Curriculum
8.- About me
9.- Peru - Cuzco
10.- Peru - Expedition I
11.- Friends
12.- Peru - Ica
13.- Peru - Lima
14.- Peru- Machupichu
15.- Peru - Nazca
16.- Peru - Paracas

TITLE = Front Page URL = http://www.csdl.tamu.edu/~l0f0954/index-right.html

Messages

```
reading signature [H:\l0f0954\Pmanager\luishomepage.signv]
read the files
```

Figure 2. Initial State of the Path Manager Interface.

Before the system can check for changes, it retrieves all the pages from the Web. It is at this stage that the tool encounters broken links or 404 errors. The system keeps count of the number of links tried and the number of 404 errors returned. It them re-compute a new signature for the pages returned. Each of these new signatures is compared to past signatures in order to determine the degree of change in the page.

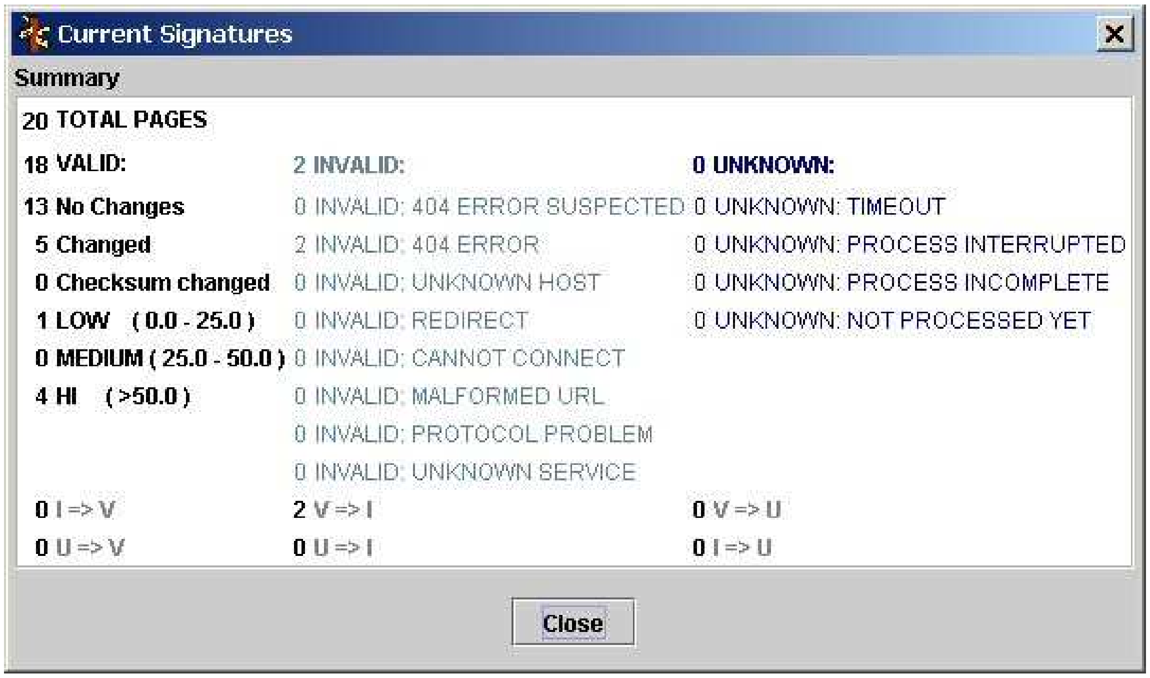
The system currently stores three past signatures for each page in the path. Each signature has a separate significance and is used to indicate specific change information regarding the page. The following are the signatures stored by Path Manager:

- Original page signature: This is the signature computed on the original page the first time the Path Manager tool is run on the path. It keeps record of the original form of the page and helps indicate the total amount of change in the page since it was added to the path.

- Last valid page signature: This is the signature pertaining to the last time the user reviewed and validated the pages in the path. This is because over time the user might update the signature of the page as the page changes. This signature represents the current state of the page in the path.

- Last computed page signature: This is the page signature computed the last time the Path Manager tool was run on the path. This signature might not be validated but helps determine the degree of change that has occurred since the last time the page or path was checked.

Figure 3 displays a status window that indicates the total number of pages checked and the number of broken links encountered.

