NEAR-Miner: Mining Evolution Associations of Web Site Directories for Efficient Maintenance of Web Archives*

Ling Chen
L3S/University of Hannover
Appelstr. 9a
30167 Hannover, Germany
lchen@l3s.de

Sourav S Bhowmick
School of Computer Engg.
Nanyang Technological
University, Singapore
assourav@ntu.edu.sg

Wolfgang Nejdl
L3S/University of Hannover
Appelstr. 9a
30167 Hannover, Germany
nejdl@l3s.de

ABSTRACT

Web archives preserve the history of autonomous Web sites and are potential gold mines for all kinds of media and business analysts. The most common Web archiving technique uses crawlers to automate the process of collecting Web pages. However, (re)downloading entire collection of pages periodically from a large Web site is unfeasible. In this paper, we take a step towards addressing this problem. We devise a data mining-driven policy for *selectively* (re)downloading Web pages that are located in hierarchical directory structures which are believed to have *changed significantly* (e.g., a substantial percentage of pages are inserted to/removed from the directory). Consequently, there is no need to download and maintain pages that have not changed since the last crawl as they can be easily retrieved from the archive.

In our approach, we propose an off-line data mining algorithm called NEAR-*Miner* that analyzes the evolution history of Web directory structures of the original Web site stored in the archive and mines *negatively correlated association rules* (NEAR) between *ancestor-descendant* Web directories. These rules indicate the evolution correlations between Web directories. Using the discovered rules, we propose an efficient Web archive maintenance algorithm called WARM that optimally skips the subdirectories (during the next crawl) which are *negatively correlated* with it in undergoing *significant* changes. Our experimental results with real data show that our approach improves the efficiency of the archive maintenance process significantly while sacrificing slightly in keeping the "freshness" of the archives. Furthermore, our experiments demonstrate that it is not necessary to discover NEARs frequently as the mining rules can be utilized effectively for archive maintenance over multiple versions.

1. INTRODUCTION

Web archiving is the process of collecting portions of the Web by repeatedly crawling entire Web sites and adding versions of the contents and link structures of the pages in the sites to an append-only local archive for future researchers, historians, and the public [16]. The most well-known effort of this type is the work of *Internet Archive* (`www.archive.org`) which aims at building a digital library of Internet sites and other cultural artifacts in digital form. National libraries, national archives and various consortia of organizations are also involved in archiving culturally important Web content. Preserving such history of digital information is a potential gold mine for all kinds of media and business analysts [16]. In essence, a Web archive is a data warehouse for Internet contents [23].

The most common Web archiving technique uses crawlers to automate the process of collecting Web pages. Ideally, these pages should be captured whenever there is a change in any of the pages in the sites. However this is an unfeasible strategy as doing so would place unacceptable burden on the crawler as well as the Web sites. Hence, revisitation policies that seek to capture Web sites at convenient time points are more realistic for Web archiving.

Two key factors govern the quality and feasibility of Web archiving from the crawler's perspective.

- *Data transfer and bandwidth issue:* We can take a brute-force approach by periodically copying the entire original Web site over the Internet. However, such approach is not feasible for archiving large Web sites. Although it takes less than one second and 3 minutes to scan 1MB and 1GB of data, respectively, it takes 2 days for scanning 1TB and 5.5 years for a PB of data [2]. Consequently, the archiving time may span hours or even days. Often this may be longer than the available time windows as well as it may cause severe network bandwidth contention problems.
- *Freshness and coherence of archive:* It is expected that the content of the archive should remain closely synchronized with the original Web site. Good freshness can be guaranteed trivially by simply revisiting all pages very frequently. However, as mentioned above frequent scan of the whole original site is not a realistic strategy. Furthermore, even if the Web site is crawled at convenient time points, the crawler requires an extended time period to gather all pages from the site. During this period the Web site continues to evolve, causing thus incoherencies in the archive [16, 23].

In this paper we focus on the first factor. In particular, we aim to reduce data transfer at the original Web site without

*This work was primarily done when the first author was a doctoral candidate at Nanyang Technological University, Singapore.

VLDB '09, August 2009, Lyon, France

sacrificing too much freshness of the pages in the archive. We assume that the original Web site is autonomous in nature and does not allow us to run any mining algorithm or indexing mechanism on its data locally. Hence, simple indexing scheme at the original site that maintains the information of inserted, deleted, or updated data cannot be used to optimize data transfer and freshness aspects of the problem.

1.1 Our Strategy

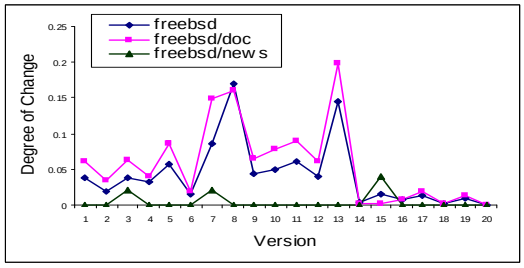
Our goal is to devise a policy for *selectively* downloading and (re)downloading of individual Web pages into the archive so as to reduce overall data transfer without significantly affecting the freshness of the archive. Naturally, there is no need to (re)download pages that have not changed since the last crawl as they can be easily retrieved from the archive. We refer to this issue as *Web archive maintenance problem*.

We assume that a Web site W to be crawled consists of n Web pages that change over time and are organized in a hierarchical *directory tree*. Note that the Web site hierarchy can be inferred both from the URL address and from a Web site database that organizes most of the dynamic URLs along an "is-a" ontology of items as described in [18]. Given such a directory tree of W to be crawled, we propose a two-tier policy for selecting pages that need to be (re)downloaded from W. At the *site-level*, we analyze the *evolutionary features* of historical Web site directory structure and select only those (sub)directories that have high chance of undergoing *significant changes* during the time period. Informally, a (sub)directory undergoes significant change if a large number of pages are inserted to or removed from it during a specific time period. (Sub)directories that are believed to have not changed are skipped. Then, at the *page-level*, we download only those pages from these selected (sub)directories whose content is believed to have changed or been created since the last run. There are several research efforts to address selective (re)downloading at the page-level in the context of search engines [8, 9, 10, 13, 17, 20, 21]. *Hence in this paper, we focus on the site-level policy.*

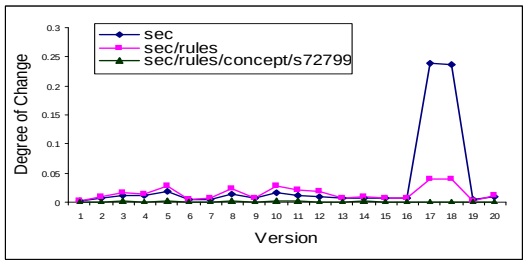
Our proposed site-level strategy for optimized Web archive maintenance consists of three steps as follows.

1. We first crawl the original Web site periodically over a period of time and retrieve and store the historical data in the archive in the form of hierarchical Web directory structure.
2. Next, we examine *certain evolutionary features* from the directory history in the archive to discover *certain rules*, which will be discussed in details in Sections 3 and 4, respectively. This mining process is carried out off-line.
3. Finally, these *rules* are used to selectively scan only relevant directories in the original Web site during the next crawl.

As there is a significant body of literature discussing efficient crawling of Web sites [8, 9, 13, 21], in this paper, we focus on the second and third steps. We now briefly describe the intuition behind the two steps. To the best of our knowledge, there exists no research work on designing optimized archive maintenance strategies by analyzing evolutionary features of historical Web site directory structures. Note that our strategy is orthogonal to the page-level strategies proposed in the literature.



(a) Degree of change of Web directories of www.freebsd.org



(b) Degree of change of Web directories of www.sec.gov

Figure 1: Degree of change of Web site directories.

1.2 Role of Evolution of Web Directories in Site-Level Archive Maintenance

Different types of evolutionary characteristics may be exploited to design optimized archive maintenance strategies. Particularly, we are interested in discovering *evolution associations* between ancestor-descendant Web directories. By evolution associations, we mean the correlations between Web directories in undergoing changes. For example, Figures 1(a) and (b) show the *Degree of Change* (Informally, it refers to the percentage of changed files in a directory.) of some directories of the Web sites at *www.freebsd.org* and *www.sec.gov*, respectively. These are extracted from the real data collected by A. Ntoulas et al. [19], who downloaded pages from 154 "popular" Web sites every week from October 2002 until October 2003, for a total of 51 weeks. In Figure 1, we show the data in the first 21 weeks. From the data, we observed the following two types of evolution associations between directories:

- *Positive evolution association.* As shown in Figure 1(a), the *Degree of Change* of the root directory *freebsd* frequently experiences peaks together with that of the subdirectory *freebsd/doc*, which means the two directories frequently undergo (relatively) significant changes together. We say the two directories have *positive association* in their evolution. It is similar for the two directories *sec* and *sec/rules* in Figure 1(b).
- *Negative evolution association.* As shown in Figure 1(a), the root directory *freebsd* rarely has (relatively) high values of *Degree of Change* together with the subdirectory *freebsd/news*. Thus, we say the two directories have *negative association* in their evolution. Similarly, the directories *sec* and *sec/rule/concepts/s72799* (or *sec/rule* and *sec/rule/ concepts/s72799*) in Figure 1(b) may have negative evolution association as well.

