Detecting and representing relevant Web deltas in WHOWEDA

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

Wee Keong Ng

Sourav S. Bhowmick

Follow this and additional works at: http://scholarsmine.mst.edu/faculty_work

Part of the Computer Sciences Commons

Recommended Citation
Madria, Sanjay Kumar; Ng, Wee Keong; and Bhowmick, Sourav S., "Detecting and representing relevant Web deltas in WHOWEDA" (2003). Faculty Research & Creative Works. Paper 791.
http://scholarsmine.mst.edu/faculty_work/791

This Article is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. For more information, please contact weaverjr@mst.edu.
Detecting and Representing Relevant Web Deltas in WHOWEDA

Sourav S. Bhowmick, Member, IEEE Computer Society, Sanjay Kumar Madria, Senior Member, IEEE, and Wee Keong Ng, Member, IEEE Computer Society

Abstract—In this paper, we present a mechanism for detecting and representing changes, given the old and new versions of a set of interlinked Web documents, retrieved in response to a user’s query. In particular, we show how to detect and represent Web deltas, i.e., changes in the Web documents that are relevant to a user’s query in the context of our Web warehousing system called WHOWEDA (Warehouse of Web Data). In WHOWEDA, Web information is materialized views stored in Web tables in the form of Web tuples. These Web tuples, represented as directed graphs, can be manipulated using a set of Web algebraic operators. In this paper, we present a mechanism to detect relevant Web deltas using Web algebraic operators such as the Web join and the outer Web join. We will use these to identify dangling Web tuples. We show how to represent these changes using delta Web tables. We develop formal algorithms for the generation of delta Web tables identifying Web documents which have been added, deleted, or modified since the last query.

Index Terms—Web deltas, Web warehouse, Web join, outer Web join, delta Web tables, algorithm.

1 INTRODUCTION

The Web offers access to large amounts of heterogeneous information and allows this information to change at any time and in any way. These rapid and often unpredictable changes to the information create a new problem of detecting and representing changes. This is a challenging problem because the information sources in the Web are autonomous and typical database approaches to detect changes based on triggering mechanisms are not usable. Moreover, these sources typically do not keep track of historical information in a format that is accessible to the outside user [10].

Recently, there has been increased research interest in detecting changes in structured and semistructured data [13], [14], [15]. In this paper, we present a mechanism for detecting and representing changes in Web documents (hence, referred to as Web deltas) which are relevant to a user’s query using two Web algebraic operators, i.e., Web join and outer Web join, in the context of our Web warehousing system called WHOWEDA (Warehouse Of Web Data) [4], [7]. Such a mechanism for detection and representation of Web deltas may be used by the following types of Web users:

1. Web site administrators. By scanning the changes, administrators will be sure whether the changes are consistent with any policies for content or format without having to review the entire set of pages at the same level of detail.

1. A shorter version of this paper appeared in [3].

2. Customers of E-commerce Web sites. A user may wish to monitor new products, services, or auction on E-commerce Web sites.

3. Analysts for gathering competitive intelligence. Companies can monitor evolution of their competitors’ Web sites to discover their new directions or offerings over a period of time that may influence their market positions.

4. Developers of Web mining applications. By detecting and representing Web deltas over a broad vista of time, our system can be used as the foundation for mining information related to trends, patterns, etc.

5. Wireless users. The ability to download or highlight only changes instead of a complete Web page can be a very desirable feature for wireless users using handheld devices.

1.1 Motivating Example

We illustrate, with an example, some of the changes that may take place in the Web. We will use this as a running example in the rest of this paper.

Assume that there is a Web site at http://www.panacea.gov/ which provides information related to drugs used for various diseases. For instance, the structure of the site as on 15 January, 2001 is shown in Fig. 1a. We can see that the Web page at http://www.panacea.gov/ (denoted by $0$) contains a list of diseases. From this list, each link of a particular disease points to a Web page (denoted by $b_0, b_1, b_2$, etc., for various drugs) containing a list of drugs used for prevention of the disease. For example, the link labeled “AIDS” in the Web page at http://www.panacea.gov/ points to a document (denoted by $b_0$) containing the list of drugs (i.e., “Indavir,” “Ritonavir,” etc.) used against AIDS. From the hyperlinks associated with each drug, one can probe further to find documents (denoted by $u_0, u_1$, etc.) containing a list of various issues related to a particular drug, i.e., “description,”...

---

S. S Bhowmick and W. K. Ng are with Nanyang Technological University, School of Computer Engineering, Blk N4, #2A-32, Nanyang Avenue, Singapore 639798. E-mail: {assourav, awkng}@ntu.edu.sg.

S. Madria is with the Department of Computer Science, University of Missouri-Rolla, Rolla, MO 65409. E-mail: madrias@umr.edu.

Manuscript received 7 Apr. 2000; revised 10 Apr. 2002; accepted 7 June 2002.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 111872.
“manufacturers,” “clinical pharmacology,” “uses,” “side-effects,” “warnings,” etc. From the hyperlinks associated with each issue, one can retrieve details of these issues for a particular drug. Note that, in Fig. 1, we only show the links related to “uses” and “side-effects” of drugs to simplify visualization.

Let us consider some modifications to this Web site on 15 February, 2001 as shown in Fig. 1b. The black boxes, the grey boxes, and the boxes with thick boundaries in this figure (and all the figures in this paper except Fig. 4), depict the addition of new documents, the deletion of existing documents, and the modification of existing documents, respectively. Furthermore, the dashed dotted arrows indicate the addition, deletion, or modification of hyperlinks. Observe the modification of the link structure of “Impotence.” Previously, the information related to “Vasomax,” a drug used against “Impotence,” was provided by the Web site at http://www.pfizer.com/ (Web pages u₄, u₅, u₆, d₄, and k₄ in Fig. 1a belong to the Web site at http://www.pfizer.com/). That is, the link labeled as “Vasomax” in b₄ in Fig. 1a, was a global link. Now, this information is provided locally by http://www.panacea.gov/ and the structure of interlinked documents is modified as shown in Fig. 1b (documents u₉, d₈, and k₈).

Suppose on 15 January, 2001, a user wishes to find out, periodically (say, every 30 days), information related to side effects and uses of drugs for various diseases and also changes to this information compared to its previous version. This query requires access to previous states of the Web site and a mechanism to detect these changes automatically, features that are not supported by the Web or the existing search engines. Thus, we need a mechanism to compute and represent changes in the context of Web data.

1.2 Overview

The work on change detection and representation reported in this paper, has four key characteristics:

- **Relevant Web deltas.** We focus on detecting relevant Web deltas. In particular, our goal is to detect and represent Web deltas that are relevant to a user’s query, not any arbitrary Web deltas.
- **Changes in interlinked Web documents.** Our focus is on detecting and representing relevant changes, given old and new versions of a set of interlinked Web documents. In particular, we are interested in detecting those Web documents in a Web site which are added to or deleted from the site, or those documents which are no longer considered relevant to a user’s query. We also want to identify a set of documents which have undergone content modification compared to their antecedent. Furthermore, we wish to determine how these modified Web documents are related to one another and with other relevant Web documents in the context of a user’s query.
- **Web algebraic operators.** We present a mechanism for detecting and representing relevant Web deltas using a set of Web algebraic operators. These operators are applied on a sequence of Web data snapshots to infer changes.
Static Web pages. Web documents that do not provide the last modification date, such as the output from Common Gateway Interface (CGI) scripts, are not considered in this paper for change detection.

Our goal is to detect and represent changes in Web data using a set of Web algebraic operators in the context of WHOWEDA, a data warehousing system for managing and manipulating relevant data extracted from the Web [4]. Informally, our Web warehouse is conceived of as a collection of Web tables. A set of Web tuples and a set of Web schemas [6] is called a Web table. A Web tuple is a directed graph consisting of a set of nodes and a set of links and satisfies a Web schema. Nodes and links contain content, metadata, and structural information associated with the Web documents and hyperlinks among the Web documents. To facilitate manipulation of Web data stored in the Web tables, we have defined a set of Web algebraic operators (i.e., global Web coupling, Web join, Web select, etc.) [3], [4], [7], [8], [22].

Specifically, a Web join operator is used to combine identical data residing in two Web tables. In Web join, the Web tuples from two Web tables (say \( W_1 \) and \( W_2 \)) containing joinable nodes (nodes participating in Web join operation) are concatenated into a single joined Web tuple that can be materialized in a Web table. A pair of nodes are joinable if they are identical in content. We consider two nodes or Web documents identical when they have the same URL and last modification date. Observe that, based on this definition of identity of Web documents, same documents stored in mirror sites having different URL are not considered identical. The Web tuples \( w_b \in W_1 \) and \( w_b \in W_2 \) are concatenated over the joinable nodes to create a joined Web tuple. The joined Web table contains such a set of joined Web tuples.

The Web tuples in \( W_1 \) and \( W_2 \) that do not participate in the Web join operation (dangling Web tuples) are absent from the joined Web table. The outer Web join operator, a derivative of Web join, identifies these dangling Web tuples in \( W_1 \) and \( W_2 \). We define two flavors of outer Web join, i.e., left and right outer Web join, to identify the dangling Web tuples from \( W_1 \) and \( W_2 \), respectively.

