Appeared in "Electronic Publishing, Artistic Imaging, and Digital Typography (7th International Conference on Electronic Publishing, EP'98)", St. Malo, France, March/April 1998, pp. 525-536. Springer Lecture Notes in Computer Science #1375

A Tool for Maintaining Multi-variant Hypertext Documents*

Shueh-Cheng Hu and Richard Furuta
{shuehu,furuta}@csdl.tamu.edu
Telephone: +1-409-845-3839 FAX: +1-409-847-8578

Center for the Study of Digital Libraries and Department of Computer Science
Texas A&M University
College Station, TX 77843–3112, USA

Abstract. With the increasing internationalization of the World-Wide Web comes a corresponding increase in the need for *multi-variant hypertext documents*—families of documents with instances that are designed based on the same theme and structure but in which the content varies, being expressed in different languages to thereby be accessible in the reader's native languages. The different variants in the document family ideally share the same hypertext structure, called a *consistent state*, but require exceptions to handle variant-unique contents. As a practical matter, differing maintainer expertise in the supported languages often means that each variant is maintained separately from the others. In this case over time the hypertext structures of the different variants gradually deviate from each other, resulting in inconsistencies. We have developed a tool to aid authors in maintaining consistency within multi-variant hypertext documents. Our tool can traverse and compare multiple variants of a hypertext document simultaneously, report inconsistencies, and keep track of all updates that may cause new inconsistencies after consistent status has already been reached. We describe our experiences with the tool on a two-language Web site that we are maintaining. Our tool discovered previously unrecognized inconsistencies that had been in our Web pages for some time. Although it still can be enhanced further in many ways, with aid of the tool, a significant amount of time can be saved on maintaining consistency in multi-variant hypertext documents.

Keywords: Hypertext, multi-variant documents, maintenance, spider, World-Wide Web.

1 Introduction

Together with Professor Eduardo Urbina of Texas A&M University's Department of Modern and Classical Languages, the authors have been working since 1995 to establish a digital library based on the works of Miguel de Cervantes

*This material is based in part on work supported by the Texas Advanced Research Program under Grant Number 99903-230.

Saavedra (1547–1616), the well-known author of *Don Quijote de la Mancha*. The project, called Cervantes Project 2001 [1], includes bibliographic records, source texts, and photographic images about Cervantes, one of the most-influential authors in Hispanic culture and literature. Consequently, one project requirement is that our Web pages be accessible both by English-speakers and also by Spanish-speakers. The two sets of Web pages are based on the same themes, and consequently mirror each other's hypertext structure and presentation format. At times the two structures merge, for example when referencing the electronic source texts in the original Spanish. Other times, there may be variant-specific differences, for example a survey directed to Spanish-speaking users of the site. We have found it is difficult to manually keep the different variants of our hypertext document consistent in terms of structure and format, especially in this frequently changing environment. There are tools [2,8,9,10] available to handle maintenance of hypertext documents, but none address the issues of maintaining consistency among multiple variants. Lack of effective tools thus motivates our work: design and implementation of a tool that can aid people in maintaining a multi-variant hypertext document consistently.

2 Problem of Maintaining Consistency

2.1 Problem Background and Influences

Updating contents and structure of hypertext documents is inevitable to keep a Web page in a fresh status. However, this kind of task is time-consuming and tedious, especially in maintaining an expanding Web site like that of the Cervantes 2001 project. The project collects information about Cervantes from international correspondents continuously. After editorial review, editing, and input (typing or OCR), new contents are added into the project's site. From time to time, the structure of the site is updated as the project develops. Maintaining consistency within the site is difficult; the situation is exaggerated since multiple variants of the same project contents must be maintained consistently. The different language variants of the Cervantes 2001 site are often maintained by different people, or by the same person at different times. Updates are made manually in individual variants, sometimes by more than one person. During periods of rapid change, the maintainer of, for example, the Spanish-language variant can forget to notify the maintainer of the English-language variant when changes occur. Even when only one person handles both variants, it is easy to forget to mirror updates in both contexts. Updates, therefore, bring previously consistent variants into a inconsistent status. Inconsistencies, and their resulting effect on information interpretation, may give rise to inconvenience, misunderstanding, or mistakes among members of a community (e.g., Cervantes scholars in our case), especially since the Web is becoming more and more a commonplace source for information.