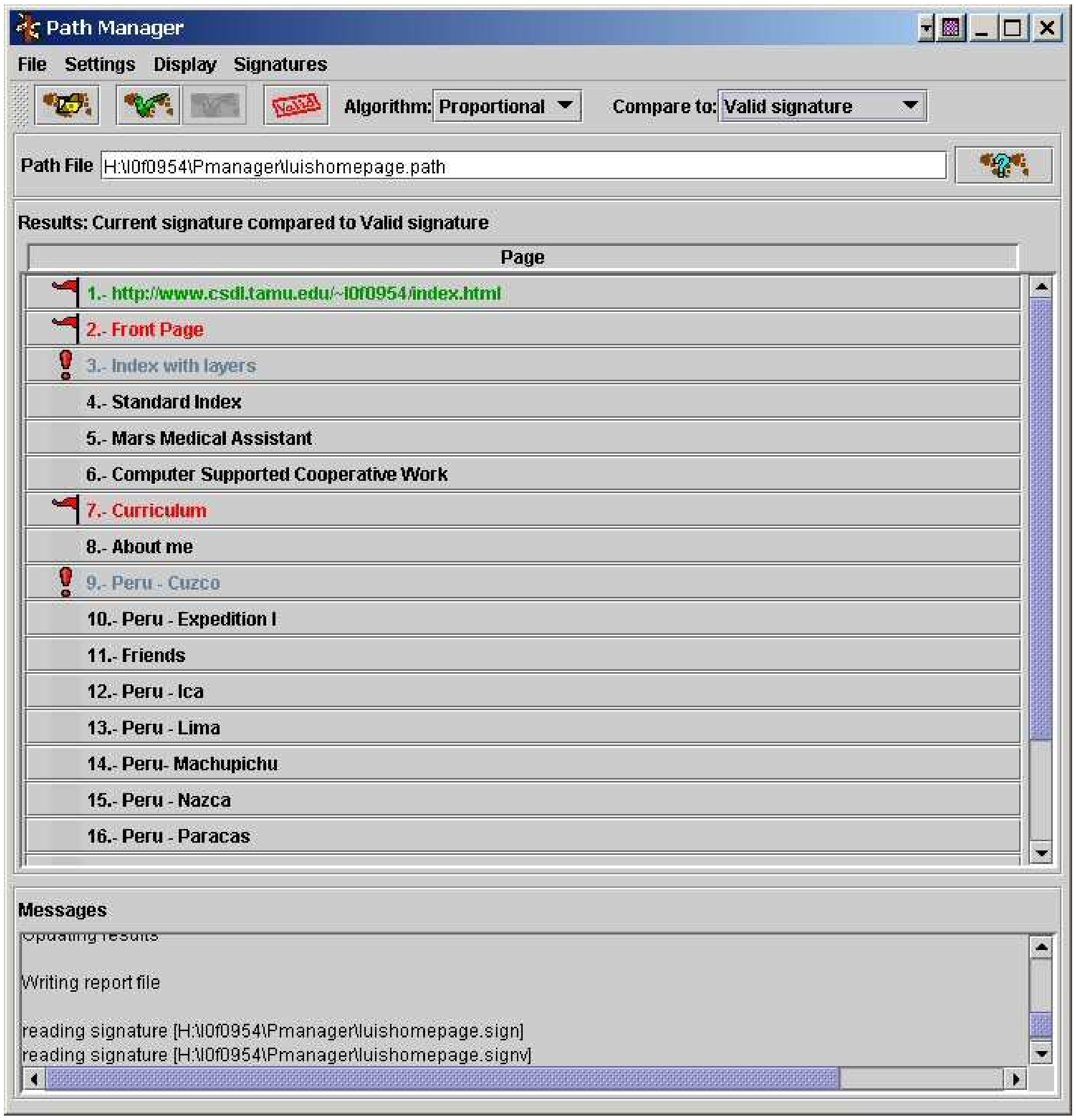


```
Current Signatures
Summary
20 TOTAL PAGES
18 VALID:                     2 INVALID:                          0 UNKNOWN:
13 No Changes                 0 INVALID: 404 ERROR SUSPECTED      0 UNKNOWN: TIMEOUT
 5 Changed                    2 INVALID: 404 ERROR                0 UNKNOWN: PROCESS INTERRUPTED
 0 Checksum changed           0 INVALID: UNKNOWN HOST             0 UNKNOWN: PROCESS INCOMPLETE
 1 LOW    ( 0.0 - 25.0 )      0 INVALID: REDIRECT                 0 UNKNOWN: NOT PROCESSED YET
 0 MEDIUM ( 25.0 - 50.0 )     0 INVALID: CANNOT CONNECT
 4 HI     ( >50.0 )           0 INVALID: MALFORMED URL
                              0 INVALID: PROTOCOL PROBLEM
                              0 INVALID: UNKNOWN SERVICE
 0 I => V                     2 V => I                            0 V => U
 0 U => V                     0 U => I                            0 I => U
                                  Close
```

Figure 3. Status Window.

Once the new page signatures have been computed the user can use either the Johnson's algorithm or the proportional algorithm to compare the new signature with any of the three previous signatures. The results of this comparison are illustrated in Figure 4.

A fluttering red flag is displayed near the page link when the page's global checksum has changed. Its absence generally indicates zero change in the page. The links are also color-coded based on the degree of change. A link colored green indicates a low degree of change, yellow indicates a medium degree and red a high degree of change in the page. A red exclamation mark to the left of the page link indicates a 404 error was returned when trying to retrieve the page indicating that a broken link. The page is also highlighted in gray to indicate that it is no longer available at its original location.



Algorithm: Proportional

Compare to: Valid signature

Path File H:\l0f0954\Pmanager\luishomepage.path

Results: Current signature compared to Valid signature

1.- http://www.csdl.tamu.edu/~l0f0954/index.html
2.- Front Page
3.- Index with layers
4.- Standard Index
5.- Mars Medical Assistant
6.- Computer Supported Cooperative Work
7.- Curriculum
8.- About me
9.- Peru - Cuzco
10.- Peru - Expedition I
11.- Friends
12.- Peru - Ica
13.- Peru - Lima
14.- Peru- Machupichu
15.- Peru - Nazca
16.- Peru - Paracas

```
Updating results

Writing report file

reading signature [H:\l0f0954\Pmanager\luishomepage.sign]
reading signature [H:\l0f0954\Pmanager\luishomepage.signv]
```

Figure 4. Overall Change Relevance Assessments.

The algorithm will have to fit into this section of the tool and get activated when a broken link is detected. It should allow the user to click on the red exclamation mark or the gray broken link and present the user with a list of alternative pages.

Solutions to the Broken Link Problem

Proposed solutions and current research can be broadly divided into two categories: author side solutions or client side solutions. Author side solutions require there to be a mechanism on the page-publishing site that keeps track of changes in the Web page and notifies users of changes to the page. In client side systems, the author is not responsible to notify users or lookup servers of page changes. The client has to locate the moved or a similar page using search engines, etc.

In author-side solutions, the author is responsible for notifying the clients or lookup servers about changes in a page. A change is considered to be anything that affects the HTML links referring to that page [Creech 1996]. If the page moves, it is the author's responsibility to maintain a forward redirect to the new page location or to report the new URL to a lookup server. Most of the proposed solutions to the broken link problem fall under the category of author side solutions. The following are some of the proposed solutions with a brief description of their mechanism.

- Forward reference approach: This approach is outlined in [Ingham et al. 1996]. In this paper, they suggest that forward referencing, call back and even an external name service to keep track of the new location of the URL. Using forward referencing is very similar to current day redirects where the user types in the original page location and a redirect is placed at this original location to the page's new location. Drawbacks of this method include having to maintain the redirect at the old location. URNs[1] also incorporate this method. Call back involves keeping track of all the documents that link to the page and notifying them with the page's new location at migration time. The main drawback of the abovementioned methods

[1]http://www.ietf.com/html.charters/urn-charter.html

is that even though the user is notified of the link change however the source document is not updated automatically with the new link. Hence one has to rely on the user updating their links with the page's new address.

- Agents approach: In [Macskassy and Shklar 1997], the authors introduce the concept of a data agent (DA) and a repository agent (RA). The DA keeps track of local files on its server and changes related to them. The RA maintains repositories of resource specifications. Hence the DA, on that server, identifies changes in a page on a server and through a link of RAs other servers are notified. A similar method is used in [Creech 1996] in building a link management system that facilitates easier page alteration by the author. However this method requires a DA to be present on each file server to keep track of changes in its local files.