As Web data in our Web warehouse are materialized views stored in the form of Web tables, any changes to the relevant Web data are also reflected in the corresponding Web tables. Consequently, in order to detect Web deltas, we materialize the old and new versions of data in two Web tables. Next, we create a set of Web tables by manipulating these input Web tables using the Web join and outer Web join operators. Finally, we create a set of delta Web tables by further manipulating the joined and outer joined Web tables. Delta Web tables encapsulate the changes that have occurred in the Web such as addition, modification, or deletion of a set of Web documents in the context of a user’s query.

2 RELATED WORK

In recent years, several tools have become available to address the problem of determining when an HTML page has changed. URL-minder [1] runs as a service on the Web itself and sends an email when a page changes. However, the need to send URLs explicitly through a form is cumbersome and may not be a feasible option when there is a large number of URLs to track.

The AT&T Internet Difference Engine (AIDE) [16] is a system that finds and displays changes to pages on the World Wide Web. A tool, called HtmlDiff, highlights changes between versions of a page and a graphical interface to view the relationship between pages over time. HtmlDiff automatically compares two HTML pages and creates a "merged" page to show the differences with special HTML markups. TopBlend [11] is an HTML differencing tool implemented in Java and significantly outperforms the old HtmlDiff in the most time consuming jobs.

AIDE also supports recursive tracking and differencing of a page and its descendants. When recursion is specified, changes to the child pages are reported separately by default. A user may specify a number of operations in AIDE which includes registering a URL including the degree of recursion through links to other pages, view textual differences between a pair of versions, and view a graph showing the structure of a page, etc.

WebGUIDE (Web Graphical User Interface to a Difference Engine) [17] is another tool that supports recursive document comparison: users may explore the differences between the pages with respect to two dates. Differences between pages are computed automatically and summarized in a new HTML page, and the differences in link structure are shown via graphical representations. WebGUIDE is a combination of two tools, Ciao [12] and the AIDE. With Ciao, the high-level structural differences are displayed as graphs that show the relationships between pages using colored nodes to indicate which pages have been modified. Using AIDE, the low-level textual differences are illustrated by marking changes between versions and modifying anchors to cause documents reached from that page to be annotated. WebGUIDE allows a user to issue queries for specific types of deltas.

The AIDE and WebGUIDE have certain limitations. First, we believe that specifying a set of URLs to track changes may not be feasible when there is a large number of URLs. Second, the recursion specification is restrictive. That is, it selects all of the children of a specified document(s), however, in reality, a user may often be interested in only some of those links. On top of that, the user may wish to track changes of successive interlinked documents, satisfying some hyperlinked structure. Such constraints cannot be specified in AIDE. Third, AIDE displays all of the changes in the documents. In reality, we may not be interested in all of the changes, but only some of these changes. Hence, it is necessary to be able to query these changes rather than browsing them to find the relevant changes. This is extremely useful when the number of documents monitored is large.

In [25], the authors define a change detection problem for ordered trees, using insertion, deletion, and label-update as the edit operations. In [14], the authors discuss a variant of change detection problem for ordered trees using subtree moves as an edit operation in addition to
insertions, deletions, and updates, and presented an efficient algorithm for solving it. They focus on the problem of detecting changes, given the old and new versions of hierarchically structured data. Change detection problem for unordered trees is presented in [13]. The authors present efficient algorithms to detect changes in operations that moves an entire subtree of nodes and that copies an entire subtree. More recently, in [10], a snapshot-delta approach has been used for representing changes in semistructured data. The authors present a simple and general model called DOEM for representing changes and also present a language called Chorel for querying changes represented in DOEM. This model is founded on the OEM data model and the Lorel query language [2]. It uses annotations on the nodes and arcs of an OEM graph to represent changes. Intuitively, the set of annotations on a node or arc represents the history of that node or arc. An important feature of this approach is that it represents and queries changes directly as annotation on the affected area instead of representing them indirectly as the difference between database states. Furthermore, they describe the design and implementation of an application of change management called a Query Subscription Service (QSS). QSS can be used to notify subscribers of relevant changes in semistructured information sources.

DOEM was not specifically developed for the Web and the model does not distinguish between graph edges that represent the connection between a document and one of its parts, and edges that represent a hyperlink from one Web document to another. We take a different approach as compared to [10], [14]. Our approach is specifically developed for finding changes to Web data. Rather than finding changes to the internal structure and content of Web documents, in this paper, we focus on identifying changes to a set of hyperlinked Web documents relevant to a user’s query. Our approach can easily be extended to detect and represent changes to internal structure and content of Web documents.

WebCQ system [20] is a prototype system for Web information monitoring and delivery. It provides a personalized notification of what and how Web pages of interest have been changed, and personalized summarization of Web page changes. Users’ update monitoring requests are modelled as continual queries [21] on the Web. WebCQ has the same limitations as of AIDE described earlier.

Recently, the XML research community has recognized the importance of change management problem and, in [15], the authors have discussed change management in the context of XML data. Their approach of change management is based on tree comparison. However, they do not address the problem of detecting changes to hyperlinked XML documents. Moreover, we only find changes that affect user’s query responses. However, in their approach, they find changes between any two given versions of XML data. Our approach can be extended for detecting and representing changes in XML data.

Finally, our approach is different from graph matching and isomorphic graph problems. A matching in a graph is a set of edges such that every vertex of the graph is on, at most, one edge in the set. Two graphs are isomorphic if one can label both graphs with the same labels so that every vertex has exactly the same neighbors in both graphs. Some of the related research on graph matching problems are [24], [27], [28]. In our approach, two nonisomorphic graphs may also join if some of the nodes are identical. Thus, we focus more on the node content similarity rather than the structure of the graph.

3 Preliminaries

In this section, we provide the framework for our subsequent discussion on change detection.

3.1 WHOM—The Data Model of Whoweda

The Warehouse Object Model (WHOM) [4] serves as the basic data model for our Web warehousing system. Informally, our Web warehouse can be conceived of as a collection of Web tables. A Web table is a 3-tuple \(W = (Z, S, T)\), where \(Z\) is the Web table name, \(S\) is a set of Web schemas [6], and \(T\) is a set of Web tuples satisfying \(S\). A Web tuple is a directed graph consisting of sets of node and link objects (hereafter, referred to as nodes and links, respectively, for brevity). Fig. 2 and Fig. 3 are examples of Web tuples in two Web tables. *Web schemas* are used to bind a set of Web tuples in a Web table and define the structure of a set of data in the warehouse. The reader may refer to [6] for details on Web schemas.

Intuitively, a *node* represents the metadata associated with a Web document and the content and structure of the document (excluding hyperlinks in the document). Specifically, it consists of two components: different metadata associated with the document (such as source URL, date, and size etc.) and a directed labeled tree to represent the content and structure of the document. Similarly, a *link* consists of a set of link metaattribute/value pairs (such as target URL, source URL, and link type [23]) and a link data tree. Link data tree is a directed labeled tree to represent the structure and content of an HTML or XML link. In Fig. 2 and Fig. 3, the boxes represent node objects and the arrow between two nodes represents the link object. The reader may refer to [4] for complete discussion on Web data representation.

3.2 Global Web Coupling

Global Web coupling [4], [8], [22] enables a user to retrieve a set of interlinked documents satisfying a Web query, regardless of the locations of the documents in the Web. To initiate a global Web coupling, the user specifies a Web query in the form of a coupling query [5]. The global Web coupling operator \(\Gamma\), takes in a coupling query \(G\) and returns a Web table \(W = (Z, S, T)\) containing a set of Web tuples \(T\) extracted from the World Wide Web satisfying the query and a set of Web schemas \(S\) generated from \(G\) and \(T\). That is, \(W = \Gamma(G)\). Each Web tuple matches a portion of the World Wide Web satisfying the constraints described in \(G\). These constraints are imposed on the metadata, content, and structure of Web documents and hyperlinks. We have omitted discussion on the coupling query here for space constraints. We assume that

---

2. Note that, in all figures related to Web tables, the Web tuples are numbered for reference.

3. We only consider simple and extended XML links.
the sets of interlinked documents retrieved by the given coupling query are materialized in the Web tables Drugs and New Drugs, respectively, as shown in Fig. 2 and Fig. 3. Each Web tuple in these tables contains information about side effects and uses of a drug used for a particular disease. Observe that, in Fig. 1a, information related to the drug “Niacin” used for heart diseases (documents $u_3$ and $k_3$) is not materialized in Drugs, as it does not satisfy the coupling query. However, it is materialized in New Drugs, as it satisfies the coupling query on 15 February due to the addition of a document related to the side-effects of “Niacin” ($d_7$) to the Web site at http://www.panaace.gov/. Notice that the Web tuples related to Diabetes and Alzheimer’s disease are not materialized in New Drugs due to the removal of documents $b_3$, $d_4$, $k_5$, $d_{12}$, and $k_{12}$ from the Web site.

### 3.3 Storage of Web Objects

In this section, we briefly introduce various physical storage structures in WHOWEDA for storing Web objects, i.e., nodes, links, Web documents, Web tables, etc. We introduce three types of storage structures: warehouse node pool, warehouse document pool, and Web table pool for storing Web objects. The warehouse node pool contains distinct nodes from all Web tables in our Web warehouse. Each node represents a Web document stored in the warehouse document pool. The links in each Web document are stored in this pool along with the corresponding node. Furthermore, each node has an identifier called node id. Note that the node id of a node is different from that of another node if their URLs are different. A node may have several versions. In order to distinguish between several versions of a node, each version of a node is identified by a unique version id. Note that each node id can have a set of version ids across different Web tables. A node in our Web warehouse can be uniquely identified by the pair (node id, version id).