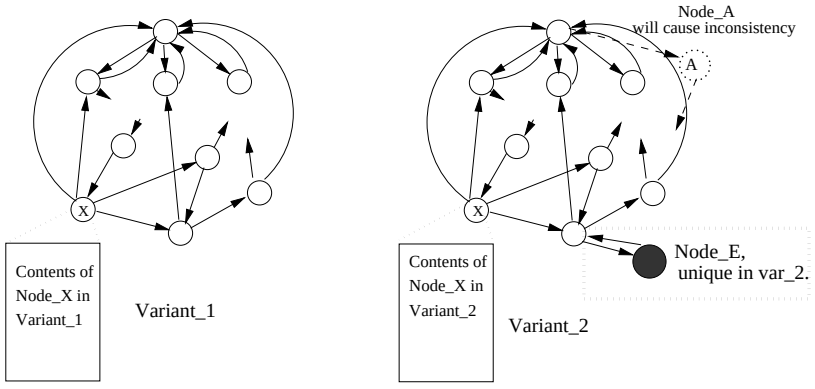

Fig. 1: Model of Multi-variant Hypertext Document

2.2 Model and Terminology

To describe related concepts and the tool clearly, it is necessary to define some terms used in this paper. Before introducing the model, we want to clarify the term "variant" and contrast it to the related term "version", which usually means "a copy of something that is slightly different from the thing being copied" in much of the literature related to software and document maintenance [8,9,10,11,12]. In this paper, we define *variant* more specifically as "a description of one thing described by different language." Figure 1 diagrams a simple multi-variant hypertext document, which will be used as example in this section. As figure 1 shows, each variant has one corresponding *network* of nodes and links. Each *node* in a network represents one document; each *link* represents a reference from one document to another document. Since they are based on the identical theme, different variants are expected to have identical hypertext structures, with only few exceptions, if any (Node_E of variant_2 in figure 1 shows one exception). In other words, networks representing different variants will be the same, except for the relatively-rare case of variant-unique nodes and corresponding links. So with the exception of Node_E in our example, if a Node_X exists in variant_1, then a counterpart Node_X and corresponding links also are found in variant_2. The contents of the two Node_Xs and their formats will be similar to each other except the languages used. Such two nodes in different networks are called *related nodes* in this paper. The tool recognizes two kinds of inconsistencies between variants: structural inconsistencies and non-structural inconsistencies.

As will be discussed, *structural inconsistencies*, can be further divided into two types. Adding/removing nodes and changing linked nodes will result in inconsistencies of the first type, which are called type_1 inconsistencies. Figure 1 shows if Node_A and corresponding links are added into variant_2 without similar action taken in variant_1, then a type_1 structural inconsistency will be created. Reordering the links in a document creates an inconsistency of the second type,

called a *type_2 structural inconsistency*; in other words type_2 structural inconsistencies reflect differences in the internal structures of related nodes. This type of inconsistency is illustrated in figure 2: Node A1 in variant_1 has three links that point to nodes B1, C1, and D1 in sequence, while its counterpart in variant_2, A2, has links that point to the related nodes in the sequence D2, C2, and B2. This type of inconsistency leads to different traversing orders in different variants.