- Global database approach: This seems to be an extremely popular method. It involves each solution having its own naming convention and a single or multiple lookup servers. Instead of providing a direct link to the page location, a link is provided to the lookup server with an identification number. The server performs the lookup and returns the location of the page. Shafer, et al., [1996] describe PURL (Persistent Uniform Resource Locators). PURLs are designed to replace URLs except that they point to an intermediate lookup service instead of the page location. The lookup server resolves the PURL and returns the location. PURLs remain constant although the location they point to can be altered. This method is also used in DOIs[2], URL-Minder[3] and network file URLs [Winograd 1993], each with their own unique naming conventions. However one of the major drawbacks of this system is its extreme reliance on the timely updating and maintenance of the database with recent page locations.

---

[2]http://www.doi.org

[3]http://www.netmind.com/URL-minder/URL-minder.html

Client-side solutions are far more flexible than author side solutions however are more complex to implement. Hence there are very few client side solutions available today.

- Page signature approach: One of the foremost solutions in this category is proposed by Phelps and Wilensky [2000]. A crawler is used initially to crawl through pages, where it appends a list of keywords to each URL. These keywords are selected so as to serve as the signature of the page and are used to find the page using a search engine should it be changed, moved or deleted. The keyword selection is carried out using multiple algorithms including TFIDF, which is also the algorithm used in [Francis et al. 1995]. This method allows the user to specify the number of keywords that make up the signature however [Martin and Holte 1998] indicates that 5 or 6 independent words are sufficient to retrieve a target web page.

- [Park et al. 2002] extends on the Phelps algorithm indicating that perhaps the TFIDF alone is not a suitable ordering factor and experiments with various combinations of TFs, DFs and TFIDFs to select representative keywords from the document. Some of the configurations they experimented with include TF3DF2, TF4DF1 and TFIDF4DF1. For example, TFIDF4DF1 represents a signature consisting of 4 keywords selected making use of their TFIDFs and 1 keyword selected on the basis of its DF. These keywords were used in various search engines like Yahoo, MSN and AltaVista in an attempt to locate the original page.

- Agents approach: This is described in [Winograd 1993], which involves an URL-minder bot that receives requests for URLs, polls them repeatedly and notifies the users about changes in the page.

- Hybrid approach: Another more complicated approach is a combination of both the page signature approach and the agents approach. Turney [2000] uses a learning algorithm to effectively generate keyphrases that are representative of the document and uses them as the document signature. Turney defines keyphrases as one, two or three word phrases made up of adjacent words not separated by punctuation.

CHAPTER III

NEED FOR A SOLUTION

Winograd has identified the needs for a uniform naming system, one whose links are persistent to change. He recommends that the naming mechanism should “not depend on the ongoing existence of a name authority or the willingness of that authority to provide real-time network services”. He also believes that there should be “as low overhead as possible for the 90% case”. Finally it should be introduced into the Web so as to require minimal changes and allow all existing software to continue working. In addition to ensure seamless integration with the Walden’s Paths Path Manager tool, it should not require administrative buy-in and should fit in with the current systems and Internet naming protocols.

There are many proposed solutions to the broken link problem as listed in the previous chapter; each offering varied naming conventions and lookup mechanisms. However certain drawbacks of each of these mechanisms prevent them from being used effectively with Walden’s Paths. The author-side approaches are recommendations and protocols that require author-side or publisher-side buy-in. This would mean that the authors of the pages included in our paths would have to make changes to the their systems. This is something that doesn’t fit it with the “seamless integration” requirement. Hence client-side approaches seem to be the more practical way to go.

The client-side agents approach [Turney 2000] is effective however it requires the agent to be trained before it can be used. The agent must be trained by providing it with a corpus of documents with the keywords manually selected from them. This approach doesn’t integrate seamlessly with the Walden’s Paths tool either.

The Phelps approach seems limited because it can only isolate exact copies of the page. It returns the exact result when the keywords are entered into a search engine but the other results returned are not contextually related in any way. Also since the approach uses keywords and not keyphrases, a keyphrase will appear in the keyword list as separate words thereby tipping the balance towards itself. Park's extension of the Phelps approach is also limited because it uses keywords instead of keyphrases.

Hence there existed a need for an algorithm that would isolate and store information regarding a page in the path, which could later aid us in locating a copy of that page on the Internet. In the event that no copy exists, the information should assist us in locating similar pages that could be presented to the author as probable replacements for the original page.

Before we began developing a solution, we believed it would be beneficial to consider the numerous causes of broken links. The following are some of the possible scenarios that might call for our algorithm to be needed.

Imagine that an author creates a path, checks it and publishes it using the Walden's Paths tool. A path can and generally will consist of pages authored and hosted by someone else. Therefore it is possible that the author of the page decides to move the page to a new location by changing the file name, the host name or even changing hosts. Whatever the reason for changing its location, the old page link in the path would result in the display of an error page and the Path Manager tool would report it as a broken link. A similar circumstance would arise if the author took the page offline but a mirror copy existed elsewhere on the Internet. In both these scenarios, the old page link is broken but a copy of the page exists on the Internet. It is the job of the proposed algorithm to locate the copy of the original page and substitute it for the missing page in the path.

It is possible that the author of the page decides to discontinue hosting a certain page and that no copies of the page exist in the public domain of the Internet. The user currently would have to remove the broken link from the path. In the absence of exact copies, if we could get our algorithm to located contextually similar pages on the Internet then we could provide the path author with possible substitute pages for the broken link.

There are times when a page can change drastically. This might be because the page author decided to update the content or possibly because the page location is reused for separate content. For example, a weekly happenings page could include concert information at one time and include information regarding a picnic at another time. We are aware of Path Manager's capability to detect contextual as well as structural page changes and deviation of the changed page's contents from the theme of the path. However once alerted of the change, it is up to the path author to accept the changed page or to replace it with a more suitable page. Two possible scenarios arise from these circumstances. It is possible at this time to present the author with a list of copies of the original page that exist on the Internet or pages similar to the original page as alternatives to accepting the changed page, if no copies exist.

It might seem that the last two scenarios do not fit into the category of the broken link problem. However since our algorithm has the ability to locate exact copies as well as similar pages we believe that it should be able to solve the last two problems too.

Hence the proposed algorithm should have the ability to locate copies of pages and/or contextually similar pages on the Internet with the aim of replacing missing or changed pages in a path. It should be assumed that the algorithm had the original page at a previous time to allow it to isolate information that would be used in later stages. This information will be generated when Path Manager is run on the path for the first time and will be stored as a part of the path signature

Once integrated with the Path Manager tool, the algorithm should automatically be invoked when Path Manager reports broken links in a path. It should display a list of exact copies of the page with radio buttons allowing the user to select the page that will replace the broken link in the path. There will be a *Replace* button following the links that when clicked will perform the actual task of replacing the error page. In the absence of there being any copies on the Internet, the algorithm must state the same and display a list of similar pages in the manner described above. In both cases clicking on the link itself should bring up the page in a new window, allowing the path author to view the page before adding it to the path.