The Web tables are stored in the Web table pool. Each Web table in this pool is stored in three types of structures, i.e., the table node pool, the Web tuple pool, and the Web schema.
pool. For each distinct node and link object in a Web table, we store the following attributes in the table node pool: identifier that the node and the link represent in the Web schema, node id and link id, version id, and URL of the node, target node id, and label and link type of the link.

Next, we store the Web tuples of a Web table in the Web tuple pool. For each tuple in this pool, we only store the ids of all of the nodes and links belonging to that tuple. Finally, we store the Web schemas and coupling query in the Web schema pool. The reader may refer to [29], [30] for detailed exposition on these storage structures.

## 4 Change Detection Problem

In this section, we first describe the change detection problem. Then, we identify the basic change operations in WHOWEDA corresponding to the changes in the Web. Finally, we show how to represent these changes in the context of our Web warehouse.

### 4.1 Problem Definition

As changes in relevant Web data are reflected on the Web tables, we can address the problem of detecting and representing changes to Web data in the context of such Web tables. We first describe the problem, informally, using the example in Section 1.1. Recall from Section 1.1, a user wishes to find a list of drugs for various diseases, their side effects, and uses starting from the Web site at http://www.panacea.gov/. The user specifies a polling coupling query with polling times $t_1 = 15$ January, 2001, $t_2 = 15$ February, 2001. At polling time $t_1$, the global Web coupling operation retrieves a set of interlinked documents and materializes them in the form of a Web table called Drugs as depicted in Fig. 2. Before polling time $t_2$, the Web site at http://www.panacea.gov/ is modified as depicted in Section 1.1. Therefore, at $t_2$, the result New Drugs (Fig. 3) of the polling coupling query contains the relevant changes that have occurred between time $t_1$ and $t_2$. Given two such Web tables, Drugs and New Drugs, containing the snapshots of two versions of relevant Web data, the problem of change detection is to find the set of Web tuples containing nodes which are inserted into or deleted from Drugs or those nodes which are modified in Drugs to transform it into New Drugs. Note that these Web tuples will reflect the changes to the Web site that are relevant to the user.

### 4.2 Types of Changes

Changes to the Web tables are reflected on the individual Web tuples in WHOWEDA. Consequently, the different types of change operations in WHOWEDA can be defined in terms of the following:

- **Insert Node.** Intuitively, the operation Insert Node creates a set of nodes $N$ in a Web tuple in the Web table $W$. The nodes must be new, i.e., $N$ must not occur in $W$ before. Note that $N$ can be a new Web tuple added to $W$ or it can be a set of nodes inserted into an existing Web tuple in $W$.

- **Delete Node.** This operation is the inverse of the Insert Node operation. It removes a set of nodes from $W$.

- **Update Node.** The operation Update Node modifies the contents of the nodes in a Web tuple. By content modification of a node, we mean the textual contents or structure of the node may change or the attributes of the links embedded in the node may change.

- **Insert Link.** Intuitively, the operation Insert Link creates a set of links $L$ in a Web tuple in the Web table $W$. The links must be new, i.e., $L$ must not occur in the Web tuple before. Observe that, in a Web table, a new link can occur in two ways. First, a new link may connect an existing node to a new node. In this case, the Insert Link results in an Insert Node operation. Second, a new link may connect to existing nodes in a Web tuple. In this case, the Insert Link operation is equivalent to the Update Node operation as the source node of the link has to be modified in order to incorporate the new link. So, we can express the Insert Link operation by the Insert Node or Update Node operation.

- **Delete Link.** It removes a set of links from $W$. Similar to Insert Link, in a Web table, deletion of a link may occur based on two cases. First, removal of a link may remove the only link to an existing node. In this case, it is equivalent to the Delete Node operation. Second, a link may be deleted between two nodes which are connected by more than one link. In this case, this operation is essentially the Update Node operation. Therefore, we can express the Delete Link operation by the Delete Node or Update Node operation.

- **Update Link.** This operation involves modification of the anchor of a link. Hence, it is essentially an Update Node operation.

### 4.3 Representing Changes

We define a structure called the delta Web table for representing Web deltas. Delta Web tables encapsulate the relevant changes that have occurred in the Web with respect to a user’s query. We define the following three types of delta Web tables to represent the above types of change operations:

- **$\Delta'$-Web table** (denoted as $W_{\Delta'}$). It contains a set of tuples containing the new nodes inserted into $W_1$ for transforming it into $W_2$. Note that this Web table represents the Insert Node operation.

- **$\Delta$-Web table** (denoted as $W_{\Delta}$). It contains a set of tuples containing nodes deleted from $W_1$ as determined by the Delete Node operation which transforms $W_1$ to $W_2$. Note that the Web tuples in $W_{\Delta}$ do not necessarily indicate that these sets of interlinked Web documents are deleted from the Web site. These documents may still exist in the Web, however, they may no longer be relevant to the user’s query due to modification of the content or interlinked structure of these pages.

- **$\Delta^M$-Web table** (denoted as $W_{\Delta^M}$). It contains a set of Web tuples that represent the previous and current
sets of nodes modified by the Update Node operation.

Definition 1 [Delta Web Tables]. Let $W_1 = (Z_1, S_1, T_1)$ and $W_2 = (Z_2, S_2, T_2)$ be two Web tables generated by a polling coupling query $G$ at time $t_1$ and $t_2$. Let $A, D,$ and $U$ be the sets of nodes added to, deleted from, and updated during $t_1$ and $t_2$ to transform $W_1$ to $W_2$. Then,

- $W_\Delta^+ = (Z_\Delta^+, S_\Delta^+, T_\Delta^+)$ is called a $\Delta^+$-web table where $S_\Delta^+ = S_1$, $T_\Delta^+ \subseteq T_2$ and, for each Web tuple $w_i \in T_\Delta^+$, $\forall 0 < i \leq |T_\Delta^+|$, there exists a node $n(w_i)$ such that $n(w_i) \notin T_1$ and $n(w_i) \in A$.

- $W_\Delta^- = (Z_\Delta^-, S_\Delta^-, T_\Delta^-)$ is called a $\Delta^-$-web table where $S_\Delta^- = S_2$, $T_\Delta^- \subseteq T_1$ and, for each Web tuple $w_i \in T_\Delta^-$, $\forall 0 < i \leq |T_\Delta^-|$, there exists a node $n(w_i)$ such that $n(w_i) \notin T_2$ and $n(w_i) \in D$.

- $W_\Delta^u = (Z_\Delta^u, S_\Delta^u, T_\Delta^u)$ is called a $\Delta^u$-web table where $S_\Delta^u$ is generated from $Z_1$ and $S_2$ and, for each Web tuple $w_i \in T_\Delta^u$, $\forall 0 < i \leq |T_\Delta^u|$, at least one of the following conditions must be true:

1. If there exists a node $n_1(w_i)$ such that $n_1(w_i) \in U$ and $n_1(w_i) \notin (D \cup A)$, then there must exist a node $n_2(w_i)$ such that $url(n_2(w_i)) = url(n_1(w_i))$ and $date(n_1(w_i)) \neq date(n_2(w_i))$.

2. If there exists a node $n_3(w_i)$ such that $n_3(w_i) \notin U$, then $n_3(w_i) \notin (D \cup A)$ must be true. In this case, there must not exist a node $n_1(w_i)$ such that $url(n_3(w_i)) = url(n_1(w_i))$.

3. If there exists a node $n_4(w_i)$ such that $n_4(w_i) \notin (D \cup A \cup U)$, then there must not exist another node $n_3(w_i)$ such that $url(n_4(w_i)) = url(n_3(w_i))$.

Typically, the delta Web tables reflect the net effect of Web site modification, that is, they contain only the net result of successive modification of a set of relevant documents in the Web. Note that, in most of the cases, the size of these delta Web tables will be much smaller than $W_1$ or $W_2$. Representing changes in the form of a set of delta Web tables enables us to view the history of a Web table as a combination of a single Web table snapshot and a collection of delta Web tables. We can obtain various states of a Web table by starting with a single Web table and applying some sequence of Web delta to it. Also, to minimize the storage cost, we materialize only a single Web table and a set of delta Web tables in lieu of the various states of the Web table.

Observe that the representation of Web deltas using delta Web tables are comparable to delta relations in the relational model. In a relational database, deltas usually are represented using delta relations: For a relation $R$, delta relations inserted($R$) and deleted($R$) contain the tuples inserted to and deleted from $R$, while delta relations old-updated($R$) and new-updated($R$) contain the old and new values of updated tuples [26]. Similarly, we store the Web documents which are added, deleted, or modified in $\Delta^+$, $\Delta^-$, and $\Delta^u$-Web tables, respectively.

4.4 Decomposition of Change Detection Problem

The problem of detecting and representing changes can now be decomposed into two parts:

1. Construction of the joined and outer joined Web tables from $W_1$ and $W_2$. We discuss the construction of these Web tables in the next section.

2. Use the joined and outer joined Web tables to generate a set of delta Web tables, i.e., $W_\Delta^+$, $W_\Delta^-$, and $W_\Delta^u$, containing Web deltas. We discuss this in Section 6.

5 Web Join and Outer Web Join

In this section, we briefly introduce the Web join and outer Web join operators. We first introduce these operators and then discuss their algorithm. Note that, in this paper, we discuss the Web join and outer Web join only to the extent that it is necessary to understand the concept of change detection and representation in WHOWEDA. The reader may refer to [4] for details.

