WUML: A Web Usage Manipulation Language For Querying Web Log Data*

Qingzhao Tan, Yiping Ke and Wilfred Ng

Department of Computer Science
The Hong Kong University of Science and Technology,
Hong Kong
{keyiping, ttqqzz, wilfred}@cs.ust.hk

Abstract. In this paper, we develop a novel Web Usage Manipulation Language (WUML) which is a declarative language for manipulating Web log data. We assume that a set of trails formed by users during the navigation process can be identified from Web log files. The trails are dually modelled as a transition graph and a navigation matrix with respect to the underlying Web topology. A WUML expression is executed by transforming it into Navigation Log Algebra (NLA), which consists of the sum, union, difference, intersection, projection, selection, power and grouping operators. As real navigation matrices are sparse, we perform a range of experiments to study the impact of using different matrix storage schemes on the performance of the NLA.

1 Introduction

The topology of a Web site is constructed according to the designers' conceptual view of the Web information. There may be a mismatch, however, between the users' behavior and the expectation of the designers. Therefore, we propose and develop a query language on Web log data. We assume that the Web usage information can be generated from log files through a cleaning process [11, 4]. Based on our earlier work in [11], we model a collection of user sessions on a given Web topology as a weighted directed graph, called a *transition graph*. A corresponding navigation matrix is then computed using knowledge of the underlying Web topology.

Herein, we extend the four basic operators of `sum`, `union`, `intersection` and `difference` from [11] on *valid navigation matrices* to a more comprehensive set of operators: `sum`, `union`, `intersection`, `difference`, `projection`, `selection`, `power` and `grouping`. We call these operators collectively the Navigation Log Algebra (the NLA). Navigation matrices generated from real Web log files are sparse. We therefore carry out a spectrum of experiments to study the performances of the NLA operators on synthetically generated navigation matrices. The results indicate that the storage schemes affect the performances differently.

To gain better insight regarding navigation behavior from the log data, we further develop a novel declarative language termed *Web Usage Manipulation*

* This work is supported in part by grants from the Research Grant Council of Hong Kong, Grant Nos DAG01/02.EG05 and HKUST6185/02E.

Language (*WUML*), which allows users to specify queries on navigation matrices. The WUML expressions are implemented as a sequence of NLA operations. Using WUML, a Web designer is able to better understand navigation details over the site structure based on the analysis of querying results. For example, the overall usage of the site can be generated by a WUML query that combines all user categories, while the deviation analysis can be generated by a WUML query that finds out the contrast between the designer's expectation and user navigation behaviors. WUML also enables the designers to view the overall performance of a set of closely-related pages using `grouping`.

Our Contributions. (1) We define the NLA on navigation matrices, which are a comprehensive set of operators of `sum`, `union`, `intersection`, `difference`, `projection`, `selection`, `power` and `grouping`. (2) We propose a validation algorithm called VALID, which is able to preserve the validity of the navigation matrix when executing some of the operations. Essentially, this is to avoid isolated sets of pages happening in NLA operations. (3) We develop a novel declarative language WUML on navigation matrices. WUML can be transformed into a corresponding sequence of operations. (4) We study three different storage schemes to deal with the sparse nature in navigation matrices. By carrying out a spectrum of experiments on synthetic Web log data, we clarify the effects of storage schemes on individual NLA operators.

Related Work. There has been a lot of research related to applying data mining techniques on the Web log data. A mass of Web usage mining tools [1, 2, 5, 14, 13, 7] have been developed to help the designers improve Web sites, attract visitors, or provide users with a personalized and adaptive service. Several mining languages, such as WUM's MINT [13] and WEBMINER's query language [7], are also proposed for these objectives. However, these languages are based on mining techniques for association rules and sequential patterns. Our WUML is developed to specify a query which is sufficiently expressive to query log data represented as a transition graph, or equivalently a navigation matrix.

The rest of this paper is organized as follows. In Section 2, we give preliminary definitions related to Web usage analysis. NLA for the navigation matrices is discussed in Section 3. We propose WUML and discuss the transformation of a WUML expression into an NLA expression in Section 4. In Section 5, three storage schemes for navigation matrices are introduced. A set of experimental results on the NLA using different storage schemes are analyzed in Section 6. Finally, we give our concluding remarks in Section 7.

2 Preliminaries

A Web topology W is defined as a directed graph, in which each node represents one Web page and each directed link represents the hyperlink between pages. A user session is a sequence of page requests from the same user such that no two consecutive requests are separated by more than X minutes. In a user session, two consecutive pages should have a link in the Web topology.