Thus, when maintaining the archive of a changed target

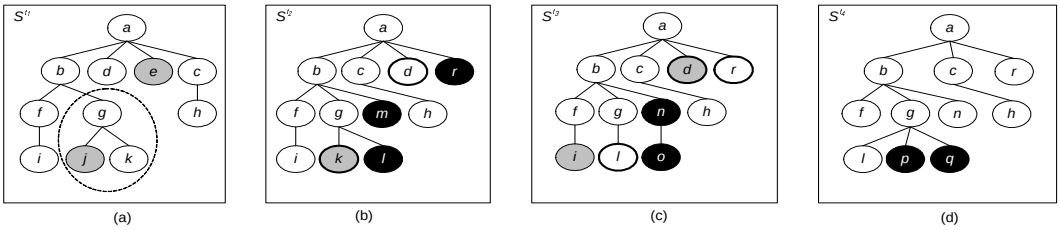


Figure 2: Historical directory trees.

directory[1], its subdirectories, which have positive evolution association with it, should not be skipped from archiving since they frequently change significantly together. On the contrary, the archive maintenance process can be optimized by skipping the subdirectories which have negative evolution association with it since these subdirectories rarely undergo significant changes together with the target directory. The intuitive meaning behind the situation that a Web directory is negatively correlated with its subdirectories in evolution is as follows. Usually, a Web site is not organized randomly. Different types of information are stored in different Web directories, which usually evolve at different frequencies with different change degrees. Changes to a directory may be attributed to subdirectories which frequently change significantly.

Therefore, in this paper, we study the evolution of a Web site and mine interesting evolution associations among Web directories. We aim to design optimized archive maintenance strategies based on discovered evolution associations. Note that our strategy leads to an *approximate* archive maintenance. That is, some updates in the original site may not be reflected immediately in the archive. Approximate archive maintenance strategies are useful in situations where the workload, archiving time, or network bandwidth contention is of main concern while the slight inconsistencies between original and the archive are tolerable. Our objective is to achieve high maintenance efficiency with minimal sacrifice in the "freshness" of archives[2].

1.3 Road Map

The rest of the paper is organized as follows. We give an overview of our proposed technique and highlight the contributions of this work in the next section. We formally define the notion of *Negative Evolution Association Rules* (NEAR) in Section 3. Section 4 presents the details of the algorithm for mining NEAR. The Web archive maintenance algorithm based on NEARs is described in Section 5. Section 6 evaluates the performance of the proposed algorithms. We review related work in Section 7. Finally, the last section concludes the paper.

2. OVERVIEW AND CONTRIBUTIONS

In order to discover interesting evolution associations (e.g., positive evolution association and negative evolution association) among Web directories of a Web site, we take the historical versions of the site as input. For example, in our study, we use the real data collected by A. Ntoulas et al. [19]. For each Web site, we represent each of its weekly version as a directory tree, which can be constructed based on the URL of pages [14, 18]. Thus, we can have a sequence of historical directory tree versions for each site. For example, the sequence of four directory tree versions shown in Figure 2 is an example input of our approach. Black nodes in the figure denote pages (or subdirectories) inserted in this version. Grey nodes denote pages (or subdirectories) deleted in the next version and nodes with bold boundary denote pages modified in this version.

We now discuss the data mining rules we aim to discover from such an input. As described in the preceding section, when archiving a target Web directory s_a, we can skip its subdirectory s_b if it has negative evolution association with s_a. On the contrary, if s_b has positive evolution association with s_a, we cannot skip it from archive maintenance. However, suppose there exists a subdirectory s_c of s_b, which has negative evolution associations with both s_a and s_b. We can still skip s_c because it rarely undergoes significant changes together with s_a and s_b (e.g., the subdirectory *sec/rules/concepts/s72799* in Figure 1(b) may have negative evolution associations with both *sec* and *sec/rules*). Hence, the objective of our approach is to discover *Negative Evolution Association Rules* (NEAR) in the form of $s_a s_b \cdots \Rightarrow \neg s_x$, where s_a through s_x is a sequence of Web directories such that each directory is an ancestor of the subsequent directory. A NEAR indicates that the Web directory on the right side of the rule rarely undergoes significant changes together with its ancestor directories on the left side of the rule.

In order to discover useful NEARs for archive maintenance, we first design a set of metrics to evaluate the interestingness of NEARs and formally define NEARs based on the metrics. Then, an efficient off-line data mining algorithm is developed for mining NEARs from a sequence of historical directory tree versions. Finally, optimized archive maintenance algorithm called WARM (**W**eb **AR**chive **M**aintenance) is developed based on discovered NEARs. Our study on real life Web data shows that NEAR-based archive maintenance strategies can improve the maintenance process significantly without losing too much "freshness" of archives. Furthermore, our experiments demonstrate that it is not necessary to discover NEARs frequently as the mining rules can be utilized effectively for archive maintenance over multiple versions.

The contribution of this study is not only at providing a novel and efficient solution to site-level Web archive maintenance problem, but also at the demonstration of how data mining technology may help solving some of the challenging issues in Web archiving. This may inspire us to further explore the application of data mining in Web archiving. In summary, the main contributions of this paper are as follows.

[1]In addition to archiving a whole Web site, it is also common to archive a particular subdirectory of a site.

[2]The discussion on remedy methods to achieve complete freshness is out of the scope of this paper.

- We introduce an approach that, to the best of our knowledge, is the first one to discover evolution associations among Web directories for Web archive maintenance.
- We propose a set of evolution metrics that quantitatively measures changes to Web directories from different aspects, based on which interesting NEARs can be identified.
- Based on the evolution metrics, we propose an efficient data mining algorithm called NEAR-*Miner* for mining NEARs.
- We describe a novel Web archive maintenance algorithm called WARM that uses NEAR to maintain archives.
- We conduct extensive experiments with real datasets to evaluate the effectiveness of the NEAR-based archive maintenance strategies. We also conduct experiments on synthetic datasets to report the performance of NEAR-*Miner*.

3. NEGATIVE EVOLUTION ASSOCIATION

In this section, we first give some preliminary definitions on directory tree structure and edit operations. Then, a set of evolution metrics is defined. Finally, *Negative Evolution Association Rule* (NEAR) among Web directories is formally defined based on the metrics.

3.1 Directory Tree Structure

As introduced in Section 1, given a Web site with directory structures, we model its file systems as a directory tree $S = \langle N, E\rangle$, where N is the set of nodes where a node represents either a Web page corresponding to a file in the Web server or a directory in the server, E is the set of edges where each edge from a parent node to a child node represents the consisting-of relationship between the corresponding directories and files. Each node carries a label which is the name of the corresponding file or directory. Each file node is also associated with a value which represents the content of file. Particularly, a node r, $r \in N$, is the root of the tree which represents the root directory of a Web site.

Accordingly, each Web subdirectory can be represented as a subtree of the directory tree. Given a directory tree $S = \langle N, E\rangle$, we say that a tree, rooted at the node i, $s_i = \langle N_i, E_i\rangle$ is a Web directory subtree of S, denoted as $s_i \preceq S$, if and only if (1) $i \in N$ (2) $N_i \subseteq N$, (3) $E_i \subseteq E$, (4) for a node $x \in N$, if $x \in N_i$ then all descendants of x (if any) must be in N_i. That is, the notion of subtree adopted by us is *bottom-up subtree* [25]. For example, given a directory tree in Figure 2 (a), the Web directory subtree s_g is highlighted by the dotted circle.

Let $\mathcal{T} = \langle t_1, t_2, \ldots, t_n\rangle$ be a sequence of time points with a particular time granularity. Given a directory tree S of some original Web site, we study the sequence of historical versions of S on $\mathcal{T}$, which is denoted as $\langle S^{t_1}, S^{t_2}, \ldots, S^{t_n}\rangle$. An example sequence of 4 historical tree versions, $\langle S^{t_1}, S^{t_2}, S^{t_3}, S^{t_4}\rangle$, is shown in Figure 2.

3.2 Edit Operations

Given two versions of a directory tree structure, an edit operation is an operation e that can be applied on a directory tree $S_1 = \langle N_1, E_1\rangle$ to produce another directory tree $S_2 = \langle N_2, E_2\rangle$, denoted as $S_1 \xrightarrow{e} S_2$. When an archiving tool compares two directories, the following three basic edit operations, *Insert*, *Delete* and *Update*, can be detected.

- **Insert**: An insertion operation, $INS(x(name, value), p)$, creates a new leaf node with a label *name* and a value *value* as a child node of node p. If x represents a directory node, its value is empty.
- **Delete**: This operation is the inverse of the insertion operation. Deletion of leaf node x, $DEL(x)$, causes x to disappear from the tree.
- **Update**: $UPD(x, new\_value)$ is an operation which changes the value of file node x to *new_value*.