5.1 Web Join

The Web join operator is used to combine two Web tables by joining a Web tuple of one table with a Web tuple of another table whenever there exist joinable nodes. Let $w_a \in W_1$ and $w_b \in W_2$ be two Web tuples. Then, these tuples are joinable if there exist at least one node in $w_a$ which is joinable to a node in $w_b$. The joined Web tuple contains the nodes from both the input Web tuples. We materialize the joined Web tuple in a separate Web table. As one of the joinable nodes in each joinable node pair is superfluous, we remove one of them from the joined Web tuple.

To perform a Web join operation on Web tables $W_1$ and $W_2$, a pair of Web tuples is selected, one from each Web table, and all of the pairs of nodes are evaluated to determine if there exist joinable nodes. The process is repeated for all $|W_1| \times |W_2|$ pairs of Web tuples. If there exist joinable nodes in a pair of Web tuples, then the Web tables are joinable. Formally, two Web tables $W_1$ and $W_2$ are joinable if $w_a \in W_1$ and $w_b \in W_2$ are joinable where $0 < a \leq |W_1|, 0 < b \leq |W_2|$. We express the Web join between $W_1$ and $W_2$ as $W_1 \bowtie W_2$.

Example 1. Consider the Web tables Drugs and New Drugs.

The joined Web table of these two Web tables is shown in Fig. 4. The nodes $b_{01}$, $u_{01}$, $d_{01}$, and $k_0$ in the first Web tuple in Drugs are identical to those in the first Web tuple in New Drugs, as these nodes remain unchanged during the transition. The joined Web tuple generated by concatenating these two Web tuples over the nodes $b_{01}$, $u_{01}$, $d_{01}$, and $k_0$ is shown in Fig. 4 (the first Web tuple). Similarly, the second joined Web tuple in Fig. 4 is the result of joining the second Web tuples in Drugs and New Drugs. The third joined Web tuple is generated by joining the first Web tuple in Drugs to the second Web tuple in New Drugs, over the node $b_{01}$, etc. Observe that the third, fourth, fifth, and sixth Web tuples in Drugs do not participate in the Web join process as the nodes in these Web tuples are not identical to any nodes in New Drugs.
5.2 Outer Web Join

The Web tuples that do not participate in the Web join operation (dangling Web tuples) are absent from the joined Web table. In certain situations, it is necessary to identify the dangling Web tuples from one or both of the input Web tables. The outer Web join operation enables us to identify them. Depending on whether the outer-joined Web table must contain the nonparticipant Web tuples from the first or second Web tables, we define two kinds of outer Web join: the left-outer Web join and the right-outer Web join, respectively. Formally, given two Web tables $W_1$ and $W_2$, the left-outer web join and right-outer web join on these two Web tables are denoted by $W_1 \hat{\bowtie} W_2$ and $W_1 \hat{\bowtie} W_2$, respectively, where the symbols $\hat{\bowtie}$ and $\hat{\bowtie}$ correspond to the different flavors of outer Web join. The resultant Web table $Wo$ for a left-outer Web join or right-outer Web join will contain the dangling Web tuples from $W_1$ or $W_2$, respectively.

Example 2. Consider the Web tables Drugs and New Drugs in Fig. 2 and Fig. 3, respectively. The Web tuples in Drugs and New Drugs, which are associated with the side effects and uses of “Beta Carotene,” a drug used for cancer (third Web tuple), do not participate in the Web join process as the content of all the nodes in the Web tuple in New Drugs has changed with respect to those in Drugs. The link structure of the Web tuple related to “Vasomax” has been modified after 15 January, 2001, and none of the nodes in this Web tuple in Drugs are joinable to the corresponding node in the Web tuple related to the drug “Hirudin” in Drugs. Four Web tuples in Drugs are dangling Web tuples. Performing a left outer join on these two Web tables enables us to identify these dangling Web tuples (Fig. 5a).

Now consider New Drugs. The last two Web tuples dealing with “Viagra” and “Tolcapone” did not exist in the previous version, as these drugs were added to the Web site after 15th January, 2001. Moreover, all of the nodes in the Web tuples related to “Vasomax” and “Beta Carotene” are modified. Thus, these tuples are dangling Web tuples. Performing a right-outer Web join on these two Web tables enables us to identify these dangling Web tuples in New Drugs (Fig. 5b).

Observe that, although the Web tuple related to “Niacin” in New Drugs does not appear in Drugs, it is not a dangling Web tuple as the node $b_2$ in this tuple is joinable with the corresponding node in the Web tuple related to the drug “Hirudin” in Drugs.

5.3 Algorithms of Web Join and Outer Web Join

In this section, we describe the algorithms for Web join and outer Web join operations. We first describe the algorithm for computing the joinable nodes in the Web tables participating in Web join (Fig. 6). The algorithms of Web join and outer Web join operators use this algorithm to determine the joinable nodes.

The algorithm in Fig. 6 takes, as input, the set of node ids in $W_1$ and $W_2$, denoted by $N_1$ and $N_2$ and returns, as output, the set of joinable node ids in $W_1$ and $W_2$ (denoted by $J$). Comparing the URLs and dates of each possible pair of Web pages represented by the nodes in $N_1$ and $N_2$ to identify the joinable nodes can have significant impact on the total cost and the query response time for large Web tables to render Web join impractical. A practical solution to this problem is to first, subdivide the set of nodes in each Web table into a disjoint and smaller subset $U$ containing the ids of the nodes which occur in $W_1$ as well as in $W_2$. This is because a node...
can be potentially joinable only if it occurs in both of the
input Web tables. Recall that two nodes have identical
URLs if their ids are identical. Thus, each element in
\(U\) is the identifier of a node that exists in \(W_1\) and \(W_2\). However, as
nodes with identical URLs in different Web tables may not
always be identical in their contents, for each node id \(u_k \in U\) the algorithm retrieves the version ids of \(u_k\) in \(W_1\) and \(W_2\) (denoted as \(v_1(k)\) and \(v_2(k)\), respectively). If the version ids
are identical, then the nodes represented by \(u_k\) in \(W_1\) and
\(W_2\) are identical and are considered as joinable nodes. In
that case, the algorithm inserts \(u_k\) in \(J\). For example, consider the Web tables Drugs and New Drugs. Suppose we
wish to find the joinable nodes in these two Web tables. The
algorithm will output the set of joinable node ids in these
two Web tables, i.e., \(J = \{b_0, b_2, u_0, u_1, d_0, d_1, k_0, k_1, k_7\}\).

The algorithms of Web join and outer Web join are given
in Fig. 6, Fig. 7, and Fig. 8. Note that the analysis we present is the worst-case analysis. We assume the
following for the purpose of complexity analysis:

- The node-ids of a Web table are stored in the Web
tuple pool in a B+- tree.
- The node-ids are also arranged in the node pool in a
B+- tree. The version-ids are stored with correspond-
ing node-ids in the node pool. Note that the look-up
time for each node-id is \(O(\log N)\) if we have \(N\) nodes
in a Web table stored as a B+- tree.
- Let \(n_1\) be the number of nodes in Web table \(W_1\) and
\(n_2\) be the number of nodes in Web table \(W_2\). When
\(n_1 = n_2\), we say that both tables have the same
number of nodes. That is, the number of nodes
deleted from \(W_1\) during transition is same as the
number of nodes inserted in \(W_2\).
- Let \(W_{t1}\) and \(W_{t2}\) be the average number of tuples in
\(W_1\) and \(W_2\), respectively.
- Let \(N_{w1}\) and \(N_{w2}\) be the average number of nodes in
each Web tuple in \(W_1\) and \(W_2\), respectively.

5.4 Complexity Analysis of Algorithms
In this section, we will analyze the time complexity of the
algorithms in terms of the number of steps required for the
algorithms in Fig. 6, Fig. 7, and Fig. 8. Note that the analysis
we present is the worst-case analysis. We assume the
following for the purpose of complexity analysis:

- The node-ids of a Web table are stored in the Web
tuple pool in a B+- tree.
- The node-ids are also arranged in the node pool in a
B+- tree. The version-ids are stored with correspond-
ing node-ids in the node pool. Note that the look-up
time for each node-id is \(O(\log N)\) if we have \(N\) nodes
in a Web table stored as a B+- tree.
- Let \(n_1\) be the number of nodes in Web table \(W_1\) and
\(n_2\) be the number of nodes in Web table \(W_2\). When
\(n_1 = n_2\), we say that both tables have the same
number of nodes. That is, the number of nodes
deleted from \(W_1\) during transition is same as the
number of nodes inserted in \(W_2\).
- Let \(W_{t1}\) and \(W_{t2}\) be the average number of tuples in
\(W_1\) and \(W_2\), respectively.
- Let \(N_{w1}\) and \(N_{w2}\) be the average number of nodes in
each Web tuple in \(W_1\) and \(W_2\), respectively.

```
Input: Set of node ids \(N_1\) and \(N_2\) of \(W_1\) and \(W_2\).
Output: Joinable node id set \(J\).

(1) \(U = N_1 \cap N_2\);
(2) for \(i = 1\) to \(|U|\) {
(3) \(v_1(i) = \text{GetVersionId}(U(i), W_1)\); /* returns the version id of a node having id \(U(i)\) in \(W_1\) */
(4) \(v_2(i) = \text{GetVersionId}(U(i), W_2)\);
(5) if \((v_1(i) = v_2(i))\) /* The nodes are joinable */
(6) Insert \(U(i)\) in joinable node id set \(J\);
(7) else
(8) The nodes are not joinable;
(9) }
(10) return \(J\);
```

Fig. 6. Algorithm for computing joinable nodes.