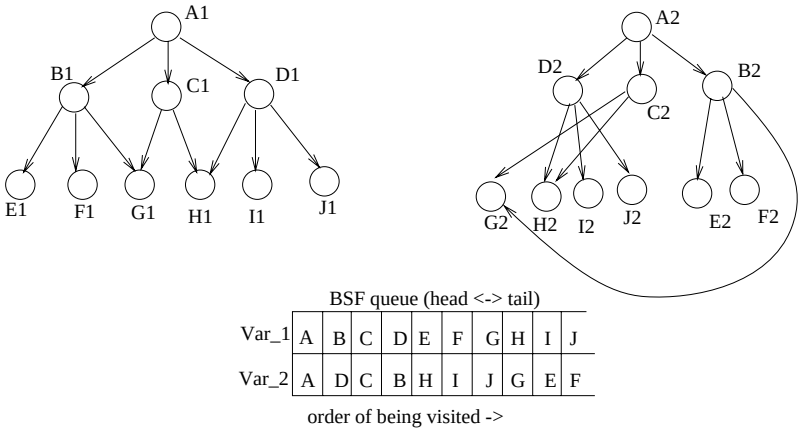

Fig. 2: Type_2 Structural Inconsistency

The second kind of inconsistency is called *non-structural inconsistency*; in other words content or formatting differences between corresponding nodes. For example, two variant hypertext documents based on the same theme presenting the analogous set of objects (excluding links) results in this kind of inconsistency if they use different HTML formatting tags, say table and list in presenting analogous items. To find the structural inconsistencies, the tool analyzes network structures of corresponding variants to look for inconsistencies between variants. Non-structural inconsistencies are harder to find because of the inherent differences caused by expression in different languages. This tool assumes sets of compared documents are aligned correctly (related), so contextual difference between variants will not be detected. However, part of any non-structural inconsistencies can be found by comparing the HTML formatting tags of the contents of related nodes in different variants.

3 Maintenance Tool Requirements

The goal of this work is to develop a tool that can help users to maintain consistent multi-variant hypertext documents. Figure 3 gives an overview of the tool's role in this maintenance task. The major requirements for the tool include: first,

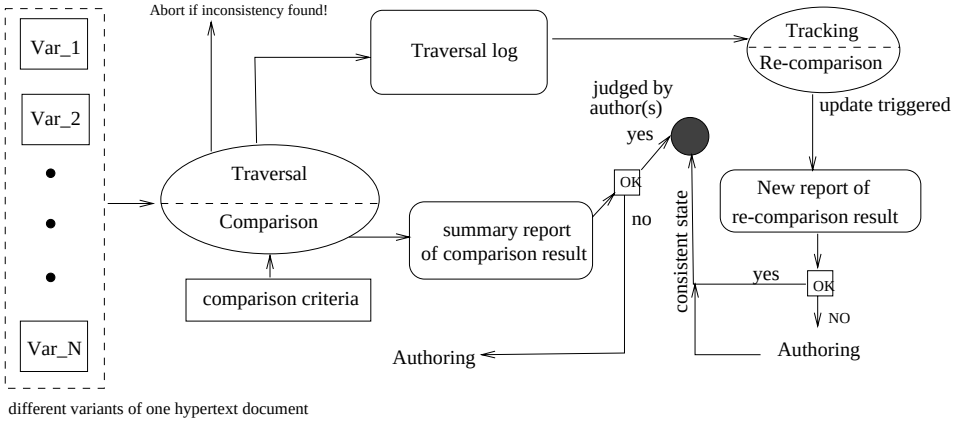

Fig. 3: Overview of the maintenance tool and its working flow

allowing the user to adjust the criteria of comparison flexibly; second, traversing networks represented by variants to find differences between variants; third, tracking all changes that were made since previous consistent status was reached to detect new inconsistencies. Each requirement will be analyzed in detail later.

3.1 Traversing Multiple Networks Automatically

To find structural inconsistencies between variants, we need to compare the hypertext structure of corresponding networks, one set of related nodes by one set of related nodes. If any inconsistency is found during traversal, the tool must report it so that the user can resolve it. After traversal, the tool should generate a log that records important information such as the entire traversal procedure and all visited sets of related nodes (documents). Regarding non-structural inconsistencies, traversal also is required. When one set of related nodes is visited, besides checking their structures, the tool must compare their contents and formats (e.g., the node's internal structure) to see if any difference exists except for language.

3.2 Adjusting Comparison Criteria Flexibly

Not every HTML tag [7] has a significant effects on a Web hypertext document, either visually or structurally. For example, `<P>` and `<BR>` both have no effect on the network structure of the document they reside in, and using either will produce the same look in browsers under many circumstances. Getting rid of those non-critical but frequently used tags can help speed up comparison between nodes in different variants. Thus, flexibility in choosing tags of interest, i.e., adjusting the scope of comparison according to the user's requirements, is important in this tool. The user can adjust the scope of comparison by adding/removing HTML tags into/from a set of tags that will be taken into account

during the comparison process; all tags that do not belong to this set will be ignored. Adjusting the scope of comparison indirectly changes the degree of tightness used to evaluate inconsistency among documents in different variants; reducing the scope of comparison means loosing the evaluation criteria. Though based on the same theme, different variants still may have unique contents to be presented; things special in one variant may have no counterparts in other variants. Thus, we need the flexibility to accommodate variant-specific contents and make variant-unique contents transparent to comparison mechanism. In this tool, a pair of special HTML comment tags are used to identify variant-specific contents: `<!-- VER_ONLY_BEGIN -->` and `<!-- VER_ONLY_END -->`. The traversal mechanism in this tool can recognize this pair of tags in hypertext documents and filter out the enclosed variant-specific contents.