Given two versions of a tree (subtree) structure, an *edit script* is a sequence of basic edit operations that convert one tree into another, $E(S^{t_i}, S^{t_{i+1}}) = \langle e_1, e_2, \ldots, e_n\rangle$. The cost of an edit script is the number of edit operations included. Then, the *edit distance* between the two versions, denoted as $D(S^{t_i}, S^{t_{i+1}})$, is the minimal cost [26] of all valid edit scripts. For example, consider the first two versions of the subdirectory tree s_g in Figure 2. The edit script with the minimal cost that transforms $s_g^{t_1}$ to $s_g^{t_2}$ is: $E(s_g^{t_1} \rightarrow s_g^{t_2}) = \langle DEL(j),\ INS(l), UPD(k)\rangle$[3]. Then, the edit distance between $s_g^{t_1}$ and $s_g^{t_2}$ is 3.

3.3 Evolution Metrics

We now define some evolution metrics to measure changes to Web directories from different aspects. Basically, we define the following three metrics, *Degree of Change* (DoC), *Frequency of Change* (FoC) and *Correlation of Change* (CoC). The first one is a *local* measure which measures how significantly a subtree changed between two versions. The last two are *global* measures which respectively measure how frequently a set of subtrees undergoes significant changes together and how two sets of subtrees are correlated in undergoing significant changes.

DEFINITION 1. (***Degree of Change***) *Given two tree versions S^{t_i} and $S^{t_{i+1}}$, let $D(S^{t_i}, S^{t_{i+1}})$ be the edit distance between the two versions. Then the Degree of Change of the tree from version S^{t_i} to $S^{t_{i+1}}$, denoted as $DoC(S^{t_i}, S^{t_{i+1}})$, is:$DoC(S^{t_i}, S^{t_{i+1}}) = \frac{D(S^{t_i}, S^{t_{i+1}})}{|S^{t_i} \uplus S^{t_{i+1}}|}$ where $|S^{t_i} \uplus S^{t_{i+1}}|$ is the size of the consolidated tree of S^{t_i} and $S^{t_{i+1}}$. A tree $S = \langle N, E\rangle$ is a consolidated tree of $S_1 = \langle N_1, E_1\rangle$ and $S_2 = \langle N_2, E_2\rangle$ if i) $N = N_1 \cup N_2$, ii) $e = (x, y) \in E$, if and only if x is parent of y in E_1 or E_2.* □

EXAMPLE 1. Consider the first two versions of subtree s_g in Figure 2 again. The consolidated tree of the two versions of s_g contains both the deleted node j and inserted node l. Hence, the size of the consolidated tree is 4. The edit distance between $s_g^{t_1}$ and $s_g^{t_2}$ is 3. Hence, $DoC(s_g^{t_1}, s_g^{t_2}) = 3/4 = 0.75$. ∎

The value of DoC ranges from 0 to 1. If a tree does not change in two versions, then its DoC is zero. If a tree is totally removed or newly inserted, then the DoC of the tree will be one. The greater the value of DoC, the more significantly the tree changes.

[3] A valid edit script with higher cost may be deleting the nodes j and k and inserting the nodes l and k which has a new value.

Another two metrics are global measures defined with respect to the whole sequence of historical tree versions. We use the following notations for the remaining definitions. Given a sequence of versions of directory tree $\Sigma = \langle S^{t_1}, S^{t_2}, \ldots, S^{t_n} \rangle$, we use $\Omega = \{s_i | \exists t_j \ (t_1 \leq t_j \leq t_n), s_i^{t_j} \preceq S^{t_j}\}$ to denote the set of directory subtrees occurring in any version of S.

DEFINITION 2. (***Frequency of Change***) *Let $X = \{s_1, s_2, \ldots, s_m\}$ be a set of subtrees, $X \subseteq \Omega$. Let the threshold of DoC be α. Then the Frequency of Change of the set X, with respect to α, denoted as $FoC_\alpha(X)$, is:*

$$FoC_\alpha(X) = \frac{\sum_{j=1}^{n-1} G_j}{n-1}, \quad where\ G_j = \prod_{i=1}^{m} G_{j_i}$$

$$and\ G_{j_i} = \begin{cases} 1, & if\ \ DoC(s_i^{t_j}, s_i^{t_{j+1}}) \geq \alpha \\ 0, & otherwise \end{cases}$$

and $n-1$ is the number of transitions between two successive versions given the sequence of n directory tree versions. □

That is, FoC_α of a set of subtrees is the fraction of transitions where the set of subtrees undergo significant changes together. The value of FoC_α ranges in $[0, 1]$. If all subtrees in the set undergo significant changes together in each transition between two successive versions, then the value of FoC_α equals to one. If the set of subtrees never undergo significant changes together in any transition, then the value of FoC_α is zero.

EXAMPLE 2. Consider the directory tree versions in Figure 2. Let $X = \{s_b, s_g\}$. The *DoC* of s_b in the sequence of three transitions are 0.5, 0.56 and 0.44 respectively. The sequence of *DoC* of s_g is 0.75, 0.67, 0.75. Let the *DoC* threshold be 0.5. Then, $FoC_{0.5}(X) = 2/3$ as both subtrees undergo significant changes in the first two transitions. ∎

We now define the metric to measure the correlation between subtrees in undergoing significant changes. Given a *DoC* threshold α, in each transition between two successive versions, a set of subtrees either undergoes significant changes together or not. This could be considered as a binary value. There are many correlation measures which are suitable for analyzing binary data, such as *ϕ-coefficient*, *odds ratio*, and the *Kappa* statistic etc. [24]. In our method, we adopt the ϕ-coefficient. Given the contingency table in Table 1, where X (Y) represents that a set of subtrees X (Y) undergoes significant changes together and $\neg X$ ($\neg Y$) represents subtrees in X (Y) do not undergo significant changes together, the ϕ-coefficient between variables X and Y can be computed by the following equation.

$$\phi(X, Y) = \frac{M f_{11} - f_{1+} f_{+1}}{\sqrt{f_{1+}(M - f_{1+}) f_{+1}(M - f_{+1})}} \qquad (1)$$

Particularly, in the context of our problem, the value of M in Table 1 equals to $n-1$, which is the total number of transitions between successive versions. Furthermore, f_{11} refers to the number of transitions where all subtrees in X and Y undergo significant changes together. Hence, f_{11} equals to $FoC_\alpha(X \cup Y) \times (n-1)$. Similarly, f_{1+} equals to $FoC_\alpha(X) \times (n-1)$ and f_{+1} equals to $FoC_\alpha(Y) \times (n-1)$. Thus, the equation (1) can be transformed as the following one, which is formally defined as *Correlation of Change* (*CoC*).

	Y	$\neg Y$	Σ_{row}
X	f_{11}	f_{10}	f_{1+}
$\neg X$	f_{01}	f_{00}	f_{0+}
Σ_{col}	f_{+1}	f_{+0}	M

Table 1: 2×2 contingency table.

DEFINITION 3. (***Correlation of Change***) *Let X and Y be two sets of subtrees, s.t. $X \subseteq \Omega$, $Y \subseteq \Omega$, and $X \cap Y = \emptyset$. Given a DoC threshold α, the Correlation of Change of X and Y, with respect to α, denoted as $CoC_\alpha(X, Y)$, is:*

$$CoC_\alpha(X, Y) =$$

$$\frac{FoC_\alpha(X \cup Y) - FoC_\alpha(X) * FoC_\alpha(Y)}{\sqrt{FoC_\alpha(X)(1 - FoC_\alpha(X)) FoC_\alpha(Y)(1 - FoC_\alpha(Y))}}$$

□

According to the definition of ϕ-coefficient, if $CoC_\alpha(X, Y)$ is greater than zero, two sets of subtrees X and Y are positively correlated in undergoing significant changes. Otherwise, they are negatively correlated in undergoing significant changes. In the rest of this paper, the subscript α in both FoC_α and CoC_α is omitted if α is understood in the context.

EXAMPLE 3. Reconsider Figure 2. Let $X = \{s_a, s_b\}$ and $Y = \{s_f\}$. The *DoC* values of s_a in the subsequence of transitions are 0.5, 0.5, 0.36. The sequence of *DoC* values of s_b are 0.5, 0.56, 0.44. The sequence of *DoC* values of s_f are 0.0, 0.0, 0, 5. Let the *DoC* threshold α be 0.5. Then $CoC(X, Y) = \frac{0-2\times1}{\sqrt{2\times(3-2)\times1\times(3-1)}} = -1$. Hence, $\{s_a, s_b\}$ and $\{s_f\}$ are negatively correlated in undergoing significant changes. ∎

3.4 Negative Evolution Association Rule (NEAR)

Based on the evolution metrics discussed above, interesting *Negative Evolution Association Rules* (NEAR) among Web directories can be defined. Recall that we are interested in NEARs in the form of $s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1}$, where s_1 through s_{k+1} is a sequence of Web directories with ancestor-descendant relationships. That is, $s_i \succ s_{i+1}$ $(1 \leq i \leq k)$. A NEAR $s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1}$ means that the sequence of subdirectories on the left side of the rule, $\langle s_1 s_2 \cdots s_k \rangle$, frequently change together. However, they rarely change together with the subdirectory on the right side of the rule, $\langle s_{k+1} \rangle$. We then first define *Positive Evolution Pattern* (PEP) as a sequence of Web directories that frequently change together and *Negative Evolution Pattern* (NEP) as two sequences of Web directories that are negatively correlated in undergoing changes. Then, NEARs can be derived from the patterns.