Fig. 5. Outer Web join. (a) Left outer Web join and (b) Right outer Web join.
1. Note that \( n_1 \hat{=} N_1 W_1 \) and \( n_2 \hat{=} N_2 W_2 \).
2. The size of an I/O block is \( B = 8,192 \) bytes.
3. The size of a tuple in the tuple pool is smaller than the I/O block so that each I/O access can obtain \( B/nt \) tuples, where \( nt \) is the average size of the tuple.
4. The size of a node in the node pool is smaller than the size of the I/O block so one access can obtain \( B/ns \) nodes, where \( ns \) is the average size of the node in the node pool.

To get the node-ids from the Web tuple pool, we need to access the tuples from the Web tuple pool. Thus, the number of block accesses needed is \( W_1 = \frac{n_1 W_1}{B} \) and \( W_2 = \frac{n_2 W_2}{B} \) for \( W_1 \) and \( W_2 \), respectively. For accessing the nodes from the node pool, \( nt \) will be replaced by \( ns \).

Analysis of the algorithm in Fig. 6 to find joinable nodes.

Step (1) of the algorithm will need \( O(n_1 \log n_2) \) steps, as each look up operation induces a time complexity of \( O(\log n_2) \). For simplicity, let us assume that the number of nodes in the two tables are the same (that is, the number of nodes deleted is equal to the number of nodes which have been inserted during the transition of Web page), then the complexity is \( O(n \log n) \). For simplicity, let us assume that the number of joinable nodes is \( n/2 \), where \( n \) is the number of nodes in \( W_1 \) and \( W_2 \). That is, half of the nodes remain unchanged during the transition. Hence, Step (3) will be performed \( n/2 \) times. To get the version-ids associated with each node-id of each node from the node pool will take \( \log N \) steps where \( N \) is the total number of nodes in the node pool (i.e., total nodes in the warehouse). Thus, Steps (4) and (5) will induce complexity of \( n/2 \times (O(\log N) + O(\log N)) \), which is \( O(n \log N) \). Hence, the total complexity is \( O(n \log N) + O(n \log n) \), which is \( O(n \log N + n \log n) \). Since \( n < N \), the complexity will be \( O(n \log N) \). Note that the I/O cost involved here is negligible, as only ids have been accessed to determine the joinable nodes.

Analysis of the Web join algorithm in Fig. 7. For simplicity, here again we assume that both of the tables have \( n \) nodes. Steps (1) and (2) both together will induce a time complexity of \( O(2 \log n) \). Thus, the total complexity of the first three steps to find joinable nodes is \( O(n \log N) + O(n \log n) + O(2 \log n) \), which is

\[
O(n \log N) + O(n \log n + 2 \log n) = O(n \log N + (n + 2) \log n).
\]

Since \( n < N \), it will be \( O(n \log N) \). Step (4) will be executed \( W_1 \) times. Step (5) will need \( \frac{n_1 n_1}{B} \) I/O accesses for each tuple. Steps (6) and (7) will need \( O(n \log (N_1/2)) \) steps where \( N_1 \) is the average number of nodes in a Web tuple in \( W_1 \) and we assumed that there are \( n/2 \) joinable nodes. Thus, the time complexity of Steps (4) to (7) will be \( W_1 (O(n \log (N_1/2)) \) plus the I/O cost of accessing \( W_1 = \frac{n_1 N_1}{B} \) blocks of memory for all the Web tuples of \( W_1 \). Similarly, there will be \( W_2 = \frac{n_2 N_2}{B} \) steps to find the joinable tuples from \( W_2 \) plus the I/O cost. Step (20)
Input: Web tables $W_1 = (Z_1, S_1, T_1)$ and $W_2 = (Z_2, S_2, T_2)$.
Output: Left or right outer web joined table $W_o$.

1. $N_1 = \text{GetAllNodeIds}(W_1)$;
2. $N_2 = \text{GetAllNodeIds}(W_2)$;
3. $J = \text{ComputeJoinableNodeIds}(N_1, N_2); /* Figure 6 */$
4. if (left outer web join is to be performed) {
   5.   $S_0 = S_1; /* Web schema of the result web table */$
   6.   danglingNodeSet = $N_1 - J; /* Ids of nodes that do not participate in web join */$
   7.   Let temporary web table $W_t = W_1$;
8. } else {
9.   $S_0 = S_2$;
10.  danglingNodeSet = $N_2 - J$;
11.  Let temporary web table $W_t = W_2$;
12. }
13. for (r = 1 to $|W_t|$) {
14.   Get tuple $w_r$;
15.   tupleNodeIdsSet[r] = GetTupleNodeIds($w_r$);
16.   if (tupleNodeIdsSet[r] ∩ danglingNodeSet = tupleNodeIdsSet[r]) /* dangling web tuple */ {
17.     for (d = 1 to |tupleNodeIdsSet[r]|) {
18.       Insert tupleNodeIdsSet[r][d] into resultantNodeIdsSet;
19.     }
20.     Store tuple $w_r$ in web tuple pool of $W_o$;
21.   } else {
22.   }
23.   $w_r$ contains joinable nodes;
24. }
25. for (k = 1 to |resultantNodeIdsSet|) {
26.   Retrieve the node having id resultantNodeIdsSet[k] from table node pool of $W_t$;
27.   Store the node in resultant table node pool of $W_o$;
28. }
29. return $W_o$.
30. 

Fig. 8. Algorithm of outer Web join.

will take $J_N \times (n/2)$ steps where $J_N$ is the number of joinable tuples and $n/2$ is the average number of nodes in the two Web tables. Here, we assume that indexes are built in the storage structure on the joinable nodes so that the cost of accessing them is $O(1)$.

The cost of Step (21) can be calculated as in Step (1) or (2). Step (22) will take $J_N \times N_{JN}$ where $N_{JN}$ is the average number of nodes in each joinable tuples. Step (23) will take $\log (n_1 + n_2)/2$ steps in the worst case of searching a node in both the table pools plus $n_1/B$ block accesses for fetching each node. Step (24) will take $\log (J_N \times N_{JN})$, on the average, to insert a node in the node pool tree plus $n_2/B$ I/O accesses for each node. Note that we do not calculate the time complexity of Step (26) as the schema generation process does not influence the Web delta detection problem and, hence, it is beyond the scope of the paper. For change detection problem, Step (26) may be ignored. The reader may refer to [6] for discussion on the schema generation process. Thus, to sum up, the Web join process induces both the operational complexity and the I/O access cost. The accumulated operational complexity after assuming that $n_1 = n_2$, ($W_t = W_o$) = $W_t$, ($N_w = N_{w2}$) = $N_w$, is

$$O(n \log N) + 2 \times W_t \times (O(n \log (N_w/2)) + J_N \times (n/2) + (J_N \times N_{JN} \times \log (J_N \times N_{JN}))(1 + O(\log n)).$$

By assuming that the values of $W_t$, $N_w$, $J_N$, and $N_{JN}$ are very small as compared to $n$ and since $n$ is smaller than $N$, we get the operational complexity as $C \times O(n \log N)$, where $C$ is a constant. The total I/O cost is

$$((2/B) \times ((n \times n_1) + (J_N \times N_{JN}) \times n_2)).$$

**Analysis of the outer Web join algorithm.** The number of steps needed in the algorithm in Fig. 8 to find outer Web join will be the same as that for finding the joinable Web tuples. This is because the number of dangling nodes (nodes which are not joinable) will be $n/2$ as we assumed that the joinable node set is of size $n/2$. Thus, to find the Web tuples in each left outer and right out Web join will need the same number of steps as in the case of the joinable Web tuples described above.

The complexity of all other algorithms in the next section can be calculated on similar lines since they all use similar type of constructs and reasoning. Therefore, they are left for the readers to verify.

### 6 Generating Delta Web Tables

This section initiates a discussion on the algorithm to detect and represent different types of change operations using the Web join and outer Web join operations. We describe the Algorithm Delta that generates a set of delta Web tables to detect and represent Web deltas. We first describe the generation of delta tables informally and then provide the complete algorithm.
6.1 Outline of the Algorithm

The algorithm for delta Web table generation can be best described by the following four phases: the join tables generation phase, the delta node identification phase, the delta tuples identification phase, and the delta table generation phase. We discuss these phases one by one.

Phase 1: Join Tables Generation Phase. This phase takes, as input, two Web tables, new and old versions, and generates the joined, right outer joined, and left outer joined Web tables. For instance, after this phase the Web tables in Fig. 4, Fig. 5a, and Fig. 5b are generated from Drugs and New Drugs.

The right outer join operation on W1 and W2 may create three categories of dangling Web tuples:

1. Web tuples which are added to W1 during the polling times t1 and t2. These tuples may contain some new nodes and the contents of the remaining ones have been changed.
2. Tuples in which all of the nodes have undergone content modification.
3. Tuples in which some of the nodes are new and the contents of the remaining ones have changed, but these tuples existed in W1. For example, consider the Web table in Fig. 5b. The last two Web tuples belong to the first category. The first Web tuple belongs to the second category and the second Web tuple is an example of third category.