3.3 Keeping Consistent Status Unchanged

After reaching a consistent status, any modifications to maintained hypertext documents need to be tracked to make sure no new inconsistencies result from those modifications. The tool must be able to detect all modifications made since the previous consistent status was reached, then determine which documents are influenced by those modifications, and finally re-compares those influenced documents to see if consistent status is changed by those modifications. If any modification results in new inconsistencies, the tool should alert the user about the appearance and origin of the new inconsistencies, thereby prompting the user to initiate the authoring steps needed to restore consistent status.

4 How the Maintenance Tool Works

Earlier, we discussed the three major mechanisms in the tool for traversal, comparison, and tracking, respectively. Besides these mechanisms, some other auxiliary files are necessary to fulfill the ultimate goal of this tool. Figure 3 presents an overview of the maintenance tool's architecture. In this section, we will describe the implementation of each of these mechanisms in more detail, explaining one mechanism and related auxiliary files in each of the following subsections.

4.1 Traversing Multiple Networks of Linked Nodes

One of the most important mechanisms in this tool is responsible for automatically traversing networks representing different variants. The traversal mechanism behaves like Web spiders, sometimes called Web Robots, which are widely used in applications like information collection for search engines [3,4,5], and for mirroring Web sites [6]. Unlike these applications, our traversal mechanism must traverse multiple related networks to collect the information on which the later comparison is based. When information is read from a node, our traversal mechanism first filters it to reduce it to a set of *effective HTML tags*. The filtering process obtains a whole document, then removes non-tag contents first,

since those contents are meaningless for comparison. Next, it removes tags that either belong to the set of ignored tags or are variant-unique. After removing these contents, the filter passes the remaining effective HTML tags to the traversal and the comparison mechanisms, which then perform structural and non-structural inconsistency detection based on effective HTML anchors and other tags. The traversal mechanism follows a breadth-first-search algorithm to traverse networks, set of related nodes by set of related nodes. Consequently, the traversal mechanism visits each set of related nodes at the same time and leaves for the next set of related nodes simultaneously. During the traversal process, it detects structural inconsistencies by comparing the numbers of out-bound links from currently visited related nodes. Except for variant-unique contents, each node in one set of related nodes should have the same number of out-bound anchors. So, any difference among numbers of out-bound anchors of related nodes will reasonably be viewed as existence of structural inconsistency. As we can detect existence of type_1 structural inconsistency if there are different numbers of out-bound links of related nodes, type_2 structural inconsistency also can be detected using the same criterion but with a slight chance of failing to detect the inconsistency. An ad-hoc proof is given in the Appendix A to explain how the type_2 structural inconsistency will be detected with the same criterion and why there is a slight chance of failing to detect the type_2 structural inconsistency.

If any structural inconsistencies are found during traversal process, the traversal mechanism will display error messages and stop traversal immediately. This restrictive traversal-stop rule prevents the traversal mechanism from getting into a confusing situation later, in which the traversal mechanism will face unrelated nodes in different variants, while the mechanism will treat them as related nodes and will compare them.

Since there may be circles of links, the traversal mechanism may face redundant sets of related nodes. To avoid wasting time on redundant comparison and producing redundant traversal log records and comparison results, all visited sets of related nodes are added into a history list, thus subsequent sets of related nodes can be passed over if they have already been compared.