DEFINITION 4. (***Positive Evolution Pattern***) *Given the DoC threshold α, the FoC threshold β $(0 \leq \alpha, \beta \leq 1)$, $X = \langle s_1, s_2, \ldots, s_k \rangle$, where $s_i \succ s_{i+1}$ $(1 \leq i < k)$, is a Positive Evolution Pattern (PEP) if $FoC(X) \geq \beta$.* □

DEFINITION 5. (***Negative Evolution Pattern***) *Given the DoC threshold α, the FoC threshold β, and the CoC threshold γ $(0 \leq \alpha, \beta \leq 1, \gamma \geq 0)$, $P = \langle X, Y \rangle$, where $X = \langle s_1, s_2, \ldots, s_k \rangle$, $Y = \langle s_{k+1} \rangle$, and $s_i \succ s_{i+1}$ $(1 \leq i \leq k)$, is a Negative Evolution Pattern (NEP) if (i) X is a (PEP), i.e. $FoC(X) \geq \beta$; (ii) $FoC(X \cup Y) < \beta$; (iii) $CoC(X, Y) \leq -\gamma$.* □

Algorithm 1: The NEAR-*Miner* algorithm

Input: $\Sigma = \langle S^{t_1}, S^{t_2}, \ldots, S^{t_n} \rangle$, α, β, γ, θ
Output: A set of NEARs Γ
1 $GDT \leftarrow$ **Change_Org**(Σ, α) /* Phase 1 */;
2 $\Gamma \leftarrow$ **Rule_Gen**$(GDT.root, \beta, \theta)$ /* Phase 2 */;
3 **return** Γ

EXAMPLE 4. Let the thresholds be $\alpha = 0.5$, $\beta = 0.6$ and $\gamma = 0.5$. Based on the information in EXAMPLE 3, $\langle s_a, s_b \rangle$ in Figure 2 is a PEP because $FoC(\langle s_a, s_b \rangle) = 2/3 \geq \beta$ while $\langle \langle s_a, s_b \rangle, \langle s_f \rangle \rangle$ is a NEP because $FoC(\langle s_a, s_b, s_f \rangle) = 0 < \beta$ and $CoC(\langle \langle s_a, s_b \rangle, \langle s_f \rangle) = -1 < -\gamma$. ∎

Given a NEP $\langle \langle s_1 s_2 \cdots s_k \rangle \langle s_{k+1} \rangle \rangle$, valid NEAR $s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1}$ can be derived only if its *confidence* is no less than some specified threshold. Similar to traditional association rule mining, the confidence of a NEAR can be defined as follows.

DEFINITION 6. (***Confidence***) *Given a* NEP $P = \langle X, Y \rangle$, *where* $X = \langle s_1 s_2 \cdots s_k \rangle$ *and* $Y = \langle s_{k+1} \rangle$, *a rule in form of* $s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1}$ *can be derived, whose confidence, denoted as* $Conf(s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1})$, *can be computed as:*

$$Conf(s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1}) = \frac{FoC(X) - FoC(X \cup Y)}{FoC(X)}$$

□

Hence, the metric *confidence* measures how frequently the directory s_{k+1} doesn't change significantly when its ancestor directories s_1 through s_k undergo significant changes. The value of confidence ranges from 0 to 1. The higher the confidence, the less likely that s_{k+1} changes significantly together with its ancestor directories.

DEFINITION 7. (***Negative Evolution Association Rule***) *Given a* NEP $\langle \langle s_1 s_2 \cdots s_k \rangle \langle s_{k+1} \rangle \rangle$ *and a confidence threshold* θ, $s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1}$ *is a Negative Evolution Association Rule (*NEAR*) if* $Conf(s_1 s_2 \cdots s_k \Rightarrow \neg s_{k+1}) \geq \theta$. □

Before formally defining the problem of NEAR mining, we identify two types of *subsumption relationship* of NEARs first. As discussed in optimized archive maintenance strategies in the next section, NEARs are not equally useful for web archive maintenance. Specifically, the subsumed NEARs add no further value to our archive maintenance algorithm. Hence, we will exclude them from NEAR mining.

DEFINITION 8. (***Tail Subsumption***) *Given two* NEARs $R_1 = (X \Rightarrow \neg s_{k+1})$ *and* $R_2 = (X \Rightarrow \neg s_{k+2})$, *where* $X = \langle s_1 s_2 \cdots s_k \rangle$, R_1 *subsumes* R_2 *(or* R_2 *is subsumed by* R_1*), denoted as* $R_1 \sqsupset R_2$, *if* $s_{k+1} \succ s_{k+2}$. □

DEFINITION 9. (***Head Subsumption***) *Given a* NEAR $R_1 = (X_1 \Rightarrow \neg s_k)$ *and a* PEP X_2, *where* $X_1 = \langle s_1 s_2 \cdots s_m \rangle$ *and* $X_2 = \langle v_1 v_2 \cdots v_n \rangle$, R_1 *is subsumed by* X_2, *denoted as* $R_1 \sqsubset X_2$, *if* $\exists i (1 < i \leq m \leq n)$ *such that* $s_1 = v_1, \cdots, s_{i-1} = v_{i-1}$ *while* $s_i \prec v_i$. □

EXAMPLE 5. Consider the sequence of subtrees, $\langle s_a s_b s_f \rangle$, in Figure 2. If both $s_a \Rightarrow \neg s_b$ and $s_a \Rightarrow \neg s_f$ are valid NEARs, then the former subsumes the latter. If there exists a PEP $\langle s_a s_b \rangle$, then the rule $s_a \Rightarrow \neg s_f$ is subsumed by the pattern. ∎

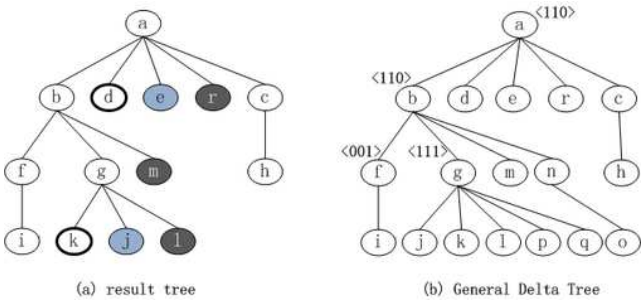


Figure 3: The *result tree* and *General Delta Tree*.

Thus, the problem of NEAR mining is defined as follows.

DEFINITION 10. (***NEAR Mining***) *Given a sequence of historical versions of a Web directory tree, DoC threshold* α, *FoC threshold* β, *CoC threshold* γ, *and the confidence threshold* θ, *the* ***problem of NEAR Mining*** *is to find a set of* NEARs *where each rule is not subsumed by any other* NEARs *(as defined in Definition* 7*) or* PEPs *(as defined in Definition* 4*).* □

4. ALGORITHM NEAR-MINER

We now present the algorithm, NEAR-*Miner*, for NEAR mining. Given a sequence of historical versions of a directory tree and a set of *DoC*, *FoC*, *CoC*, and *confidence* thresholds, the NEAR-*Miner* algorithm is shown in Algorithm 1. Basically, the algorithm can be decomposed into following two phases.

- **Phase I: Historical change organization**. Given the input as a sequence of historical versions of a directory tree, this phase detects changes between each two successive versions and organize historical changes into a special data structure, called *General Delta Tree* (*GDT*), which not only records the change information of subtrees but also preserves the ancestor-descendant relationships between subtrees.
- **Phase II: Rule generation**. The input of this phase is the *GDT* constructed in the first phase. The set of desired NEARs are discovered from the *GDT* with respect to the set of given thresholds.

4.1 Phase I: Historical Change Organization

Given a sequence of historical versions of a directory tree as the input, the changes between each two successive versions should be detected first. Many tree-structure based change detection algorithms were proposed in the literature [26] [4]. Here, we borrow the algorithm X-Diff [26], which effectively detects changes to unordered trees[4]. We optimized X-Diff based on the speciality of a Web directory tree that no two sibling nodes carry the same label. Please refer to our technical report [7] for the details. Given two tree versions, X-Diff generates a *result tree* which is a consolidated tree of the two versions. For example, Figure 3(a) shows the example result tree generated by X-Diff after detecting changes between the first two versions in Figure 2.

[4]Web directory tree is unordered because changes to orders of sibling subtrees incur no changes to the Web site.