CHAPTER IV

APPROACH

This chapter outlines the approach taken to solving the broken link problem in Walden's Paths. As mentioned previously the algorithm will function in two phases: a keyphrase extraction phase and a result generation phase. However it is essential to define the connotation of the word *keyphrase* in the context of this thesis.

Keyphrases

Turney defines keyphrases to be one, two or three word phrases that do not include and are not separated by stop words or punctuation marks. However since add to the definition to include one, two or three word phrases that are also not separated by HTML tags (except <LINK> or <A> tags). This is because generally phrases would be encompassed by the same set of surrounding tags. For example, you rarely see a phrase in which one of the three words is bold. In addition, the HTML link tags were considered as exceptions because there are numerous occasions where people will link words in a sentence to a dictionary or other reference sources, or might link all the words in a phrase to different resources.

In addition we hypothesize that "significant" keyphrases, those that represent the context of the document, will be of the form *noun, noun-noun*, *adjective-noun*, *noun-noun-noun*, or *adjective-noun-noun*.

Hence in the context of this thesis, keyphrases are considered to be one, two or three word phrases, not separated by punctuation marks or HTML tags (except the <LINK> and <A> tags) and of the form *noun, noun-noun*, *adjective-noun*, *noun-noun-noun*, or *adjective-noun-noun*.

Keyphrase Extraction

The keyphrase extraction phase is the first phase of the algorithm. It is executed on a set of pages in a path the first time the Path Manager tool is run on the path. Its function is to isolate a set of keyphrases from the page that would be available when the page failed to be present at its original documented location. This phase is run iteratively on every page in the path and it returns two lists of keyphrases for each page; one to help find a list of similar pages and the other a list of exact copies of the original page.

The text of the page is obtained off the Web, for at this stage none of the pages are missing. Once the entire text of the page is available it is fed into a part-of-speech tagger. A part-of-speech tagger is a program that takes as input a sentence or collection of sentences and returns them as tagged sentences. Tagged sentences are similar to the originally input sentences except that each word in the original sentence has a tag added to it as a suffix. The tag corresponds to the grammatical position of the word in a sentence. There are a number of recognized tagsets, one of the most popular being the Penn Treebank tagset [Marcus et al. 1993]. The entire tagset is included in Appendix A. Figure 5 illustrates an example of the part-of-speech tagging mechanism.

```
Input:
The northern republic of Congo contains among the largest areas of
intact tropical forest left in Africa.

Output:
The/DT northern/JJ republic/NN of/IN Congo/NNP contains/VBZ among/IN
the/DT largest/JJ areas/NNS of/IN intact/JJ tropical/NN forest/NN
left/VBD in/IN Africa/NNP ./..
```

Figure 5. Sample Part-of-Speech Tagger Output

Once the entire text of the page has been tagged, keyphrases are extracted based on the abovementioned criteria. The phrases selected are believed to be keyphrases that represent the semantic context of the page. However the number of keyphrases varies from document to document and depends on the manner in which the document is written, the context and the length of the document.

In order to prune this list of keyphrases it is essential that they be ordered or ranked in some way. In order to ensure that the phrases were ranked in a manner that was not partial to the documents length and took phrase rarity into consideration, the TFIDFs of all the phrases were calculated. TFIDFs are the product of the term frequency and the inverse document frequency of the phrase. The term frequency of the phrase is simply the total number of times that particular phrase has occurred within the context of the document. The inverse document frequency is a measure of the overall commonality or rarity of a phrase. These were calculated by posing a query to a search engine and dividing the total number of results returned by the total number of pages indexed by the search engine.

However since this method required us to query the search engines a large number of times, a cache file called a *phraselist* file was created. This file contains a list of previously occurred phrases with their IDFs. Hence once a phrase has been isolated as a keyphrase and its IDF has been calculated, it is stored in the *phraselist* file for more efficient lookup in the future. Also since the IDFs returned for the same keyphrase by different search engines differ, a *phraselist* file is created for each search engine. After the IDFs of all the keyphrases have been determined, either by querying a search engine or retrieving a previous result from the *phraselist* file, the entire *phraselist* file is written back to storage for use in a later run of the program.

Now the process of ordering the keyphrases takes two separate approaches based on their proposed future use. This means that a separate method will be used to order the keyphrases to be used to locate exact copies of the page and another for the ones to be used to locate similar pages. We begin with the one for exact pages followed by the process for similar pages.

When trying to locate exact copies of the original page, we simply order the list of keyphrases in descending order of TFIDFs. Ranking in this manner will ensure that we select the best combination of words that have occurred a large number of times within the page and/or that rarely occur in other documents. This list of keyphrases is stored in its sorted order to be used by the next phase of the algorithm.

When it comes to locating similar pages however, we require a more generic list of phrases. Simply ordering the list of keyphrases in descending order of TFIDFs might not be the solution. This is because a spelling mistake might result in an extremely low IDF indicating that the phrase is very rare and hence move it high in the keyphrase list. Since this spelling mistake generally won't occur in other documents that are similar in context, they will never be located because they don't contain the misspelled phrase. It is therefore essential to weed out phrases that have occurred below a certain threshold number of times in the page. This is done by tallying the frequencies of occurrences of all the keyphrases within the page and then not considering all those below the certain threshold. The remaining keyphrases are now ordered in descending order of TFIDFs and the list is stored to be used by the next phase of the algorithm.

Exact Result Generation

This is the second phase of the algorithm and is invoked when Path Manager detects a broken link. When this happens the missing page URL is compared and the corresponding list of keywords is obtained from the path signature. This list along with the user's preference of search engine will be used to locate exact copies of the document.

The algorithm has a certain minimum number of terms, say *n*, to be used for exact result generation as specified by the user. It also has a certain number of expected "exact" results, say *e*. The program selects the first *n* phrases from the "exact" sorted keyphrase list and queries the search engine of choice. The number of results returned by the search engine is computed. If the number of results returned by the search engine is less than *e* then the URLs of the results are parsed from the results page. These are displayed as the "exact" replacements of the missing page. However if the number of results returned is more than *e*, it indicates that our query needs to be more specific. The algorithm will then pick the next keyphrase, reform the query and re-query the search engine. Again the number of results returned are checked and the process continues until *e* or less results are returned or the keyphrase list is exhausted in which case all the available results are returned as copies of the page.