During the process of traversal, the traversal mechanism generates a traversal log file that records each set of related nodes and their comparison result, in the order of being visited. This traversal log will be used by the tracking mechanism, which is responsible for keeping consistent state unchanged and will be discussed later. The format of each log record is as follows:

filename	URL_1	...	URL_N
file stores	URL of the	...	URL of the
comparison result	1st variant's node	...	N_th variant's node

The first item is summary report, followed by all dependent hypertext documents represented by one set of related nodes in networks. Each summary contains comparison result of all dependent hypertext documents, generated by the comparison mechanism. One sample traversal log is illustrated in figure 4, which shows that four sets of related nodes were visited during one traversal session,

```
#IGNORED: IGNTAGS
#Summary report SM_AAAa004XG
    http://www.csdl.tamu.edu/cervantes/eng/cbib/cibo
    http://www.csdl.tamu.edu/cervantes/spa/cbib/cibo
#
#Summary report SM_MAAa004XG
    http://www.csdl.tamu.edu/cervantes/eng/cbib/abc/
    http://www.csdl.tamu.edu/cervantes/spa/cbib/abc/
#
#Summary report SM_SAAa004XG
    http://www.csdl.tamu.edu/cervantes/eng/cbib/collabor.html
    http://www.csdl.tamu.edu/cervantes/spa/cbib/collabor.html
#
#Summary report SM_YAAa004XG
    http://www.csdl.tamu.edu/cervantes/eng/cec.html
    http://www.csdl.tamu.edu/cervantes/spa/cec.html
```

Fig. 4: Sample of traversal log

the last pair is the document `http://csdl.tamu.edu/cervantes/eng/cec.html` and the document `http://csdl.tamu.edu/cervantes/spa/cec.html`. Their comparison was stored in file: SM_YAAa004XG. Further details about the comparison mechanism and summary report are explained in the next subsection. Regarding findings of non-structural inconsistencies, the traversal mechanism can not handle the task completely by itself; it must cooperate with the comparison mechanism to detect non-structural inconsistencies.

4.2 Comparison of Contents of Documents

Whenever one set of related nodes are visited, not only structural inconsistencies will be checked, but the traversal mechanism will also invoke a comparison mechanism to do a comparison of the contents of related nodes. With one filter handling the preprocessing step of removing unimportant contents, the comparison mechanism focuses only on effective HTML tags of each node. Since the number of ignored tags usually is less than the number of tags of interest, specifying the set of ignored tags will be more convenient. Thus, all HTML tags intended to be removed are stored in one file before traversal, and the filter will remove these tags from each document. The comparison mechanism takes one variant as the reference base, then uses the Unix *diff* utility to find out differences between the reference variant and other variant(s). The comparison result of one set of related nodes is listed in a file, called the summary report in this work. One sample of the summary report is illustrated in figure 5. Since the report is generated through the *diff* utility, the contents of the summary report closely resemble the output format of *diff*. If any differences exist between a reference variant and other variant(s) at one set of nodes, the corresponding summary report lists instructions on how to change other variant(s) to the reference variant. The sample in figure 5 shows three differences between the reference variant node `http://www.csdl.tamu.edu/cervantes/eng/engtitle.html` and an-

```
# This report generated by compNdocs
# The ignored tags file: IGNTAGS
# The reference document file:
    http://www.csdl.tamu.edu/cervantes/eng/engtitle.html
#
# The comparison result between
    http://www.csdl.tamu.edu/cervantes/spa/spatitle.html and Ref. doc:
34c34
< <IMG src=http://www.csdl.tamu.edu/cervantes/ukusa_flag.gif border=0>
---
> <IMG border=0 src=http://www.csdl.tamu.edu/cervantes/span_flag.gif>
44,45c44,45
< <b>
< </b>
> <b><i>
> </i></b>
```

Fig. 5: Sample of summary report

other variant's related node `http://www.csdl.tamu.edu/cervantes/spa/spatitle.html`. Each difference is indicated by locations (line numbers) and the *ed* editor's commands used to resolve it. Authors of these hypertext documents are responsible for reviewing the summary reports to determine if the differences need to be resolved or can be ignored as insignificant.

4.3 Keeping Consistent Status Unchanged

A tracking mechanism has been designed to fulfill requirements of keeping consistent state unchanged. It reads in the log generated by the traversal mechanism. For each log record, it compares timestamps of dependent hypertext documents with timestamp of the summary_report. Whenever the timestamps indicate an update is required, the tracking mechanism invokes the comparison mechanism to re-compare all dependent documents, replacing the old summary_report with the new comparison result. Besides replacing the old summary_report, the tracking mechanism also generates a list of new summary_reports which should be investigated by authors to confirm an unchanged consistent state. If any new update results in inconsistencies, the authors must be responsible for restoring the consistent state.