Algorithm 2: Change_Org

Input: $\Sigma = \langle S^{t_1}, S^{t_2}, \ldots, S^{t_n} \rangle$, α
Output: A General Delta Tree GDT

```
1  Initialize GDT.root;
2  foreach i < n do
3  |  result_tree_i ← WD-Diff(S^{t_i}, S^{t_{i+1}});
4  |  delta_tree_i ← Depth_Traverse(result_tree);
5  |  Merge_Tree(GDT.root, delta_tree, root, i, α);
6  return GDT
7  procedure: Merge_Tree(x, x', i, α);
8  if DoC(s_{x'}) ≥ α then
9  |  Set the ith element of x.Bitmap = 1;
10 foreach child y' of x' do
11 |  find or create a child of y of x to match y';
12 |  call Merge_Tree(y, y', i, α);
```

Algorithm 3: Rule_Gen

Input: $GDT.root$, β, γ, θ
Output: A set of NEARs Γ

```
1  Initialize Γ = null, n = GDT.root;
2  if γ = null then
3  |  γ = 0.5
4  if FoC(s_n) ≥ β then
5  |  Initialize current pattern CP with n;
6  |  while (|R| == 0) and (γ > 0.3) do
7  |  |  R = Specific_Rule_Gen(n, CP, β, γ, θ);
8  |  |  γ − −;
9  |  Γ = Γ ∪ R;
10 foreach child x of n do
11 |  Γ = Γ∪ Rule_Gen(x, β, θ);
12 return Γ
13 procedure: Specific_Rule_Gen(x, CP, β, γ, θ);
14 foreach child y of x do
15 |  if FoC(CP ∪ s_y) ≥ β then
16 |  |  CP = CP ∪ ⟨s_y⟩,
   |  |  CP.bitmap = CP.bitmap ∩ y.bitmap;
17 |  |  Γ = Γ∪ Specific_Rule_Gen(y, CP, β, θ);
18 |  else
19 |  |  if (CoC(CP, ⟨s_y⟩) ≤ −γ) and
   |  |  (Conf(CP ⇒ ¬s_y) ≥ θ) then
20 |  |  |  Γ = Γ ∪ {CP ⇒ ¬s_y};
21 return Γ
```

Similarly, gray nodes represent the detected deletions. Black nodes represent the detected insertions and nodes with bold boundaries denote detected updates.

For each result tree, the DoC value of each Web directory subtree can be computed easily by traversing the result tree in the depth-first manner. For example, for each node x, we maintain two counters which respectively record the number of its descendants (including itself) and the number of its changed (inserted, deleted and updated) descendants. Then, the DoC value of the subtree rooted at x can be computed by dividing the latter counter by the former one.

Given a sequence of historical tree versions, we can obtain a sequence of corresponding result trees. To record the sequence of change information of each subtree concisely and maintain the ancestor-descendant relationship between Web directory subtrees, we construct a *General Delta Tree* (GDT) by merging all the result trees together. Each node in the GDT is associated with a bitmap which reflects the change information of the subtree rooted at this node.

DEFINITION 11. (***General Delta Tree***) *Given a sequence of historical tree versions Σ, a sequence of consolidated trees of each two successive versions can be obtained, $\langle S_1^{'} = \langle N_1, E_1 \rangle, S_2^{'} = \langle N_2, E_2 \rangle, \cdots, S_{n-1}^{'} = \langle N_{n-1}, E_{n-1} \rangle \rangle$, where $S_i^{'} = S^{t_i} \uplus S^{t_{i+1}}$. A General Delta Tree $GDT = \langle N, E \rangle$, where $N = N_1 \cup N_2 \cup \cdots \cup N_{n-1}$ and $e = (x, y) \in E$ only if x is a parent of y in any $E_i (1 \leq i \leq n-1)$. Each node x is associated with a bitmap such that bit i is 1 only if subtree $DoC(s_x^{t_i}, s_x^{t_{i+1}}) \geq \alpha$.* □

EXAMPLE 6. Consider the sequence of historical tree versions in Figure 2. Let the DoC threshold be $\alpha = 0.5$. The constructed GDT is shown in Figure 3(b). For simplicity, we only show the bitmaps of several nodes. For example, the bitmap of the node b indicates that the DoC value of the subtree s_b is no less than the 0.5 in the first two transitions between successive versions. ∎

The complete algorithm of Phase I, *Change_Org*, is shown in Algorithm 2. Firstly, the GDT is initialized. Then, for each two successive Web directory tree versions, we use the modified version of X-Diff, called WD-Diff, to detect changes. The function *Depth_Traverse* traverses the result tree to calculate DoC values for each subtree. Then, we merge the tree containing DoC information with the GDT. The complexity of this phase can be computed as follows. Let $|S|$ be the maximum tree size of all historical versions. According to [7], the complexity of **WD-Diff** is $O(|S| \log(|S|))$. In the worst case, the complexity of $Depth\_Traverse$ is $O(2|S|)$. The complexity of the merging process is $O(2|S|)$ as well. Hence, the total complexity of phase I is $O((n-1) \times |S| log(|S|))$.

4.2 Phase II: Rule Generation

The input of this phase is the GDT and the set of thresholds. We aim to discover a set of NEARs where each rule satisfies the thresholds and is not subsumed by any other NEARs or PEPs.

If the threshold of CoC γ is not given specifically, our algorithm decides it automatically according to Cohen's study [12] on the strength of the ϕ-coefficient value. The correlation is strong if CoC is above 0.5, moderate if CoC is around 0.3, and weak if CoC is around 0.1. Hence, our algorithm sets the initial threshold of CoC γ as 0.5 if it is not given by users. If no NEAR is discovered with respect to this value and γ is greater than 0.3, γ is reduced progressively. Similar to traditional association rule mining, the selection of other thresholds is application-dependent.

The complete set of desired NEARs Γ can be divided into disjoint partitions, $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \ldots \cup \Gamma_k$, such that each partition contains rules starting from the same subtree. That is, $\forall$ $R_x = (s_x s_{x+1} \ldots s_{m-1} \Rightarrow \neg s_m) \in \Gamma$ and $R_y = (s_y s_{y+1} \ldots s_{n-1} \Rightarrow \neg s_n) \in \Gamma$, if $s_x = s_y$ then $R_x, R_y \in \Gamma_i$; otherwise, $R_x \in \Gamma_i, R_y \in \Gamma_j$, *where* $i \neq j$ $(1 \leq i, j \leq k)$. Then, the problem of rule generation can be decomposed into a set

of subproblems where each mines rules of a particular partition. Algorithm 3 shows the general algorithm of *Rule_Gen*, which calls the function *Specific_Rule_Gen* to mine NEARs of a particular partition.

When mining NEARs starting from a particular subtree, we aim to directly discover rules where each is not subsumed by any other NEARs or PEPs, instead of performing some filtering process afterwards. For this purpose, we employ the following three strategies. *Firstly, when mining* NEARs *starting from a subtree s_x, candidate* NEPs *are generated and examined by traversing s_x in a depth-first manner.* For example, as shown in Algorithm 3, *Specific_Rule_Gen* performs a depth-first traversal on the subtree rooted at node x. *Secondly, once a* PEP *is discovered, candidate* NEPs *must be extended from this* PEP. For example, Line 3 of *Specific_Rule_Gen* updates the candidate NEP CP by growing itself with the current subtree s_y. This strategy prevents generating NEARs which are subsumed by any other PEPs. *Thirdly, once a* NEAR *is discovered, we stop traversing nodes deeper than the current one.* For example, Line 6 of *Specific_Rule_Gen* discovers a NEAR and stops there without iteratively calling itself. This strategy keeps from generating NEARs that are subsumed by any other NEARs.

Lemma 1. No NEAR *which is subsumed by any* PEP *or any other* NEARs *will be discovered by* NEAR-*Miner*. □

PROOF. Suppose there is a NEAR $R1 = (s_1 s_2 \cdots s_m \Rightarrow \neg s_{m+1})$, which is head-subsumed by a PEP $X_2 = \langle v_1 v_2 \cdots v_n \rangle$. Based on the definition of head subsumption, $\exists x (1 < x \leq m \leq n)$, *s.t.* $s_1 = v_1, \cdots, s_{x-1} = v_{k-1}$, and $s_x \prec v_x$. Since X_2 is a PEP, it is easy to prove that $X_1 = \langle v_1 \cdots v_{x-1} \rangle$ is a PEP as well (this is because of the downward closure property of the FoC metric). According the Strategy 1, X_1 is discovered earlier than X_2. And after discovering X_1, candidate pattern $\langle v_1 v_2 \cdots v_x \rangle$ will be examined before $\langle s_1 s_2 \cdots s_x \rangle$. Since X_2, $\langle v_1 v_2 \cdots v_x \rangle$, is a PEP also, $\langle s_1 s_2 \cdots s_x \rangle$ will not be generated. Otherwise, it contradicts to Strategy 2. Hence, no head-subsumed NEARs will be generated. Given a NEAR $R_1 = (X \Rightarrow \neg s_m)$, if another NEAR $R_2 = (X \Rightarrow \neg s_{m+1})$, which is tail-subsumed by R_1, is discovered, then $s_{m+1} \prec s_m$. Then, it contradicts with Strategy 3. Hence, there is no tail-subsumed rules that will be discovered. Therefore, neither head-subsumed nor tail-subsumed rules will be generated by NEAR-*Miner*. □

The complexity of this phase can be computed as follows. For each node x in the GDT, we need to call the *Specific_Rule_Gen* method which traverses the subtree s_x. Hence, each node in GDT is traversed at most m times, where m is the depth of the node. Then, let the depth of GDT be $dep(GDT)$, the complexity is $O(|GDT| \times dep(GDT))$.