A transition graph is a weighted directed graph constructed from W by adding two special pages: the starting page S and the finishing page F. Given a

set of user sessions, we define the weight of the link from S to any other page as the number of times that the page is first requested. Similarly, the weight of the link from any page to F is the number of times that the page is last requested. As for links between any other two pages in W, called *internal links*, the weight is the number of times that two pages appear as consecutive pages in the set of user sessions. We dually model a transition graph as a *navigation matrix* [11]. A navigation matrix is defined as the *adjacency matrix* of a transition graph. A one-to-one correspondence exists between transition graphs and navigation matrices.

Now, we discuss the notion of validity of a transition graph. A node in a transition graph is said to be *balanced* if the total weights of its in-links equal to the total weights of its out-links. And the *node degree* is the total weights of in-links and out-links. A transition graph is said to be *valid* if it satisfies the following four conditions. (1) In-degree of S, out-degree of F and the link weight from S to F are zero; (2) Every internal link having non-zero weight is also a link in the Web topology, (Note that this excludes self-looping in the graph.); (3) Every node except S and F should be balanced; (4) Every node which has non-zero degree should be reachable from S.

The validity of the navigation matrix is equivalent to the validity of the transition graph. That is, a navigation matrix is said to be valid if and only if its corresponding transition graph is valid. As we can see in the subsequent sections, after execution of some NLA operators, such as `difference`, `intersection` and `power`, the output navigation matrix may not be valid. There may be nodes with non-zero degree which cannot be reached from S. However, the validity in outcomes is essential since it ensures that the operations continue in a procedural manner. It is thus necessary to guarantee the output navigation matrix is valid.

We outline an algorithm, VALID, as shown in Algorithm 1, which employs DFS strategy, to keep the validity of a navigation matrix. The input of the algorithm is a navigation matrix whose nodes are balanced. Using VALID, connected components of the transition graph can be found. If there is more than one component, we add a link from S to the root of that component and also a link from that root to F to keep it balanced. The output matrix is then valid since all pages with non-zero degree are reachable from S.

Note that VALID needs three extra arrays (namely, color, parent and tag), each with size of $n+2$, the space complexity is $O(n)$. The time determining steps are the two nested loops over $n+2$ in Lines 1 and 3 in Algorithm 1, which takes $O(n^2)$ time. As for the execution of the DFS-VISIT procedure, it takes $O(E)$ time, where E is the number of edges in the transition graph corresponding to the input navigation matrix. In the worst case, E equals to $(n+2)^2$. To conclude, the time complexity for running VALID is $O(n^2)$.

3 Navigation Log Algebra

In this section, we define the Navigation Log Algebra (the NLA). For more illustrated examples on using some operations, the readers may refer to [11].

Sum. The `sum` of two navigation matrices M_1 and M_2, denoted as $M_1 + M_2$, is defined as a navigation matrix M_3 such that, for all $i, j \in \{0, \ldots, n+1\}$,

Algorithm 1 VALID Algorithm

Input: Matrix M with $n+2$ dimensions
VALID(M)

```
1.  FOR all 0 ≤ i ≤ n + 1                    //Initialization
2.      DO color[i] := WHITE; parent[i] := NIL; tag[i] := 0;
3.          FOR all 0 ≤ j ≤ n + 1
4.              DO IF M[i][j] != 0
5.                  THEN tag[i] := 1; break;//tag indicates pages with non-zero degree
6.  tag[0] := 1;                  //Make sure DFS starts from page S
7.  FOR all 0 ≤ i ≤ n + 1                    //DFS: Find connected components
8.      DO IF tag[i] = 1 and color[i] = WHITE
9.          THEN DFS-VISIT(i);
10. FOR all 1 ≤ i ≤ n
11.     DO IF tag[i] = 1 and parent[i] = NIL          //Ensure validity by adding
12.         THEN M[0][i] +=  1; M[i][n + 1] += 1;  //links from S and links to F
DFS-VISIT(i)          //Recursively search to form connected components
1. color[i] := GRAY
2. FOR all 0 ≤ j ≤ n + 1
3.     DO IF M[i][j] != 0 and color[j] = WHITE
4.         THEN parent[j] := i; DFS-VISIT(j);
```

Output: The valid matrix of M

$(a_{ij})_3 = (a_{ij})_1 + (a_{ij})_2$. Actually, `sum` is exactly the generic sum of two matrices. Trivially, the outcome M_3 remains to be valid.