5 Experiments and Discussion

We have conducted some experiments using a prototype implementation of the maintenance tool and the materials of the Cervantes 2001 project. Some inconsistencies, both structural and non-structural, were found. They probably would not be found without the tool because the necessary manual searches and comparisons are quite time-consuming and tedious. Since the tool requests all the

contents through network connection to the corresponding HTTP servers, the process of traversal and comparison took more time than we initially expected. Including the time for generation of traversal log file and all summary_reports, about 30 minutes was required to complete one session of traversal and comparison of two variants, each one with about 90 nodes. The comparison time was dominated by the time need to fetch documents via the network from the HTTP servers. This time-consuming process can be improved by caching documents in local site for following sessions' comparison.

In this tool, two different approaches have been used to handle the classification of structural and non-structural inconsistencies. An occurrence of structural inconsistency is judged by the traversal mechanism; the reason is that once we define the structural inconsistency, the traversal mechanism can detect properties of networks which lead to inconsistencies. However confirmation of non-structural consistency must be made by authors. The reason is there are many factors, some of them are subtle and hard to predict, that can contribute to differences during comparison process. Unless comparison criteria are extremely loosely defined, the comparison mechanism, which compares contents of documents strictly, will list all differences among variants in summary_reports. So, we can not view all differences found by comparison mechanism as inconsistencies, the people who designed and implemented those documents need to get involved in the confirmation of inconsistencies, since they have relevant knowledge about the theme, the design principles, and the HTML syntax. These knowledge are not easily incorporated into the comparison mechanism. Authors must read all summary_reports to confirm that discovered differences are not inconsistencies; this takes more time than we expected.

Our prototype implementation adopted the simple rule of terminating whenever the traversal mechanism detected a structural inconsistency. Although easy to implement, this of course required that the tool be rerun to detect any subsequent inconsistencies. A more robust implementation would either be able to continue from the point of discovery or would be able to checkpoint itself to reduce the cost of restarting.

6 Conclusion and Future Work

Providing multi-language variants of Web pages based on the same theme is necessary to reach international viewers. Based on the same theme and design principle, different variants of one hypertext document use different languages, but share identical structure and format, besides a few minor exceptions brought by some variant-unique contents. Deviation from the common structure and/or format, which is caused by frequent and spontaneous updates, results in confusing situations and dissonance between members of the on-line community since following the same navigation path but not being able to obtain the same information in different variants can be very confusing. This work seeks to resolve the problem of consistently maintaining multi-variant hypertext documents. The result is a tool that can traverse, compare multiple variants to find out inconsis-

tencies among variants, and detect inconsistencies caused by updates. With aid of the tool, we did save much time on consistently maintaining our two-variant Web page, although there is room for improvement, such as the performance of traversal and comparison mechanism. More intelligent comparison mechanisms can be designed to enhance the capability of judging inconsistencies, and can ease the user's burden in confirming inconsistencies. The readability of summary report also can be improved by using general instructions instead of the *ed* editor commands generated by the *diff* utility. Finally, the interface of the tool can be changed to a Web-based one. Thus, users would be able to perform operations, like adjusting comparison criteria, starting a session of traversal, viewing traversal log or comparison result, and tracking all via their Web browser. The tool focuses on the detection of hypertext structural inconsistencies, i.e., the tool assumes all of the compared documents in different languages are related (with the same theme and contents) before traversal. However, if the compared documents are not related and their hypertext structures are relative simple, then it is possible that the current tool will think these hypertext documents are in consistent state even though they have different contents. To make the tool more robust, it is necessary to add one more phase to verify the correct alignment between documents in different languages before traversal and comparison. We are exploring the feasibility of applying techniques which used in cross-language text retrieval [13,14,15] in the verification of multi-lingual documents' alignment.