Similar Result Generation

It is possible that the above phase might return no results owing to the circumstance that no copies of the missing page exist on the Internet. In such a case, this alternate second phase will be invoked. Again a list of stored "similar" keyphrases along with the user's preference of search engine will be used to locate similar copies of the missing page.

The algorithm has a certain minimum number of terms, say *m1*, and a certain number of maximum terms, say *m2*, to be used for the similar result generation as specified by the user. It also has a certain number of expected "similar" results, say *s*. The program selects the first *m2*, phrases from the "similar" keyphrase list and queries the search engine. The number of results returned by the search engine is computed. If the number of results returned by the search engine this time are equal to or more than *s* then the URLs of the *s* results are parsed from the results page. These are displayed as pages "similar" to the original page and are suggested as replacements. However if the number of results returned is less than *s*, it indicates that the query was too specific and needs to be simplified. The algorithm discards the last keyphrase from the query and re-queries the search engine. Again the number of results returned are checked and the process continues until *s* or more results are returned or the keyphrase list falls below *m1* in which case all the available results are returned as similar pages.

CHAPTER V

THE SYSTEM

This chapter describes the architecture, features and design decisions involved in the development of the keyphrase extraction mechanism and the subsequent exact and similar result generation. It also describes the available input arguments as well as the desired output of the program.

Input Arguments

It was decided that the algorithm would be used to suggest possible replacements for all the broken links present in a path. The Path Manager tool would load the path, detect the broken links in it and would pass a list of URLs to the algorithm. Hence the algorithm would be required to take as arguments a list of URLs and would be run iteratively on each of them.

Once the keyphrases were isolated the result generation mechanisms would require them to be input into a search engine. The search engine selected was critical because different search engines have different ranking mechanisms. In addition, the path authors might have their own preference depending on their personal preferences, the context of the document and coverage of the topic by the search engine. Hence the search engine to be used for result generation would also be required as a part of the argument.

It was decided that the algorithm would support most of the well-known search engines used today so as to provide the path author with a wide range of choices. The algorithm would support search engines by Google[4], Yahoo[5], MSN[6] and Altavista[7].

---

[4]http://www.google.com

[5]http://new.search.yahoo.com

[6]http://search.msn.com

[7]http://www.altavista.com

A verbose tag was also to be included as part of the input arguments to enable detailed status reporting and numerous other messages that could be used to debug the program. The structure of the command line input arguments is outlined in Figure 6.

```
Html google | yahoo | altavista | msn URL [URL1] [URL2] … [URL3] [-v]
```

Figure 6. Structure of Input

In addition to the above mentioned command line arguments, there are certain hard-coded arguments that allow the expert to tweak the system to perform as desired. The following are a list of possible arguments with their hard-coded defaults. These use of these terms will be described in greater detail in the following chapter.

- noOfExactTerms: The minimum number of keyphrases to be used to search for exact copies. Its default value is 10 terms.
- noOfSimilarTermsMin: The minimum number of keyphrases to be used to search for similar pages. Its default value is 3 terms.
- noOfSimilarTermsMax: The maximum number of keyphrases to be used to search for similar pages. Its default value is 8 terms.
- noOfExactResults: The maximum number of exact results expected from the program. Its default value is 1 result.
- noOfSimilarResults: The maximum number of similar results expected from the program. Its default value is 10 results.
- phraseListFileName: The prefix of the file name of the file that stores the phrases and their IDFs. One file is created for each search engine with a ".<search engine name>" suffix. The default value is *phraselist*. For example, the file name of the phrase list corresponding to the Altavista search engine would be *phraselist.altavista*.

Argument Verifier

The program begins with an argument verifier that is required to ensure that all the input arguments to the program have been specified and in the correct order. The first argument that is required is the name of the search engine that is to be used when searching for the replacement pages. This could be one of four options: *google*, *yahoo*, *altavista* and *msn*. This argument is followed by a list of one or more URLs. Each URL specified corresponds to a page that is to be replaced because it is currently unavailable. The last and optional argument is the verbose argument, which when specified allows for verbose status reporting. If this module detects missing or incorrect arguments it displays appropriate error messages.

Keyphrase Extraction

The keyphrase extraction algorithm makes up the next phase of the algorithm. It is this phase that will be carried out on every page in the path, the first time the Path Manager tool is added to the path.

For each page URL specified in the input argument, the following steps are carried out. The phrase list file is opened and the entire phrase list, phrases and their respective IDFs, is loaded in a data structure called the *phraselist*. This will serve as the master phrase list for the algorithm. Now an HTML connection is created and opened and the source text of the page is obtained. An open source part-of-speech tagger, part of the openNLP[8] project, is then used to tag the entire page text with tags from the Penn Treebank tagset.

Once tagged, a modified HTML parser, created from the standard Java HTMLEditorKit HTML parser, is used to parse the tagged sentences are isolate keyphrases. The standard HTML parser had to be modified in order to accommodate the phrase extraction mechanism and also keep track of whether a phrase was within a set of tags or outside it.

[8]http://opennlp.sourceforge.net

Java's XML parser could not be used for the parsing because it lacked the ability to handle the bad, incomplete and sometimes incoherent HTML Web pages are written in. Only keyphrases that were of the form *noun, noun-noun*, *adjective-noun*, *noun-noun-noun*, or *adjective-noun-noun* were selected. If the selected keyphrase was already present in the master phrase list its term frequency was incremented by one. If it wasn't already present in the phrase list it is added to the list with term frequency initialized to one and IDF initialized to zero.

Once all the text has been parsed and all the keyphrases in the text have been accounted for, the TFIDF of all keyphrases is calculated. All the new keyphrases in *phraselist* that were present in the current document, i.e., whose term frequency was greater than zero and IDF was zero need their IDFs calculated. This is done for this select set by entering each entire phrase as a query in the search engine previously selected by the user. The IDF is calculated by dividing the number of results returned by the total number of documents indexed by the search engine. Once all the IDFs are determined, *phraselist* is saved to the phrase list file for later use. Then the TFIDF of each phrase is calculated for the document by calculating the product of the term frequency of the phrase in the document with its IDF and is written back to the "phraselist".

At this stage, two more data structures are generated namely, the *exactKeyPhraseList* and the *similarKeyPhraseList.* As their names indicate, *exactKeyPhraseList* will contain a list of keyphrases with their TFIDFs that will be used to search for exact copies of the missing page while *similarKeyPhraseList* will contain a list of keyphrases with their TFIDFs that will be used to search for similar pages.