The Web join operation on W1 and W2 may contain the following three types of Web tuples:

1. Web tuples in which all of the nodes are joinable nodes. These tuples are the results of joining two versions of Web tuples in W1 and W2 that have remained unchanged during t1 and t2.
2. Web tuples in which some of the nodes are joinable nodes and remaining nodes are the result of insertion, deletion, or modification operations during the transition.
3. Some of the nodes are joinable nodes and out of the remaining ones, some are the result of insertion, deletion, or modification, and the remaining ones are not joinable in this Web tuple, but they have remained unchanged during the transition. That is, these nodes may be joinable nodes in some other joined Web tuple(s).

While generating delta Web tables, the algorithm ignores the first category of Web tuples in the joined Web table as it does not reflect any changes. For instance, in the joined Web table in Fig. 4, all of the Web tuples are of second and third categories. Specifically, the first two and the last three tuples contain nodes whose contents are modified. The fourth Web tuple contains nodes whose contents are modified as well as a node k4 which is inserted during t1 and t2. Finally, the fifth tuple contains a node u8 which is deleted as well as a set of nodes which are modified. Hence, these Web tuples represent the second category. On the other hand, the third Web tuple represents the third category. This is because a0 represents a modified node and the nodes u0, u1, d0, d1, k0, and k1 are not joinable in this Web tuple but they are joinable nodes in the first and second Web tuples.

Similar to the right outer joined Web table, the left outer joined table may contain the following three categories of Web tuples:

1. Web tuples which are deleted from W1. These tuples do not occur in W2.
2. Tuples in which every node has undergone content modification.
3. Tuples in which some nodes are deleted from W1 and remaining ones have been modified.

The new and old versions occur in both of the tables in W1 and W2. For instance, the second and fourth Web tuples in Fig. 5a belong to the first category. The first and the third Web tuples belong to the second and third categories, respectively.

Phase 2: Delta Nodes Identification Phase. In this phase, the nodes which are added, deleted, or modified during t1 and t2 are identified. This phase takes as input the Web tables W1 and W2 and the set of joinable nodes from the joined table and generates sets of nodes which are added, deleted, or modified during the time interval. Thus, nodes which exist in W2 but not in W1 are the new nodes that are added to W1. Similarly, nodes which only exist in W1, but not in W2, are the nodes that are removed from W1. Furthermore, the nodes which are not joinable nodes, but they exist in W1 as well as W2 are essentially the nodes that have undergone content modification during t1 and t2. For instance, \{b1, u4, u5, u6, u8, d4, d5, d12, k5, k6, k12\} are the ids of nodes which appear in Drugs but not in New Drugs. Hence, these nodes were removed during transition. Similarly, \{k4, u9, k8, d6, d7, u13, u12, d9, k9, u10, u11, d10, k10\} are the ids of nodes that exist in New Drugs but not in Drugs. Hence, these nodes were added during t1 and t2. Finally, the nodes with ids a0, b1, d2, k2, u2, d3, k3, b4, and u12 appear in both of the web tables but are not joinable. Hence, these nodes have undergone content modification.

Observe that, at this point we have identified the nodes which are inserted, deleted, or modified during the transformation. Next, the algorithm proceeds to determine how these nodes are related to one another and how they are associated with those nodes which have remained unchanged during t1 and t2.

Phase 3: Delta Tuples Identification Phase. In the delta tuples identification phase, we are interested in identifying those Web tuples which contain nodes which are added, deleted, or modified during the transition. It should be clear that we are not just simply identifying these Web tuples as it can be done by inspecting W1 and W2 without performing any Web join or outer Web join operations. Our objectives are the following:

- In case of added or deleted nodes, we are not simply interested in identifying tuples containing these nodes only, but also how these nodes are linked to or related to the existing nodes (nodes which prevailed during the transition). Moreover, we wish to determine how the new or deleted nodes are related to one another.
For nodes which have undergone content modification during the transition, we wish to determine how these nodes are linked to one another and to those nodes which have remained unchanged. We also wish to present the old and new versions of the nodes in a single tuple so that a user can view them effectively. Finally, we wish to highlight the changes in the overall hyperlink structure due to the content modification.

To achieve this, we scan the joined and outer joined Web tables. The delta tuples identification phase takes, as input, these Web tables and the sets of nodes which are added, deleted, or modified. It returns, as output, sets of tuples containing nodes which are added, deleted, and modified, respectively. In the remaining portion of this paper, these sets are denoted as \textit{insertTupleSet}, \textit{deleteTupleSet}, and \textit{updateTupleSet}, respectively. Observe that the nodes which are added during the transition can occur in the following tables:

- In the right outer joined table, if the remaining nodes in a tuple containing the new nodes are modified they are not joinable. The second, third, and fourth Web tuples in Fig. 5b are examples of such tuples.
- In the joined Web table, if some of the nodes in the tuple containing new nodes have remained unchanged during the transition they are joinable. For instance, the fourth Web tuple in Fig. 4 is an example of such a Web tuple where $k_4$, $u_3$, and $d_7$ are the new nodes and $b_2$ has remained unchanged during the transition and therefore joinable.

Hence, we need to scan the joined and right outer joined tables to identify the tuples containing nodes which are inserted during $t_1$ and $t_2$. Similarly, the nodes which are deleted during the transition may occur in the following two Web tables:

- In the left outer joined table, if the remaining nodes in a tuple containing the deleted nodes are modified they are not joinable. The second, third, and fourth Web tuples in Fig. 5a are examples of such tuples.
- In the joined Web table, if some of the nodes in the tuple containing deleted nodes have remained unchanged during the transition they are joinable. For instance, the seventh Web tuple in Fig. 2 contains the node $u_3$ which is deleted. As the nodes $d_9$ and $k_7$ have remained unchanged during the transition, these nodes are joinable. The tuple containing this deleted node can be detected from the joined Web table in Fig. 4 (third Web tuple).

Thus, the algorithm scans the left outer joined and joined tables to retrieve the tuples containing deleted nodes. Finally, the nodes which are modified during the transition can be identified by inspecting all of the three Web tables:

- Tuples in the left and right outer joined tables which do not contain any new or deleted nodes represent the old and new versions of these nodes, respectively. These Web tuples do not occur in the joined table as all of the nodes are modified. For instance, the first Web tuples in Fig. 5a and Fig. 5b are examples of such tuples.
- Tuples in the left and right outer joined tables that contain modified nodes as well as inserted or deleted nodes. Note that these modified nodes may not appear in the joined Web table if no other joinable Web tuples contain these modified nodes.
- Tuples in the joined Web tables whose some of the nodes represent the old and new versions of these modified nodes. For instance, the first Web tuple in Fig. 4 contains the old and new versions of $a_0$.

**Phase 4: Delta Web Tables Generation Phase.** Finally, the three types of delta Web tables are generated in this phase. It takes, as input, the three sets of tuples, i.e., \textit{insertTupleSet}, \textit{deleteTupleSet}, and \textit{updateTupleSet} generated in the previous phase and generates the delta Web tables from these sets. The procedure to generate these tables is straightforward. The tuples in \textit{insertTupleSet} are stored in $\Delta^+\text{-Web}$ table. The tuples in \textit{deleteTupleSet} and \textit{updateTupleSet} are stored in $\Delta^-\text{-Web}$ and $\Delta^M\text{-Web}$ tables, respectively.

Next, we illustrate the generation of delta Web tables informally with an example given below.

**Example 3.** Consider the two Web tables \textit{Drugs} and \textit{New Drugs} in Fig. 2 and Fig. 3. We would like to find the various change operations that transform \textit{Drugs} into \textit{New Drugs}. Changes may include inserting nodes, deleting nodes, and updating nodes in \textit{Drugs}. For each type of these changes, we create the $W_{\Delta^+}$, $W_{\Delta^-}$, and $W_{\Delta^M}$ tables. We discuss the generation of each delta Web table in turn. Fig. 9 depicts the $\Delta^+\text{-Web}$ table. The patterned boxes in this figure in each Web tuple are the old and new versions of the nodes. For example, the second Web tuple in Fig. 9 contains the old and new versions of the nodes $a_0$, $u_2$, $d_9$, and $k_3$, along with the joinable node $u_2$ (content of $u_2$ has remained unchanged during the transition). Each Web tuple shows how the set of modified nodes is related to one another and with the joinable nodes. Observe that the first four Web tuples are extracted from the joined Web table in Fig. 4. The last Web tuple (enclosed in a dotted box) is the result of the integration of two Web tuples—one from the left outer joined Web table in Fig. 5a and another from the right outer joined table in Fig. 5b.

Fig. 10a illustrates the $\Delta^-\text{-Web}$ table. The black boxes in each Web tuple are the new nodes inserted into \textit{Drugs} during 15 January, 2001 and 15 February, 2001. Similar to the $\Delta^+\text{-Web}$ table, each Web tuple in the $\Delta^-\text{-Web}$ table shows how the new nodes are related to other relevant nodes in the Web table. Note that the last three Web tuples in Fig. 10a are extracted from the right outer joined Web table in Fig. 5b. However, as the node $b_9$ in the first Web tuple is a joinable node, the new nodes $k_4$, $u_3$, and $d_7$ in this tuple are identified from the fourth Web tuple of the joined Web table in Fig. 4.

Finally, Fig. 10b depicts the $\Delta^M\text{-Web}$ table containing all of the nodes that are deleted from \textit{Drugs}. The last three Web tuples are extracted from the Web table in Fig. 5a. However, the tuple containing the deleted node $u_9$ is extracted from the fifth Web tuple in the joined Web table in Fig. 4. Observe that we do not materialize the joined Web tuples containing new or deleted nodes in $W_{\Delta^+}$ and $W_{\Delta^-}$, respectively. Instead, we extract the
original Web tuple containing these nodes from the joined Web tuple and materialize them in \( W_{\Delta^+} \) and \( W_{\Delta^-} \), respectively.