Union. The `union` of two navigation matrices M_1 and M_2, denoted as $M_1 \bigcup M_2$, is defined as a navigation matrix M_3 such that, for all $i, j \in \{1, \ldots, n\}$, $(a_{ij})_3 = max((a_{ij})_1, (a_{ij})_2)$; $(a_{0j})_3 = max((a_{0j})_1, (a_{0j})_2) + max(0, \sum_{k=1}^{n+1} max((a_{jk})_1, (a_{jk})_2) - \sum_{k=0}^{n} max((a_{kj})_1, (a_{kj})_2))$; $(a_{i(n+1)})_3 = max((a_{i(n+1)})_1, (a_{i(n+1)})_2) + max(0, \sum_{k=0}^{n} max((a_{ki})_1, (a_{ki})_2) - \sum_{k=1}^{n+1} max((a_{ik})_1, (a_{ik})_2))$; and all other elements are zero. For `union`, we do not need to use VALID because the *max* function used in `union` is able to maintain the reachability of nodes from S.

Difference. The `difference` of two navigation matrices M_1 and M_2, denoted as $M_1 - M_2$, is defined as a navigation matrix M_3 such that, for all $i, j \in \{1, \ldots, n\}$, $(a_{ij})_3 = max(0, ((a_{ij})_1 - (a_{ij})_2))$; $(a_{0j})_3 = max(0, ((a_{0j})_1 - (a_{0j})_2)) + max(0, \sum_{k=1}^{n+1} max(0, ((a_{jk})_1 - (a_{jk})_2)) - \sum_{k=0}^{n} max(0, ((a_{kj})_1 - (a_{kj})_2)))$; $(a_{i(n+1)})_3 = max(0, ((a_{i(n+1)})_1 - (a_{i(n+1)})_2)) + max(0, \sum_{k=0}^{n} max(0, ((a_{ki})_1 - (a_{ki})_2)) - \sum_{k=1}^{n+1} max(0, ((a_{ik})_1 - (a_{ik})_2)))$; and all other elements are zero. As the result may be invalid, we run VALID after executing the above operation.

Intersection. The `intersection` of two navigation matrices M_1 and M_2, denoted as $M_1 \bigcap M_2$, is defined as a navigation matrix M_3 such that, for all $i, j \in \{1, \ldots, n\}$,$(a_{ij})_3 = min((a_{ij})_1, (a_{ij})_2)$; $(a_{0j})_3 = min((a_{0j})_1, (a_{0j})_2) + max(0, \sum_{k=1}^{n+1} min((a_{jk})_1, (a_{jk})_2) - \sum_{k=0}^{n} min((a_{kj})_1, (a_{kj})_2))$; $(a_{i(n+1)})_3 = min((a_{i(n+1)})_1, (a_{i(n+1)})_2) + max(0, \sum_{k=0}^{n} min((a_{ki})_1, (a_{ki})_2) - \sum_{k=1}^{n+1} min((a_{ik})_1,$

$(a_{ik})_2)$); and all other elements are zero. We run the VALID algorithm on the intermediate answer from executing `intersection`.

Projection. The `projection` of one navigation matrix M_1, denoted as $\prod_P(M_1)$ where P is a set of Web pages, is defined as a navigation matrix M_3 such that, for all $P_i, P_j \in P$, $(a_{ij})_3 = (a_{ij})_1$; $(a_{0j})_3 = (a_{0j})_1 + \sum_{k=1,\ldots,n;P_k \notin P}(a_{kj})_1$; $(a_{i(n+1)})_3 = (a_{i(n+1)})_1 + \sum_{k=1,\ldots,n;P_k \notin P}(a_{ik})_1$; and all other elements are zero. `Projection` does not need the VALID algorithm.

Selection. The `selection` of one navigation matrix M_1, denoted as $\sigma_{\theta x}(M_1)$, where θ is a comparator such as >, <, >=, <=, or =, and x is a positive integer, is defined as a navigation matrix M_3 such that, for all $i, j \in \{1, \ldots, n\}$, $if (a_{ij})_1 \theta x, (a_{ij})_3 = (a_{ij})_1; (a_{0j})_3 = (a_{0j})_1 + \sum_{k=1,\ldots,n;(a_{kj})_1 \bar{\theta} x}(a_{kj})_1; (a_{i(n+1)})_3 = (a_{i(n+1)})_1 + \sum_{k=1,\ldots,n;(a_{ik})_1 \bar{\theta} x}(a_{ik})_1$; and all other elements are zero. The notation $\bar{\theta}$ denotes the complement of θ (e.g. $\bar{\theta}$ is "<=" when θ is ">"). The output is already valid, so it does not need to be processed through VALID.