The *exactKeyPhraseList* is generated by adding all the phrases from *phraselist* that have term frequency greater than zero, i.e. all the phrases that occurred within the current document. The *exactKeyPhraseList* is then sorted in descending order of TFIDFs and will be used by the result generation phase of the algorithm.

Before creating *similarKeyPhraseList*, a frequency-tallying module tallies the frequencies of all the phrases in *phraselist* that have occurred in the current document. It then determines the term frequency at which the cumulative frequency plot crosses eighty percent (approximately selected) of the total number of phrase occurrences. This is done to eliminate phrases that occur too few times in the context of a page. The remaining phrases whose term frequency is equal to or greater than this minimum frequency are added to the *similarKeyPhraseList*, which is also sorted in descending order of the phrases' TFIDFs. This list too will be used by the result generation phase of the algorithm.

Exact Result Generation

This phase takes as input the *exactKeyPhraseList* data structure. It also takes as input the minimum number of terms required for the "exact" page search as well as the maximum number of results expected. A vector called *topExactResults* is generated using the minimum number of keyphrases from *exactKeyPhraseList* as search terms. The results page of the search engine is parsed and the total number of the results returned is computed. If the total number of results returned is less than or equal to the maximum number of results expected then the search is a success. The results page of the search engine is parsed and the result URLs are isolated and added to *topExactResults*. These in turn are displayed as a list of URLs, copies of the missing page. In the event that the number of results is greater, the next keyphrase from *exactKeyPhraseList* is added to the search query and the process is repeated. If the number of results becomes zero at any point, the algorithm backtracks and displays the previous set of results as a list of URLs.

Similar Result Generation

This phase takes as input the *similarKeyPhraseList* data structure. It also takes as input the minimum and maximum number of terms required for the "similar" page search as well as the minimum number of results expected. A vector called *topSimilarResults* is generated using the minimum number of keyphrases from *similarKeyPhraseList* as search terms. The results page of the search engine is parsed and the total number of the results returned is computed. If the total number of results returned is more than or equal to the minimum number of results expected then the search is a success. The results page of the search engine is parsed and the result URLs are isolated and added to *topSimilarResults*. These in turn are displayed as a list of URLs, similar pages as possible replacements of the missing page. In the event that the number of results is less, the last keyphrase from search query is omitted and the process is repeated. If the number of search terms fall below the minimum allowed search terms, the algorithm backtracks and displays the previous set of results as a list of URLs.

Desired Output

The program when incorporated with the Path Manager tool must display a pop-up window when the user clicks on the broken link. The pop-up window must contain a list of exact copies of the page. In the absence of an exact copy, it must contain a list of similar pages that will serve as replacements. The keyphrases for all pages in the path are computed and stored as a part of the path signature the first time the Path Manager tool is run on the path.

However since this algorithm is currently implemented as a stand-alone program that will take as input a list of URLs, it will return for each URL, a list of exact copies (if any) and a list of similar pages (if any). In addition it will also display the list of keywords isolated from the page that result in the list of replacement pages. This is necessary to allow the user to gauge the effectiveness and accuracy of the algorithm.

In addition to these critical elements, the output will also contain basic or verbose status information that would indicate the current processing stage to the user. The verbose flag supplied as part of the input argument will be used to decide between basic and verbose status reporting. A sample output run is as shown in Figure 7.

```
*** URL: http://students.cs.tamu.edu/zdalal/robust/1.html ***

Initializing master phrase list using phrase list file.

Getting HTML.

Initializing Part-Of-Speech Tagger.

Parsing source HTML.............................................

Getting total number of documents indexed by google.

Calculating the TFIDF for phrases in the current phrase list......

Saving master phrase list to file.

Generating "exact page" keyphrase list.....................

Sorting the "exact page" keyphrase list in descending order of TFIDF.

Displaying "exact page" keyphrase list.
elephants
samburu
collars
musth
meru
gps radio
george wittemyer
radio tracking
shimba hills
movement
samples
kenya
data
conservation
project
areas
laikipia
kws
ste
wildlife

Getting exact copies of the page from google.
```

Figure 7. Sample Output

```
Displaying copies:
http://www.save-the-elephants.org/research_main.htm
http://www.csdl.tamu.edu/~suvendud/Elephants/Save%20The%20Elephants%20-
%20Research_files/research_main.htm


Tallying phrase frequencies.


Generating "similar page" keyphrase list.........................
Sorting the "similar page" keyphrase list in descending order of TFIDF.

Displaying "similar page" keyphrase list.
elephants
samburu
collars
musth
meru
gps radio
george wittemyer
radio tracking

Getting similar pages from google.

Displaying similar pages:
http://www.save-the-
elephants.org/STENewsletters/STE%20Newsletter%20Pages/update%202001%20p
age%202.htm
http://www.positioning.televilt.se/800/kod/mali/mali.htm
http://www.cnr.berkeley.edu/~georgew/Community.html
http://www.awf.org/news/11330
http://www.wild.org/western_africa/mali.html
http://www.resort-
hotel.cc/Kenya%20resort%20hotel/Elephant%20Watch%20Safaris%20kenya.htm
http://www.gristmagazine.com/week/christ032102.asp
http://nlcs.k12.in.us/bedjrhi/studentwebpages/randykennedywebpage/about
elephants1.html
```

Figure 7. Continued

CHAPTER VI

DISCUSSION

This chapter presents a discussion on the performance of the keyphrase extraction mechanism as well as the two result generation mechanisms. This solution to the broken-link problem is not yet optimum, however it is a large step in the right direction. We have the basic idea and a strong base with which to continue work and testing.

The keyphrase extraction algorithm performed as desired. It was able to retrieve the text of the page, parse it and isolate keyphrases based on the requirements specified in earlier chapters. At a glance, the keyphrases generated varied between one, two and three word phrases with no clear majority emerging. Visual examination indicates that a majority of the keyphrases represents the central context of the page at hand. Two lists of keyphrases are generated as desired, one for the exact page location mechanism and the other for the similar page location mechanism.

The desired result generation mechanisms were also implemented. Each used the above keyphrase lists to obtain a list of page URLs that were suggested replacement pages for the given missing page.

The combination of the keyphrase extraction algorithm and the exact result generation mechanism seems perfect. This function of the system was tested with test pages extracted from currently available paths previously authored by users of the Walden's Paths tool. In all the test scenarios where applicable, at least one exact copy of the page was located. In cases where numerous copies of the page existed, more than one replacement page was suggested. A page was deemed to be an exact copy of the original page by comparing it manually to the page for context, arrangement and images.