References

1. Eduardo Urbina, Richard Furuta, et al. *Cervantes Project 2001*, 1997, http://www.csdl.tamu.edu/cervantes.
2. Roy T. Fielding. *Maintaining Distributed Hypertext Infrastructures: Welcome to MOMspider's Web.* Proceedings of the first International Conference on the World Wide Web(WWW94), Geneva, Switzerland, May 25-27, 1994.
3. Michael L. Mauldin. *Lycos*, 1994, http://lycos.cs.cmu.edu/.
4. Brian Pinkerton. *WebCrawler*, 1995, http://webcrawler.com/.
5. Michael Schwartz, Mic Bowman, Peter Danzig, Udi Manber. *Harvest*, 1994, http://harvest.transarc.com/.
6. Andreas Ley. *HTMLgobble*, 1996, `ftp://ftp.rz.uni-karlsruhe.de/pub/net/www/tools/htmlgobble.tar.gz`.
7. World Wide Web Consortium. *Hypertext Markup Language(HTML)*, 1997, http://www.w3.org/pub/WWW/MarkUp/.
8. Antonina Dattolo and Antonio Gisolfi. *Analytical version control management in a hypertext system.* Proceedings of ACM CIKM 94, Nov.29-Dec.2, 1994, Pages 132-139.
9. Kasper Østerbye. *Structural and Cognitive Problems in Providing Version Control for Hypertext.* Proceedings of the ACM conference on Hypertext, Nov.30-Dec.4, 1992, Pages 33-42.
10. Anja Haake. *CoVer: A Contextual Version Server for Hypertext Applications.* Proceedings of the ACM conference on Hypertext, Nov.30-Dec.4, 1992, Pages 43-52.
11. Walter F. Tichy. *RCS-A System for Version Control.* Software - Practice and Experience (SPE), volume 15, number 7, pp. 637-654, July 1985.

12. John Plaice, William W. Wadge. *A New Approach to Version Control.* IEEE Transactions on Software Engineering, Vol. 19, No. 3, pp. 268-276, 1993.
13. Landauer, T.K. and Littman, M.L. *Fully Automatic Cross-language Document Retrieval Using Latent Semantic Indexing.* Proceedings of the Sixth Annual Conference of the UW Centre for the New Oxford English Dictionary and Text Research. UW Centre for the New OED and Text Research, Waterloo, Ontario, pp. 31-38, October, 1990.
14. Dumais, S. T., Landauer, T. K. and Littman, M. L. *Automatic Cross-linguistic Information Retrieval Using Latent Semantic Indexing.* In SIGIR'96 - Workshop on Cross-Linguistic Information Retrieval, pp. 16-23, August 1996.
15. Susan T. Dumais, Todd A. Letsche, Michael L. Littman, and Thomas K. Landauer. *Automatic Cross-Language Retrieval Using Latent Semantic Indexing.* Proceedings of the AAAI Spring Symposium on Cross-Language Text and Speech Retrieval. Stanford University, pp. 18-24, March 1997.

Appendix A

Assume there are two variants of one hypertext document, and the first ten visited nodes in each variant are as shown in figure 2. Each node is labeled with a capital letter from A to J and a variant number. Two nodes labeled with the same letter are related; i.e., they are counterparts to each other. As figure 2 shows, a type_2 structural inconsistency does exist between this two variants because appear in different order. In variant_1, Node_A has three anchors pointing to three nodes, in sequence of B, C, D, while its counterpart in variant_2 points to related nodes in sequence of D, C, B. Using the criterion that can correctly detect the type_1 structural inconsistency can not tell two variants are inconsistent until the fourth node is visited because

$$F(A_1) = F(A_2), F(B_1) = F(B_2), F(C_1) = F(C_2), F(D_1) = F(D_2)$$

where $F(X)$ is the number of out-bound anchors of Node_X. However, when traversal continues up to the tenth node, the criterion will has better chance to discover the hidden inconsistency unless following equations hold:

$$F(E_1) = F(H_2), F(F_1) = F(I_2), F(G_1) = F(J_2),$$

$$F(H_1) = F(G_2), F(I_1) = F(E_2), F(J_1) = F(F_2).$$

the condition will be much more restrictive if there is no type_1 structural inconsistency between pairs of related nodes, since that means following equations hold:

$$F(E_1) = F(E_2), F(F_1) = F(F_2), F(G_1) = F(G_2),$$

$$F(H_1) = F(H_2), F(I_1) = F(I_2), F(J_1) = F(J_2).$$

Combining these two sets of equations, we know that the possibility of passing over the criterion till the tenth node visited is indeed small since it requires that each of the nodes, from E to J, have the same out-bound anchors.