5. WEB ARCHIVE MAINTENANCE

We now present the algorithm called WARM (**W**eb **AR**chive **M**aintenance) for maintaining web archives based on discovered NEARs. Recall that as an approximate maintenance strategy, it aims to achieve high archive maintenance efficiency while keeping the loss of freshness of archives as little as possible. Hence, the basic strategy is that when maintaining the archive of a Web directory s_x, we do not skip any of its subdirectories which have positive evolution associations with it. We also do not skip any subdirectories which have neither positive nor negative evolution association with it. Thus, only subdirectories which appear on the right-side of NEARs starting with s_x will be skipped. However, NEARs starting from s_x are not equally useful. Some of them are redundant with respect to others while some of them may even cause detriment to the quality of archives. Particularly, we identify the following two situations.

Algorithm 4: WARM

Input: Target directory s_x, Σ, α, β, γ, θ

```
1  Γ = NEAR-Miner(s_x, Σ, α, β, γ, θ);
2  Initialize P = ⟨s_x⟩;
3  Depth_Archive(x, P, Γ);
4  procedure: Depth_Archive(x, P, Γ);
5  foreach child y of x do
6      if y is a leaf node then
7          archive y;
8      else
9          if (∃ a rule P ⇒ ¬s_y ∈ Γ) then
10             skip archiving s_y;
11         else
12             if (∃ a rule P ∪ s_y ⇒ ¬s_z ∈ Γ) then
13                 P = ⟨s_x, s_y⟩;
14             Depth_Archive(y, P, Γ);
```

- *First, if a* NEAR *is subsumed by another* NEAR*, then the subsumed rule adds no value to optimized archive maintenance.* For example, if both $s_x \Rightarrow \neg s_y$ and $s_x \Rightarrow \neg s_z$ are valid NEARs, where $s_z \prec s_y$, then the subdirectory s_z will be skipped based on the first rule as well as the second rule.

- *Second, if a* NEAR *is subsumed by a* PEP*, then the subsumed rule is either redundant or excessive with respect to other rules in terms of archive maintenance.* For example, given a NEAR $s_x \Rightarrow \neg s_z$ and a PEP $\langle s_x s_y \rangle$, where $s_z \prec s_y$, if $s_x s_y \Rightarrow \neg s_z$ is a valid NEAR also, then the first rule is redundant. Otherwise, archive maintenance strategies based on the first rule maybe too excessive because the subdirectory s_y, which contains s_z, should not be skipped based on the PEP (Our experimental results in next section investigate the excessiveness of such rules).

Since our NEAR mining algorithm discovers rules which are not subsumed by any other NEARs or PEPs, the mining results can be used in maintenance directly.

Algorithm 4 shows our archive maintenance algorithm WARM. Basically, it first discovers NEARs starting with the target Web directory. Then, it performs a top-down traversal on the directory and either skips a subdirectory based on NEARs or archive pages using existing tools. Note that in the algorithm, we assume every parent directory is changed when traversing the directory tree in consideration of minimizing the loss in freshness.

Two NEARs are *contradictable* if one rule indicates some subdirectory should be archived while the other indicates that it should be skipped. Then the following lemma holds.

Lemma 2. At each node, there exists no contradictable NEARs *that can be used.* □

PROOF. For each node, the subtree rooted at the node either isn't involved in any NEAR or is involved in some NEAR.

If it is involved in some NEAR, it either occurs in the head of the NEAR or occurs in the tail of the NEAR. In the former situation, the corresponding subdirectory will be archived. In the latter situation, the corresponding subdirectory will be skipped. However, it is impossible that the subtree occurs in the head of a rule as well as in the tail of another rule. Suppose s_x is the current subtree and there is a PEP $\langle s_1 \cdots s_{x-1} \rangle$ along the path arriving at s_x. According to the algorithm of NEAR-$Miner$, descendant subtrees of s_{x-1} should be examined by extending the PEP $\langle s_1 \cdots s_{x-1} \rangle$. If s_x occurs in the head of a rule, $FoC(s_1 \cdots s_{x-1} s_x) \geq \beta$. If s_x occur in the tail of a rule, then $FoC(s_1 \cdots s_{x-1} s_x) < \beta$. Hence, a subtree does not occur both in the head of a rule and in the tail of another rule. That is, at each node, no contradictable NEARs that can be used. □

6. PERFORMANCE EVALUATION

In this section, we study the performance of our site-level web archive maintenance approach. Our proposed algorithms are implemented in the Java programming language. All experiments are conducted on a Pentium IV 2GHz PC with 2 GB memory. The operating system is Windows XP professional. We first present some empirical experimental results on several real-life Web sites with different features of Web directory evolution. Then, we evaluate our archive maintenance approach from different aspects based on a large set of real Web sites. Finally, we report the performance of the NEAR mining algorithm. Note that we do not compare our approach with existing techniques for efficient crawling [8, 9, 10, 13, 17, 20, 21] as these techniques are page-level strategies and orthogonal to our problem. We are not aware of any publicly-available site-level Web archive maintenance techniques.

6.1 Empirical Examples

Dataset. A. Ntoulas et al. [19] collected Web pages of 154 "popular" Web sites every week from October 2002 until October 2003, for a total of 51 weeks. In their paper, they explained how the sites are selected so that it is a "representative" yet "interesting" sample of the Web. We first conduct experiments on some example sites, where Web directories exhibit different evolutionary features. Basically, the group of example sites, shown in Figure 4, are different in the evolution of their directory tree size. For each site, Figure 4 shows the relative weekly-size, which is normalized by the size of its first version. We observed that the evolutions of the two sites *www.sec.gov* and *www.ksc.nasa.gov* exhibit certain patterns, e.g., generally, the size of both sites increase gradually. On the other hand, the evolutions of the two sites *www.fags.org* and *www.oracle.com* are relatively irregular. In addition, the size of the site *www.biosis.org* does not change frequently or significantly. We aim to study the effectiveness of our approach in maintaining archive of sites with different evolutionary characteristics.

Methodology and Metrics. For each Web site, we have total 51 historical versions of Web directory trees. We conduct experiments by using k successive versions as the training data to mine NEARs. With discovered rules, we test the performance of our approach with respect to a baseline approach, which archives the complete subsequent $(k+1)th$ version. Basically, we follow the algorithm WARM in Algorithm 4. However, instead of really archiving a page or

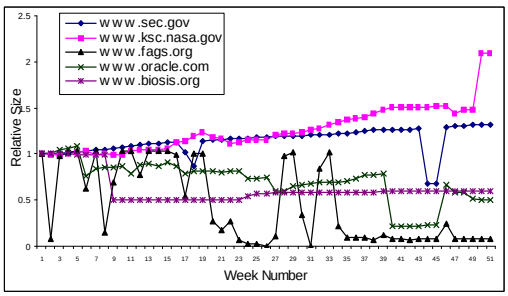


Figure 4: Size evolution of example sites.

skipping a subdirectory, we count the number of archived pages, the number of skipped pages as well as the number of changed pages (both archived and skipped). Let $N(V_t)$ and $C(V_t)$ be the set of pages and the set of changed pages in the test version respectively, $N(V_m)$ and $C(V_m)$ be the set of pages in the archived version and the set of changed pages updated by our approach respectively, and $S(V_m)$ be the set of pages skipped by our approach. The following four metrics are used to measure the performance:

$$Bypass\ Ratio(BR) = \frac{|S(V_m)|}{|N(V_t)|} \quad (2)$$

$$Overall\ Precision(OP) = \frac{|N(V_t) \cap N(V_m)|}{|N(V_m)|} \quad (3)$$

$$Overall\ Recall\ (OR) = \frac{|N(V_t) \cap N(V_m)|}{|N(V_t)|} \quad (4)$$

$$Change\ Recall(CR) = \frac{|C(V_t) \cap C(V_m)|}{|C(V_t)|} \quad (5)$$

The *Bypass Ratio* evaluates the efficiency gain of our approach. The *Overall Precision* and *Overall Recall* measures the accuracy of our approach with respect to the baseline approach, which achieves 100% precision and recall. The *Change Recall* measures the accuracy of our approach with respect to the real changes to the test version. Note that since all changed pages updated by our approach belong to the real changes to the test version, the change precision of our approach is always one. In a good approximate archive maintenance approach, values of all the four metrics should be as high as possible.