The combination of the keyphrase extraction algorithm with the similar result generation mechanism however seems to yield mixed results. This function was tested with the same set of pages that the exact result generation mechanism was tested with. In order to ensure that the results returned by both mechanisms were mutually exclusive, any results returned by the exact results generation mechanism were deleted from the list of similar results returned. The remaining results were considered to be similar replacement pages as suggested by the algorithm.

In all the test scenarios, anywhere from three to ten pages were generated as possible similar pages. In a majority of these cases, at least one page qualified as a similar replacement page.

Length of the original document had a large role to play in the process of ordering the extracted keyphrases. If the source document was too short, the word frequencies would not exhibit a good distribution resulting in the rarest ones appearing higher in the list irrespective of the actual context of the document. This tended to cause a deviation in the results from what was expected. Hence it is essential that the algorithm work around the problem of extremely concise documents.

Another important problem was created when the context of the path appeared in the picture. In some cases, even though the generated similar pages were actually similar to the original page, they did not fit into the context of the path itself. The main reason for this discrepancy is that there are very few ways of determining why an author added a particular page to a path i.e. what is the significance of the page itself to the context of the path. Hence we should be looking for pages similar to the page in the context of the path rather than the context of the page itself. One proposed method of doing this is to take into account the path author’s annotation for the page. This is a clear indicator of the author’s interpretation of the context of the page keeping in mind the context of the path.

It was conceived that maybe the algorithm could be applied to the annotation as well and room for certain keyphrases isolated from the annotation be reserved in the page keyphrase signature. However since most annotations typically run from a sentence to a short paragraph the length poses the same problem as mentioned above. Another suggestion involved checking to see if any of the isolated keywords from the page appeared in the annotation as then increasing their weight in the ordering algorithm.

Another method of weighting phrases could depend on the way they are formatted or located in the document. Bolded or italicized words and phrases could be given special important and their weight could be increased. Also by assigning a special weight to topics and headings including the page title we could better comprehend the actual context of the document. In addition, the text of page links could also be considered for special weighting.

The final issue centered on the use of text-heavy or link-heavy pages. There were certain test cases where the original page was text-heavy and the returned page was similar but link-heavy and vice-versa. Opinions are divided here as to whether these still constitute similar documents. Some believe that these can be considered since they deal with the same topic while others believe that the author might have had a special reason for selecting a text-heavy or a link-heavy page and hence it is necessary to adhere to the style of the original document.

The solution to this problem is not a simple one since there are many reasons why a document may be link-heavy. This could either be intrinsically created link-heavy if the author of the original page wanted a page of links as a cover page. However it could also be a text-heavy page that appears link-heavy because of the inclusion of large menus as is common with most current day commercial Web pages.

On of the possible solutions has to do with examining another level of links. This means that while we parse the original document we also make a list of all the pages this page links to. We then continue to run the algorithm on all those pages and form a combined keyphrases list for the cover page and all its first level linked pages. This would help us better comprehend and isolate representation keyphrases from a link-heavy page.

It is easy to detect if the algorithm locates copies of the page and this can be verified visually with ease. However when it comes to determining whether selected pages are similar to a page, it is a daunting task. This is because people have different ways of gauging similarity. If there were a similarity measure that would allow us to accurately compute the similarity based on the semantic context of the two pages that would go a long way in allowing us to tweak the algorithm to get the desired results. Such a measure if automated would also allow for the algorithm to be tested with a large number of pages thereby ensuring us that the results are statistically significant.

In the meanwhile, a formal study could be carried out where participants were allowed to gauge the similarity between sets of pages, rank selected pages in order and determine if the keyphrases returned for a document. We are aware that the algorithm is going to suggest replacement pages for missing pages and the author of the path must be satisfied with the alternatives provided to him. Hence it is essential to see if the algorithm comprehends what the author is looking for in a similar page. A formal study would test whether individuals were satisfied with the alternative pages provided and whether they would replace the page with an alternative suggested or rather remove the broken link from the path.

Finally, on completion of localized testing of the algorithm, it would need to be incorporated into the Walden's Paths Path Manager tool as described in previous chapters.

CHAPTER VII

FURTHER WORK

In lieu of the performance of the algorithm discussed in the previous chapter, this chapter suggests ways in which to improve it further.

We could experiment with considering the context of the page with respect to the path in the algorithm. If we were able to determine the reason why the author added the page to the path in the first place, we could look for pages based on that knowledge instead of only trying to locate a page similar to the missing page itself. This can be done by allotting a special additional weight to keyphrases present as part of the annotation.

In addition, keyphrases that appear in the page title, in bold or italicized should also be allocated a special weight so as to make them more significant in the ordering phase. The problem of text-heavy and link-heavy pages could easily be solved by giving special attention to links and link text. Keyphrases constituting a part of the link text could also be allotted a higher weight. All links appearing within the context of the original document should be extracted and the algorithm should be run on the set of pages as a whole to retrieve a more complete and accurate set of keyphrases.

An algorithm that could automatically measure the similarity between two pages would help greatly by avoiding human intervention in the similarity determination phase. This could speed up the process of similarity matching allowing the algorithm to be run on larger and more diverse test sets with ease.

A formal evaluation to verify the expected results would cement the algorithm's claim of being able to located contextually similar and exact pages. Finally on verification of the algorithm's performance, it would need to be incorporated into the Path Manager tool.

CHAPTER VIII

CONCLUSION

The Path Manager tool is used for maintenance of paths created and displayed using the Walden's Paths tool. The algorithm proposed in this thesis, once incorporated into the Path Manager tool, will serve to solve the problem of broken-links in paths.

It will do so by generating a list of representative keyphrases from the text of the document and storing them as a part of the document signature. These keyphrases can be retrieved at a later time when the page is no longer available and can be used to locate replacement pages using a search engine. Keyphrases are defined as one, two or three word phrases that are not separated by punctuation marks or most HTML tags.

Different methodologies are used in the algorithm to isolate the keyphrases, depending largely on whether the keyphrases are going to be used to locate exact copies of the page or similar pages. The similar page generation mechanism will be activated in the event that no exact copies of the page were located by the algorithm on the Web.

The algorithm has been successfully able to locate exact copies of the page on the Web for a test set of documents. It also has, with limited success, been able to locate similar pages on the Web. We are certain that in time with the use of a more efficient similarity measures that these results will improve further.

REFERENCES

BUSH, V. 1945. As We May Think. *The Atlantic Monthly*, 176, 1, 101-108.

CREECH, M. L. 1996. Author-Oriented Link Management. *Fifth International World Wide Web Conference*, Paris, France, May 1996, Elsevier, 1015-1025.