### 6.2 Algorithm Delta

We now provide the formal algorithm for the four phases discussed in the previous section. We describe how, given two Web tables \( W_1 \) and \( W_2 \) created by a polling global coupling operation at polling times \( t_1 \) and \( t_2 \), we compute a set of delta Web tables corresponding to various types of changes to transform \( W_1 \) to \( W_2 \). The pseudocode for this algorithm is given in Fig. 11. It takes, as input, two Web tables \( W_1 \) and \( W_2 \) created by a coupling query \( Q \) at polling times \( t_1 \) and \( t_2 \), respectively. It returns, as output, a set of delta Web tables \( W_{\Delta^+} \), \( W_{\Delta^-} \), and \( W_M \). The steps to generate these delta Web tables are as follows.

#### 6.2.1 Algorithm for Phases 1 and 2

Steps (1) - (4) implement the first phase of the algorithm and generates the joined, the right, and the left outer joined Web tables from \( W_1 \) and \( W_2 \) (Line (4)). In the algorithm, these Web tables are denoted as \( W_j \), \( W_{ro} \), and \( W_{lo} \), respectively. Steps (5) - (7) implement the second phase of the algorithm and identify the node ids which are added to \( W_1 \), the node ids which are removed from \( W_1 \), and ids of the nodes that have undergone content modification during \( t_1 \) and \( t_2 \).

#### 6.2.2 Algorithm of Phase 3

We now discuss the algorithm for implementing the delta Web tuples identification phase. To determine the association of nodes (represented by the identifiers in \( addNodeSet \), \( delNodeSet \), and \( updateNodeSet \)) with each other and with other relevant nodes in \( W_1 \), we identify the Web tuples in \( W_j \), \( W_{ro} \), and \( W_{lo} \) containing these nodes and store them in the sets of Web tuples denoted in the algorithm as \( insertTupleSet \), \( deleteTupleSet \), and \( updateTupleSet \), respectively. Each element in \( insertTupleSet \) and \( deleteTupleSet \) are Web tuples containing nodes that are inserted to or deleted from \( W_1 \) during \( t_1 \) and \( t_2 \). Each element in \( updateTupleSet \) is an integrated Web tuple containing old and new versions of the nodes which have undergone content modification. Note that these sets of Web tuples encapsulate the various change operations introduced in Section 4.

---

Fig. 9. \( \Delta^+ \)-Web table.

Fig. 10. (a) \( \Delta^+ \) Web tables and (b) \( \Delta^- \) Web tables.
Steps (8) - (22) in Fig. 11 present the pseudocode for the delta Web tuples identification phase. Step (9) computes those nodes which are updated during the transition but are not captured by the joined Web table. The set $K$ contains ids of those updated nodes which do not exist in $N_j$, i.e., $K$ represents those updated nodes which are not present in $W_j$, but in $W_{ro}$ and $W'_o$. If $n$ is a node in $K$, then $n$ does not occur in $W_j$. That is, the tuple containing $n$ is a dangling Web tuple and consequently is ignored by the Web join operation. That is, if $n$ represents a node which has undergone update during $t_1$ and $t_2$, then the Web tuples containing $n$ in $W_1$ and $W_2$ must be dangling Web tuples. Otherwise, $n$ must occur in the joined Web table. As a result, the tuples containing $n$ in $W_1$ and $W_2$ must be captured by the left and right outer joined Web tables, respectively ($W_{ro}$ and $W_{lo}$). If $K = \emptyset$, then all updated nodes are captured by the joined Web table.

Next, (Steps (10) and (11)) the sets of node ids of new and deleted nodes are copied to the sets $A$ and $D$, respectively. This is because, subsequently, $A$ and $D$ need to be updated every time we identify new and deleted nodes while scanning the right and outer joined Web tables. However, we do not wish to modify the $addNodeSet$, $delNodeSet$, and $updateNodeSet$ since the algorithm is going to use them again to generate the delta Web tables in Steps (23) - (25).

Finally, $U$ (Step (12)) represents those modified nodes which occur only in the joined Web table $W_j$, i.e., $U \cap K = \emptyset$. Then, Steps (13) and (14) scan the right and left outer joined Web tables to identify the Web tuples containing inserted, deleted, and updated (if any) nodes. We elaborate on these algorithms now.

**Algorithm of DeltasFromRightOuter** ($A$, $K$, $W_{ro}$, $temp1$, $temp2$) (Step (13) in Fig. 11). This algorithm takes as input the set of node ids which are added during the transition, i.e., $A$, the set $K$, the right outer joined Web table $W_{ro}$, and two empty sets $temp1$ and $temp2$ to store specific Web tuples. It returns, as output, a set of tuples containing the nodes which are inserted during $t_1$ and $t_2$, denoted by $insertTupleSet$, modified $A$ and $K$, and $temp1$ and $temp2$ (possibly nonempty). Let us elaborate on the purpose of the tuple sets $temp1$ and $temp2$. $temp1$ stores those Web tuples in the right or left outer joined tables that do not contain any new or deleted nodes (Category 2 Web tuples as discussed in the previous section). That is, the tuples in $temp1$ represent those Web tuples in which all of the nodes content are modified during the transition. For instance, $temp1$ will store the first Web tuples in Fig. 5a and Fig. 5b. Recall that in $updateTupleSet$, the old and new versions of a Web tuple are integrated and stored together. However, the right outer join operation only identifies the new version of the modified Web tuple. The old version of the tuple can be extracted from the result of the left outer join operation. Thus, the insertion of the Web tuples containing modified nodes into $updateTupleSet$ is deferred to the execution of the inspection of the left outer joined Web table. Consequently, we store these tuples temporarily in $temp1$. 

---

**Fig. 11. Algorithm delta.**

| Input: Web tables $W_1 = \langle Z_1, S_1, T_1 \rangle$, $W_2 = \langle Z_2, S_2, T_2 \rangle$.  
| Output: Delta web tables $W_{\Delta r} = \langle Z_{\Delta r}, S_{\Delta r}, T_{\Delta r} \rangle$, $W_{\Delta} = \langle Z_{\Delta}, S_{\Delta}, T_{\Delta} \rangle$ and $W_{\Delta m} = \langle Z_{\Delta m}, S_{\Delta m}, T_{\Delta m} \rangle$.  
| (1) $N_1 = GetAllNodeIds(W_1)$; /* Phase 1 */  
| (2) $N_2 = GetAllNodeIds(W_2)$;  
| (3) $J = ComputeJoinableNodeIds(N_1, N_2)$; /* Figure 6 */  
| (4) $W_{12} = GenerateResultTables(N_1, N_2, W_1, W_2, J)$ where $W_{12} = \{W_{ro}, W_{lo}, W_j\}$;  
| (5) $delNodeSet = N_1 - N_2$; /* Lds of the nodes deleted from $W_1$ */ /* Phase 2 */  
| (6) $addNodeSet = N_2 - N_1$; /* Lds of the nodes added to $W_1$ */  
| (7) $updateNodeSet = (N_1 - J - delNodeSet) \cup (N_2 - J - addNodeSet)$;  
| (8) $N_j = GetAllNodeIds(W_j)$; /* Phase 3 */  
| (9) Let $K = updateNodeSet - (N_j - J)$;  
| (10) Let $A = addNodeSet$;  
| (11) Let $D = delNodeSet$;  
| (12) Let $U = (N_j - J) \cap updateNodeSet$;  
| (13) $insertTupleSet = DeltasFromRightOuter(A, K, W_{ro}, temp1, temp2)$; /* Figure 12 */  
| (14) $deleteTupleSet = DeltasFromLeftOuter(D, K, W_{lo}, temp1, temp2)$;  
| (15) if ($|A| = 0$ and $|D| = 0$) { /* checks if all the inserted and deleted nodes are identified */  
| (16) $updateTupleSet = DeltasFromJoin( W_j, U, N_1, N_2)$; /* Figure 13 */  
| (17) else  
| (18) $updateTupleSet = DeltasFromJoin( A, D, U, W_j, insertTupleSet, deleteTupleSet)$; /* Figure 14 */  
| (19) if ($|temp1| \neq 0$)  
| (20) Insert into $updateTupleSet$ web tuples from $temp1$;  
| (21) if ($|temp2| \neq 0$)  
| (22) Insert into $updateTupleSet$ web tuples from $temp2$; /* Phase 4 */  
| (23) $W_{\Delta m} = CreateDeltaM(updateTupleSet, W_1, W_2, S_{12}, updateNodeSet)$;  
| (24) $W_{\Delta} = CreateDeltaPlus(insertTupleSet, W_1, W_2, addNodeSet)$;  
| (25) $W_{\Delta m} = CreateDeltaMinus(deleteTupleSet, W_1, W_2, delNodeSet)$;  
| (26) return $W_{\Delta r}, W_{\Delta m}$ and $W_{\Delta m}$;
On the other hand, temp2 contains those tuples from the left or right outer joined tables which contain the updated nodes not captured by the joined Web table. That is, temp2 contains tuples from Wro and Wlo where each tuple contains at least one node that is an element of K and some of the remaining nodes must be new or deleted nodes. Notice the differences between the tuples in temp1 and temp2. In temp1, all of the nodes in each Web tuple have undergone content modification. However, in temp2 each Web tuple must contain at least one node which is added or deleted during the transition and remaining nodes must have undergone content modification. Also observe that temp1 \∩ temp2 = \empty. Note that the nodes in a tuple in temp1 may also occur in K, however, it is not captured in temp2 because each Web tuple in temp2 contains one or more new or deleted nodes. Specifically, temp2 enables us to capture the Web tuples containing modified nodes that cannot be identified from the joined Web tables. For similar reasons as explained in the case of temp1, we defer the insertion of these tuples in the updateTupleSet. The pseudocode of this algorithm is given in Fig. 12.