These metrics are computed as follows by WARM. In WARM, we maintain five counters: N_s is the number of skipped pages, N_m is the maintained pages, C_m is the maintained changed pages, C_t is the total number of changed pages (including insertion, deletion and modification), $C1_s$ is the number of skipped insertions and $C2_s$ is the number of skipped deletions. Then,

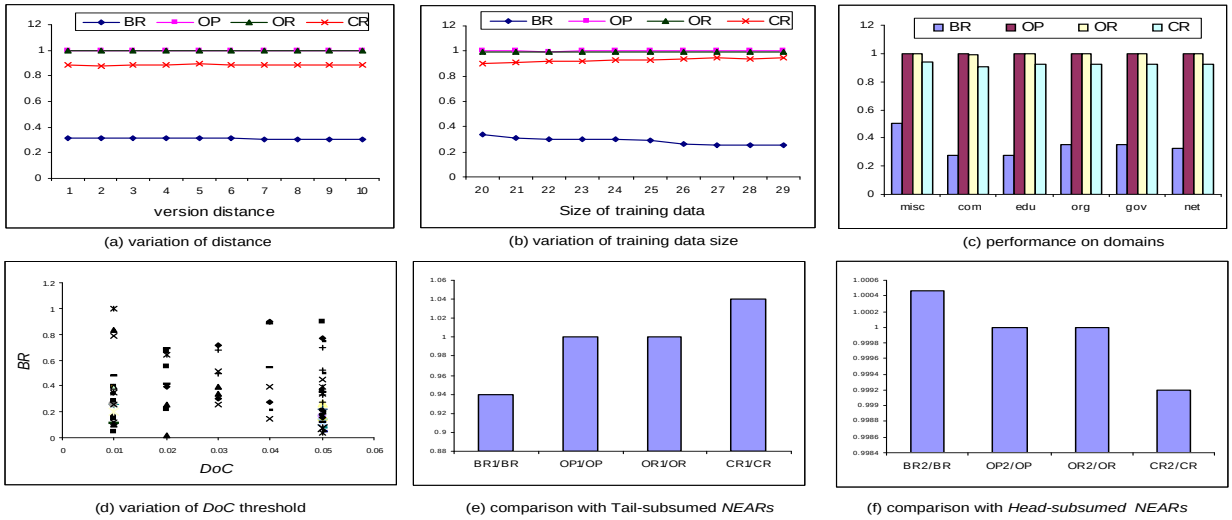
$$BR = \frac{N_s}{N_s + N_m} \quad (6)$$

$$OP = \frac{N_s + N_m - C1_s}{N_s + N_m + C2_s} \quad (7)$$

$$OR = \frac{N_s + N_m - C1_s}{N_s + N_m} \quad (8)$$

$$CR = \frac{C_t - C1_s - C2_s}{C_t} \quad (9)$$

Results. Table 2 shows the results of the five example sites by using 20 successive versions as the training data and the subsequent version as the test data. Note that we inten-



(a) variation of distance (b) variation of training data size (c) performance on domains

(d) variation of *DoC* threshold (e) comparison with Tail-subsumed *NEARs* (f) comparison with *Head-subsumed NEARs*

Figure 5: Performance of archive maintenance.

Site	*BR*	*OP*	*OR*	*CR*
www.sec.gov	0.62	0.99	0.99	0.87
www.ksc.nasa.gov	0.61	0.99	0.99	0.85
www.fags.org	0.30	0.99	0.90	0.81
www.oracle.com	0.40	0.99	0.97	0.79
www.biosis.org	0.85	0.99	0.99	0.65

Table 2: Empirical results.

tionally chose the the training data to be relatively "small" as effective rules should be discovered "quickly" from smaller number of versions (smaller datasets) so that they can be quickly deployed for archive maintenance. The results are averaged by running the experiment for 10 times on sequential 20 versions. From the results, we have the following observations. First of all, with respect to the size of the whole test version, our archive maintenance approach sacrifices only slight freshness. For example, both the *Overall Precision* and the *Overall Recall* are high for all sites. Secondly, our approach gains more efficiency improvement on sites with certain patterns in their evolution than sites with irregular evolution. For example, the *Bypass Ratio* values of the sites *www.sec.gov* and *www.ksc.nasa.gov* are higher than those of the sites *www.fags.org* and *www.oracle.com*. Similarly, the *Change Recall* value of the former two sites are higher than those of the latter two sites. Thirdly, for a site which does not change frequently, i.e., *www.biosis.org*, our approach achieves a high *BR* value but a low *CR* value because it is difficult to predict the location of changes for a site which rarely changes.

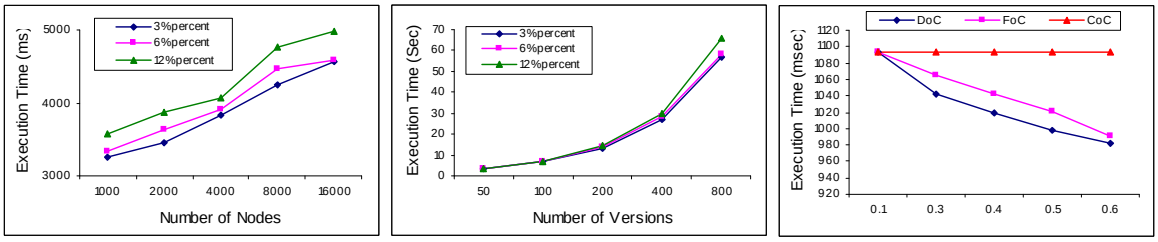
6.2 Performance of Archive Maintenance

Dataset. In this experiment, we evaluate the performance of our archive maintenance approach from different aspects based on the whole set of Web sites collected by A. Ntoulas et al. However, we noticed that data of some Web sites are incomplete. For example, the Web server accommodating the site *www.interplay.com* blocked Web crawling. Thus, we perform a preprocess on the set the of Web sites collected by A. Ntoulas et al. to filter those sites with more than 5 blocked historical versions. The list of remaining 102 sites used in our study is given in [7].

Methodology and Metrics. The following experiments are conducted on the set of Web sites. (a) We fix the size of the training data and vary the *distance* between training data and test data. For example, NEARs are mined from the first k versions and performance is evaluated with respect to the $(k+1)th$ version, the $(k+2)th$ version, and so on. This experiment aims to study the temporal validity of discovered rules. (b) We vary the size of the training data and use the subsequent version as the test data to study how the training size affects the performance. (c) We evaluate the effectiveness of our approach on sites from different domains, such as *.com*, *.gov*, *.org* etc. (d) We study how the thresholds (e.g., α, β, θ) affect the performance of our approach. (e) We evaluate the performance of archive maintenance strategies based on NEARs which are subsumed by others. The same four metrics, *Bypass Ratio*, *Overall Precision*, *Overall Recall* and *Change Recall* are used to evaluate the performance.

Results. Figure 5(a) shows the results of the first experiment which varies the distance between the training data and the test data. Similarly, the results corresponding to each distance are averaged by running experiments 10 times on sequential data. It can be observed that our approach *skips around one third of the pages of a site* on average and loses slight "freshness", e.g., values of *OP*, *OR* and *CR* are high (Note that the results are obtained by using the same thresholds on all sites. As we show later, the performance on each site can be tuned by setting appropriate thresholds). Furthermore, the performance of our approach does not deteriorate obviously with the increase of the distance. This may be because that discovered evolution associations hold for most of sites in the tested period. Consequently, we *do not need to incrementally maintain* NEAR for every version. The discovered rules are stable enough to be used for several subsequent versions without incurring the cost of maintaining them incrementally.

Figure 5(b) shows the results of the experiment which varies the size of the training data. We noticed that when the size of the training data increases, the *BR* value of our approach slightly decreases. It is because we fix the thresholds for training data of different size. As a result, when the size of training data increases, different sets of rules, based on which fewer pages are skipped, are discovered. Accordingly, fewer changed pages are missed by our approach,



(a) Variation on tree size (b) Variation on versions (c) Variation on metric thresholds

Figure 6: Performance of NEAR-*Miner*.

which results in the improvement of the CR value.

In Figure 5(c), we present the performance of our approach with respect to sites from different domains. According to the results, it indicates that our approach works better on sites from domains like *.org* and *.gov* than on sites from domains such as *.com* and *.edu*. The reason may be that Web sites from the latter two domains evolves more irregularly.

Figure 5(d) shows the resulting BR value of sites on variation of the DoC threshold. Each point in the figure corresponds to a site which achieves highest BR value at current threshold (relatively highest in the DoC threshold range of $[0.01, 0.05]$). It indicates that, similar to traditional association rule mining, appropriate thresholds are site-dependent.

Figure 5(e) shows the performance of archive maintenance based on NEARs subsumed by others NEARs. We normalize the results (e.g., $BR1$) with respect to the performance of our approach (e.g., BR). It can be observed that maintenance strategies based on subsumed rules gain less efficiency than our approach does (e.g., $BR1/BR < 1$). Correspondingly, it sacrifices less freshness (e.g., $CR1/CR > 1$). However, the magnitude of the former is larger than the latter. Similarly, Figure 5(f) shows the normalized performance of maintenance strategies based on NEARs subsumed by other PEPs. Compared to our approach, such strategies skip more pages and miss more changed pages. However, the magnitude of the efficiency they improve is less than that of the freshness they lose. Therefore, as an approximate archive maintenance strategy which aims to achieve high efficiency as well as sacrifice slight freshness, our approach works better than approaches based on subsumed rules.

6.3 Performance of NEAR-Miner

We also evaluate the performance of our algorithm for NEAR mining on synthetically generated sequential tree versions. Basically, we generate the first tree version with respect to three parameters, tree *size* (N), tree *fanout* (F) and tree *depth* (D), and generate subsequent versions based on the parameters of version numbers (V) and the change percentage C. We conduct experiments to evaluate the scalability as well as the efficiency of our algorithm. The experimental results are shown in Figure 6. Figure 6(a) shows the scale-up feature with respect to the increase of the (first) tree size (N) on three datasets which are generated with different value of C. Figure 6(b) presents the scalability with respect to the increase of version numbers (V) on three datasets generated with three C values also. Figure 6(c) presents the execution time of the second phase of our algorithm by varying the thresholds (the variation of the thresholds do not affect the complexity of the first phase). In all experiments, when the threshold of a metric is varied, the other two thresholds are fixed at 0.1. From the results, we noticed that the efficiency decreases when the thresholds of DoC and FoC increase. While the threshold of CoC does not affect the efficiency of the NEAR-*Miner* algorithm.