FOGG, B., MARSHALL, J., LARAKI, O., OSIPOVICH, A., VARMA, C., FANG, N., PAUL, J., RANGNEKAR, A., SHON, J., SWANI, P., AND TREINEN, M. 2001. What Makes Web Sites Credible? A Report on a Large Quantitative Study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, March 2001, Seattle, WA, ACM Press, New York, 61-68.

FRANCIS, P., KAMBAYASHI, T., SATO, S., AND SHIMIZU, S. 1995. Ingrid: A Self-Configuring Information Navigation Infrastructure. *4th International World Wide Web Conference*, Boston, MA, May 1995, O'Reilly, 519-538.

FRANCISCO-REVILLA, L., SHIPMAN III, F.M., FURUTA, R., KARADKAR, U., AND ARORA, A. 2001. Managing Change on the Web. In *Proceedings of First ACM/IEEE-CS Joint Conference on Digital Libraries*, June 2001, Roanoke, VA, ACM Press, New York, 67-76.

FURUTA, R., SHIPMAN III, F.M., MARSHALL, C.C., BRENNER, D., AND HSIEH, H. 1997. Hypertext Paths and the World-Wide Web: Experiences with Walden's Paths. In *Proceedings of the eighth ACM conference on Hypertext* (HT '97), Southampton, United Kingdom, April 1997, ACM Press, New York, NY, 167-176.

HALASZ, F.G., MORAN, T. R. AND TRIGG, R. H. 1987. NoteCards in a Nutshell. In *Conference Proceedings on Human Factors in Computing Systems and Graphics Interface* (CHI '87), Toronto, Ontario, April 1987, ACM Press, New York, 45-52.

INGHAM, D.B., CAUGHEY, S.J., AND LITTLE, M.C. 1996. Fixing the "Broken-Link"' Problem: The W3Objects Approach. *Fifth International World Wide Web Conference*, Paris, France, May 1996, Elsevier, 1255-1268.

KARADKAR, U., FRANCISCO-REVILLA, L., FURUTA, R., HSIEH, H., AND SHIPMAN III, F.M. 2000. Evolution of the Walden's Paths Authoring Tools. *Webnet 2000*, Oct 2000, 299-304.

MACSKASSY, S.A., AND SHKLAR, L. 1997. Maintaining information resources. In *Proceedings of the Third International Workshop on Next Generation Information Technologies* (NGITS'97), June 1997, Neve Ilan, Israel.

MARCUS, M., SANTORINI, B., AND MARCINKIEWICZ, M.A. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19, 2, 313-330.

MARTIN, J.D., AND HOLTE, R. 1998. Searching for Content-Based Addresses on the World-Wide Web. In *Proceedings of the third ACM conference on Digital Libraries*, June 1998, Pittsburgh, PA, ACM Press, New York, 299-300.

PARK, S., PENNOCK, D., GILES, C.L., AND KROVETZ, R. 2002. Analysis of Lexical Signatures for Finding Lost or Related Documents, In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, August 2002, Tampere, Finland, ACM Press, New York, 11-18.

PHELPS, T.A., AND WILENSKY, R. 2000. Robust Hyperlinks: Cheap, Everywhere, Now. In *Proceedings of the Eight International Conference on Digital Documents and Electronic Publishing (DDEP00)*, September 2000, Munich, Germany, Springer-Verlag.

PITKOW, J. 1998. *Web Characterization Activity Answers to the W3C HTTP-NGs Protocol Design Group's Questions*. HTTP-NG Web Characterization Group. http://www.w3.org/WCA/reports/1998-01-PDG-answers (January 1998), Accessed: April 25, 2003.

PITKOW, J.E. AND KEHOE, C.M. 1998. GVU's Tenth WWW User Survey, Question 11, Problems Using the Web. http://www.gvu.gatech.edu/user_surveys/survey-1998-10/tenthreport.html (May 14, 1999), Accessed: April 25, 2003.

SHAFER, K., WEIBEL, S., JUL, E., AND FAUSEY, J. 1996. Introduction to Persistent Uniform Resource Locators. In *Proceedings of the 6th Annual Conference of the Internet Society (INET '96)*, June 1996, Montreal, Canada. http://www.isoc.org/inet96/proceedings/ (March 13, 1997), Accessed: April 12, 2003.

SHIPMAN III, F.M., MARSHALL, C.C., FURUTA, R., BRENNER, D.A., HSIEH, H., AND KUMAR, V. 1997. Using Networked Information to Create Educational Guided Paths. *International Journal of Education Telecommunications*, 3, 4, 383-400.

TRIGG, R. 1998. Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment. *ACM Transactions on Information Systems*, 6, 4, 398-414.

TRIGG, R., AND WEISER, M. 1986. Textnet: A Network-based Approach to Text Handling. *ACM Transactions on Information Systems*, 4, 1, 1-23.

TURNEY, P. 2000. Learning Algorithms for Keyphrase Extraction. *Information Retrieval*, 2, 4, 303-336.

WINOGRAD, T. 1993. Stable Network File URLs as a mechanism for uniform naming. http://www.acl.lanl.gov/URI/archive/uri-93q4.messages/0496.html (December 2, 1993), Accessed: April 24, 2003.

ZELLWEGER, P. 1989. Scripted Documents: A Hypermedia Path Mechanism. In *Proceedings of the second annual ACM conference on Hypertext*, November 1989, Pittsburgh, PA, ACM Press, New York, 1-14.

APPENDIX A

PENN TREEBANK TAGSET

The tagset used in the algorithm described in this thesis is a modification of the Penn Treebank tagset, originally proposed in *Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz: Building a Large Annotated Corpus of English: The Penn Treebank*, in *Computational Linguistics, Volume 19, Number 2 (June 1993), pp. 313--330 (Special Issue on Using Large Corpora)*. Additional tags were added to the original tagset to accommodate for punctuation marks.

Tag	Interpretation
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NP	Proper noun, singular
NPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PP	Personal pronoun
PP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

VITA

Name	Zubin Jamshed Dalal
Local Address	3500 Finfeather Road, Apt. No. 1014, Bryan, TX 77801.
Permanent Address	1101 Grande, Turner Road, Bandra (W), Mumbai – 400050, India.
Education	M. S. Computer Science, Texas A&M University, 2003. B. E. Computer Engineering, University of Mumbai, India, 2001.
Publications	P. Dave, U. P. Karadkar, R. Furuta, L. Francisco-Revilla, F. Shipman, S. Dash, Z. J. Dalal. Browsing Intricately Interconnected Paths. Accepted for publication at *Hypertext 2003*, Nottingham, England, August 26-30, 2003.