Algorithm of DeltasFromLeftOuter (D, K, Wlo, temp1, temp2) (Step (14) in Fig. 11). Next, the algorithm Delta inspects the left outer joined table Wlo to identify the deleted or modified nodes. The pseudocode for this algorithm will be similar to Fig. 12 and, hence, it is omitted.

Algorithm of DeltasFromJoin (Wj, U, Ni, Nj) (Step (16) in Fig. 11). Finally, the Algorithm Delta in Fig. 11 proceeds to inspect the joined Web table Wj. If A = D = \empty, then the joined Web table will only contain the updated nodes. Consequently, Step (16) is executed. Each joined Web tuple in Wj is inspected to determine the existence of the old and new versions of the modified nodes. It may seem that each tuple containing dangling node(s) (nodes which are not joinable) may represent the old and new versions of the modified nodes. However, such assumption is not true. Note that we are specifically interested in those joined Web tuples which contain only both the old and new versions of the updated nodes. Note that not all joined Web tuples may satisfy this condition. For example, consider the joined Web table in Fig. 4. The fourth Web tuple contains the dangling nodes w2, k2, d2, etc. However, both the old and new versions of these nodes are missing in this tuple. Specifically, it exists in the last joined Web tuple. Hence, the last Web tuple is inserted in updateTupleSet but not the fourth Web tuple. Observe that this condition is checked in Step (7) of the algorithm in Fig. 13. For instance, for the last Web tuple, N1(wa) = \{a0, u2, d3, k3\}, N2(wa) = \{a0, u2, d3, k3\}, and X = \{a0, u2, d3, k3\}. Hence, X - \{N1(wa) \∩ N2(wa)\} = \empty and the joined tuple is inserted in updateTupleSet. However, for the fourth Web tuple,

\[ N1(wa) = \{a0, u3, d3, k3\}, N2(wa) = \{a0, u3, d7, k4\}, \]

and X = \{a0, u2, d3, k3\}. Hence, \[ X - \{N1(wa) \∩ N2(wa)\} = \{u2, k3, d3\}. \]

Consequently, the condition is not satisfied.

Algorithm of DeltasFromJoin (A, D, U, Wj, insertTupleSet, deleteTupleSet) (Step (18) in Fig. 11). If all of the new or deleted nodes are not identified after scanning Wlo and Wro then |A| \neq 0 or |D| \neq 0. Hence, in that case, Step (18) in Fig. 11 is executed. The pseudocode for the construct DeltasFromJoin is given in Fig. 14. Note that, in this case, the algorithm is not only looking for the Web tuples containing the updated nodes but also those Web tuples which contain the new or deleted nodes. We elaborate on these steps now. Note that Steps (2) - (10) are similar to the previous steps. Steps (12) - (25) are executed if not all of the dangling nodes in wa represent the new and old versions of the node. We explain these steps with examples. Consider the fifth Web tuple in Fig. 4. Here, Y = \{a0, b1, u2, u6\}, N1(w6) = N2(w6) = \{a0, b1, u2\}. Hence, \[ Y - \{N1(w6) \∩ N2(w6)\} = \{u6\}. \] Hence, the condition in Step (8) is evaluated false. As wa represents deleted node, i.e., wa \in D. Consequently, the condition in Step (12) is evaluated true. We insert the tuple in updateTupleSet and extract the seventh Web tuple in Drugs. Observe that the
seventh Web tuple in Drugs contains \( u_8 \) which is deleted during transition. Finally, we insert this tuple in the \textit{deleteTupleSet}.

At this point, the Web tuple containing the new nodes \( k_4, w_9, \) and \( d_7 \) in the joined Web table (fourth joined tuple) have not been identified yet. Note that, for this Web tuple, \( Y = D - (N_1(w_4) \cap N_2(w_4)) = \{v_2, k_5, d_8, k_4, w_9, d_7\} \). Also, \( Y \subseteq A \) or \( Y \subseteq D \) is not satisfied. Consequently, the condition in Step (12) is evaluated false. In order to identify this node, Steps (18) - (23) are executed. The condition in Step (19) is satisfied by \( Y \cap A = \{k_4, u_3, d_7\} \). Hence, Step (20) is executed and the original Web tuple (fifth Web tuple in New Drugs) is retrieved and inserted in \textit{insertTupleSet}. If \( Y \cap D \neq \emptyset \), then Step (22) is executed and the original Web tuple from \( W_1 \) is retrieved and inserted in \textit{deleteTupleSet}. Steps (24) and (25) update \( A \) and \( D \) by removing those ids which are already used to identify the Web tuples.

We have now identified all of the Web tuples containing the new or deleted nodes. We have also identified some of the tuples containing the updated nodes. The remaining tuples containing the updated nodes are identified from \textit{temp1} and \textit{temp2} (Steps (19) - (22) in Fig. 11). The Web tuples in \textit{temp1} representing the old and new versions of the modified nodes are inserted in \textit{updateTupleSet} as a single Web tuple. For instance, the first Web tuples in Fig. 5a and Fig. 5b are contained in \textit{temp1}. These two Web tuples are combined together and inserted as a single Web tuple (represented by the fourth Web tuple in Fig. 9). Similarly, the new and old versions of the Web tuples in \textit{temp2} are determined and inserted in \textit{updateTupleSet}. After step (22) in Fig. 11, the algorithm generates three sets, \textit{insertTupleSet}, \textit{deleteTupleSet}, and \textit{updateTupleSet} representing the Web tuples that contain all of the relevant nodes that are inserted, deleted, or updated.

### 6.2.3 Algorithm of Phase 4

Next, the algorithm proceeds to create the delta Web tables, i.e., \( W_{\Delta^+} \), \( W_{\Delta^-} \), and \( W_{\delta} \) from \textit{insertTupleSet}, \textit{deleteTupleSet}, and \textit{updateTupleSet} (Steps (23)-(25) in Fig. 11). We describe the construction of \( \Delta^+ \)-Web table here. As the construction of the \( \Delta^+ \) and \( \Delta^- \)-Web tables is straightforward, we do not discuss this in detail in this paper.

The algorithm first materializes each Web tuple in \textit{updateTupleSet} in the Web tuple pool of \( W_{\Delta^+} \). Then, it retrieves the node objects (old and new versions) from the table node pools of \( W_1 \) and \( W_2 \) for each node id in \textit{updateNodeSet} and materializes these nodes in the table node pool of \( W_{\Delta^+} \). Observe that, in the table node pool of \( W_{\Delta^+} \), we only materialize nodes which have undergone content modification. However, in Web tuple pool of \( W_{\Delta^+} \), we materialize the node ids of the joinable nodes in addition to the identifiers of the modified nodes. This is because each tuple in the Web tuple pool contains not only the old and new versions of the modified nodes but also how these nodes are related to other nodes which have remained unchanged during \( t_1 \) and \( t_2 \). Finally, the schema of the joined Web table and the outer joined Web table are manipulated to generate the schema of \( W_{\delta} \). As mentioned earlier, we do not discuss it in this paper.

The creation of \( \Delta^+ \) and \( \Delta^- \) Web tables are quite similar to that of \( \Delta^M \)-Web table. The only difference is that the tuples are created from \textit{insertTupleSet} and \textit{deleteTupleSet}, respectively, and the schemas of \( W_{\Delta^+} \) and \( W_{\Delta^-} \) are identical to the Web schema of \( W_1 \) or \( W_2 \).

### 7 Conclusions and Future Work

In this paper, we have formally defined the change detection problem in the Web warehouse context. To solve this problem, we have presented algorithms that are based on representing two versions of Web data as Web tables and manipulating these Web tables using a set of Web algebraic operators for detecting changes. We have represented the Web deltas in the form of delta Web tables. We have implemented algorithms for computing and representing changes in Web data relevant to a user’s query.

As ongoing work, we are addressing the following issues:
1. Analytical and empirical studies of the algorithms for generating the delta Web tables. We wish to perform experiments to evaluate the performance of the algorithms. We are also investigating the scalability issues in this context.

2. Currently, the delta Web tables contain tuples where only some of the nodes represent the insertion, deletion, or update operation during the transition. This is because we wish to show how these nodes are related to one another and to other nodes which have remained unchanged during the transition. Therefore, we need a mechanism to distinguish between the modified, new, or deleted nodes in each delta Web tables. We are currently building a data model over our warehouse data model to allow annotation on the affected nodes to represent these changes.

3. As we represent the Web deltas in the form of Web tables, these tables can be further manipulated using the existing set of Web operators and queries. We are designing and implementing a powerful query language for the change management system in the context of our Web warehouse.

4. We intend to implement a change notification mechanism in WHOWEDA similar to one proposed in [10]. We intend to support various subscription services such as allowing changes to be detected, queried, and reported whenever a user is interested.

5. Design an event-condition-action trigger language for WHOWEDA based on the ideas from the change detection system.

REFERENCES