7. RELATED WORK

Negative association rule mining. Although traditional association rule mining focuses on mining positive associations between items, it was observed that negative associations were useful as well in some situations [27]. Consequently, several work have focused on mining negative association rules [1, 3, 27]. They are different from each other in the employed correlation metrics. As a result, the developed data mining algorithms are different as well. For example, Wu et al. [27] added on top of the *support-confidence* framework another measure called *mininterest*. They developed level-wise algorithms to find interesting itemset pairs first and then derive negative association rules. Antonie and Zaïane [1] used the *correlation coefficient* and integrated two phases to mine patterns and derive rules together. However, none of the above efforts mine negative associations from changes to tree structures. Also, our proposed rules differ in the way that the Web directory trees have ancestor-descendant relationships.

Mining evolution of trees. In our previous work [6], we proposed a novel approach for mining structural evolution of XML data to discover FRACTURE patterns. A FRACTURE pattern is a set of subtrees that frequently changes together and frequently undergoes significant changes together. Our work differs from FRACTURE mining in the following aspects. Firstly, different evolution metrics are used. Secondly, the NEARs are defined on a sequence of subtrees with ancestor-descendant relationships, while the FRACTUREs are defined on subtrees which may or may not be structurally related. Lastly, unlike FRACTUREs, subsumption relationships are taken into account in NEAR mining. Recently in [5], we proposed to discover NECTAR patterns where each pattern refers to subtrees negatively correlated in undergoing significant changes. Our work differs from NECTAR mining fundamentally because we focus on site-level Web archive maintenance, while [5] addresses a data mining problem defined on general trees. Consequently, the specialty of Web directory structure is exploited in our work to optimize the knowledge discovery. Moreover, NECTAR discovers only patterns of subtree sets. To develop effective web archive maintenance strategies, we discover association rules from subtrees which provide additional predictive power. Lastly, the key contribution of this paper is a novel web archive maintenance algorithm based on discovered NEARs. Such algorithm is not discussed in [5].

Web archiving and crawling. None of the existing

archiving techniques [16, 23] exploit evolutionary association to select pages for archiving. Since a crawler is a core component of any Web archiving system, we compare our work with existing Web crawling strategies.

Web crawling is a well-studied research problem. Majority of recent research in this area focus on improving efficiency and scalability of the crawler. Pant et al. [21] describes the technical process involved in Web crawling. Cho et al. [9] introduced the importance of crawler efficiency and present an algorithm that retrieves most "relevant" pages first. Subsequently, Cho and Garcia-Molina developed an effective *incremental* crawler [8]. An incremental crawler does not visit the entire Web each time it runs. Rather, the crawler visits only those pages that it believes to have changed or been created since the last run. They also studied refresh policies for Web pages in [10]. In another study in [11], they develop several change frequency estimators in order to improve Web crawlers and Web caches. Edward et al. [13] also propose a change frequency-based adaptive model to optimize performance of incremental crawlers. Olston and Pandey [22] present a strategy to schedule Web pages for selective (re)downloading into a search engine repository. This strategy aims to maximize the quality of the user experience for those who query the search engine. More recently, they characterize *longevity* of information in Web pages and then develop new and improved recrawl scheduling policies that incorporate information longevity into account [20].

In comparison to the above approaches, our proposed technique of Web archive maintenance differs in the following ways. Firstly, our strategy for selecting pages to archive is at the level of Web directories whereas the above techniques are defined at much finer granularity (page level). Hence, our approach complements these efforts. Secondly, to the best of our knowledge, none of these crawling techniques exploit negatively correlated subtree structures to improve crawling efficiency.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel optimized web archive maintenance approach for autonomous Web sites. In our approach, we first present an algorithm NEAR-*Miner* to mine the evolution history of Web directories to extract rules related to negative evolution correlations (referred to as NEARs) between Web directories with ancestor-descendant relationships. In the next step, we designed an efficient archive maintenance approach based on discovered rules. Our maintenance strategies optimally skip the subdirectories (in the original site) which are negatively correlated with it in undergoing significant changes. We conducted extensive experiments on real-life datasets to evaluate the performance of our archive maintenance algorithm called WARM. The promising results show that the proposed maintenance strategies significantly improve the efficiency of maintenance process without sacrificing too much "freshness" of the archives. We believe that our proposed approach can also be used in applications beyond archiving (e.g., maintaining local warehouse of remote data sources).

As part of future work, we are interested in combining our site-level maintenance strategies with page-level strategies. For example, our current solution considers a leaf node as either updated or not. To take into account the degree of update, we may need to revise the definition of DoC so that it considers not only changes to structures but also changes to content. The combined strategies can be expected to be more effective by skipping pages with minor changes (e.g., near-duplicate Web documents [15]) and subdirectories with many slightly updated pages.

9. ACKNOWLEDGEMENT

This work is partly funded by the School of Computer Engg., NTU, the European Commission under Pharos (IST 045035), and LiWA (IST 216267).

10. REFERENCES

[1] M. Antonie and O. R. Zaiane. Mining Positive and Negative Association Rules: An Approach for Confined Rules. *In PKDD*, 2004.

[2] R. Bayer. Information Life Cycle, Information Value, and Data Management. *In DEXA* , 2007.

[3] S. Brin, R. Motwani, and C. Silverstein. Beyond Market Baskets: Generalizing Association Rules to Correlations. *In SIGMOD*, 1997.

[4] S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data. *In SIGMOD*, 1997.

[5] L. Chen and S. S. Bhowmick. In the Search of NECTARs from Evolutionary Trees. *In DASFAA*, 2009.

[6] L. Chen, S. S. Bhowmick, and L. T. Chia. FRACTURE Mining: Mining Frequently and Concurrently Mutating Structures from Historical XML Documents. In *Data and Knowl. Eng.*, volume 59, pages 320–347, 2006.

[7] L. Chen, S. S. Bhowmick, and W. Nejdl. Autonomous Web Archive Maintenance Based on Evolution Associations of Web Site Directories. Techical report available at `http://www.cais.ntu.edu.sg/~assourav/TechReports/NEAR-TR.pdf`

[8] J. Cho and H. Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler. *In VLDB*, 2000.

[9] J. Cho, H. Garcia-Molina, and L. Page. Efficient Crawling Through URL Ordering. *In WWW*, 1998.

[10] J. Cho and H. Garcia-Molina. Effective Page Refresh Policies for Web Crawlers. *In ACM TODS*, 28(4), 2003.

[11] J. Cho and H. Garcia-Molina. Estimating Frequency of Change. *In ACM TOIT*, 3(3), 2003.

[12] J. Cohen. Statistical Power Analysis for the Behavioral Sciences. *Lawrence Erlbaum Associates*, 1988.

[13] J. Edwards, K. S. McCurley, and J. A. Tomlin. An Adaptive Model for Optimizing Performance of an Incremental Web Crawler. *In WWW*, 2001.

[14] Y. Fu, K. Sandhu, and M. Shih. A Generalization-based Approach to Clustering Web Usage Sessions. *In WebKDD*, 1999.

[15] M. R. Henzinger. Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms. *In SIGIR*, 2006.

[16] J. Masanès. Web Archiving. Springer, New York, Inc., Secaucus, N.J., 2006.

[17] F. Menczer, G. Pant, P. Srinivasan, M. E. Ruiz. Evaluating Topic-Driven Web Crawlers. *In SIGIR*, 2001.

[18] O. Nasraoui, M. Soliman, E. Saka, A. Badia, and R. Germain. A Web Usage Mining Framework for Mining Evolving User Profiles in Dynamic Web Sites. *IEEE TKDE*, 20(2), 2008.

[19] A. Ntoulas, J. Cho and C. Olston. What's new on the Web?: The Evolution of the Webfrom a Search Engine Perspective. *In WWW*, 2004.

[20] C. Olston and S. Pandey. Recrawl Scheduling based on Information Longevity. *In WWW*, 2008.

[21] G. Pant, P. Srinivasan, and F. Menczer. Crawling the Web. *Web Dynamics*, 2004.

[22] S. Pandey, and C. Olston. User-Centric Web Crawling. *In WWW*, 2005.

[23] M. Spaniol, D. Denev, A. Maxeika, et al. Data Quality in Web Archiving. *In ACM WICOW*, 2009.

[24] P.-N. Tan, M. Steinbach, and V. Kumar. Introduction to Data Mining. *Addison Wesley*, 2006.

[25] G. Valiente. Algorithms on Trees and Graphs. *Springer-Verlag*, 2002.

[26] Y. Wang, D. J. DeWitt, and J.-Y. Cai. XDiff: An Effective Change Detection Algorithm for XML Documents. *In ICDE*, 2003.

[27] X. Wu, C. Zhang, and S. Zhang. Mining both Positive and Negative Association Rules. *In ICML*, 2002.