Power. The `power` of one navigation matrix M_1, denoted as $(M_1)^x$, where x is a non-negative integer, is defined as a navigation matrix M_3 such that $M_3 = 0_{(n+2)\times(n+2)}$ if $x = 0$; $M_3 = M_1$ if $x = 1$; and $M_3 = M_1 \cdot (M_1)^{x-1}$ if $x \geq 2$. Herein, the operator "$\cdot$" denotes the multiplication of two matrices. The result M_3 may not be valid, since we need to ignore the possible non-zero values in its diagonal and at $(a_{0(n+1)})_3$. We should maintain the pages balance and run VALID. The semantics for non-zero entry $(a_{ij})_3$ in $M_3 = (M_1)^x$ is that there is a trail from P_i to P_j with the length of x in M_1. If $(a_{ij})_3$ is large, it indicates that many users have traversed from P_i to P_j. If there is no link from P_i to P_j in the Web topology, the designer may add this link to facilitate better navigation.

Grouping. The `grouping` of a navigation matrix M_1, denoted as $G_P(M_1)$, where P is a set of pages in W, returns a navigation matrix M_3 which is an aggregated view of M_1. `Grouping` groups all the pages in P as a single page in M_3 by ignoring the links within the pages in P and adding up the weight of links from and to the outside pages, respectively. By grouping a set of pages which are closely related in semantics, we are able to understand the navigation in terms of *information units* [9]. Our approach is to view `grouping` as an aggregation of log information. Therefore, we do not define ungrouping here, since ungrouping introduces uncertainty in log information which needs further study.

The following properties follow from the definitions of the NLA operators. We will see later these properties pave the way for choosing an efficient execution plan for a given WUML expression. We only state the following properties where we assume $\delta_1 \in \{+, \cup, \cap\}$ and $\delta_2 \in \{\cup, \cap\}$, since the proof is straightforward.

Associative Property. $(M_1\ \delta_1\ M_2)\ \delta_1\ M_3 = M_1\ \delta_1\ (M_2\ \delta_1\ M_3)$.

Commutative Property. $M_1\ \delta_1\ M_2 = M_2\ \delta_1\ M_1$; $\Pi_P(\sigma_{\theta x}(M)) = \sigma_{\theta x}(\Pi_P(M))$

Distributive Property. $\Pi_P(M_1\ \delta_1\ M_2) = \Pi_P(M_1)\ \delta_1\ \Pi_P(M_2)$; $G_P(M_1\ \delta_1\ M_2) = G_P(M_1)\ \delta_1\ G_P(M_2)$; and $\sigma_{\theta x}(M_1\ \delta_2\ M_2) = \sigma_{\theta x}(M_1)\ \delta_2\ \sigma_{\theta x}(M_2)$.

4 The Web Usage Manipulation Language

We now introduce a declarative language on navigation matrices termed the Web Usage Manipulation Language (the WUML). A WUML expression is executed via the NLA operators defined in Section 3, which shares the same principle

of translating a SQL expression into a sequence of relational algebra operations. We now define the WUML syntax in Backus Naur Form (BNF):

<query> :: = <selectClause><fromClause>[<conditionClause>][<groupClause>]
<selectClause> :: = SELECT <pageList>
<queryList> :: = query [, query...]
<fromClause> :: = FROM <matrixIdentifier> | FROM <operator> <matrixList>
<conditionClause> :: = WHERE LINKWT <compOp> integer
<groupClause> :: = GROUP BY <pageList>
<pageList> :: = pageIdentifier [, pageIdentifier...]|∗
<matrixList> :: = matrixIdentifier [, matrixIdentifier...]
<operator> :: = SUM|UNION|DIFF|INTERSECT|POWER integer
<compOp> :: = | <= | >= | < | =

There are four main clauses in a query expression: the *select*, *from*, *condition*, and *group* . Among them the *select* and the *from* clauses are compulsory, while the *condition* and the *group* clauses are optional. Similar to SQL, WUML is a simple declarative language but is powerful enough to express query on the log information stored as navigation matrices.

We execute a WUML expression by translating it into a sequence of NLA operations using Algorithm 2. Suppose $\mathcal{P}$ is a set of l Web pages which appears in the select clause, where $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$, $\mathcal{M}$ is a set of m matrices which appears in the from clause, where $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, P_G is a set of n Web pages which appears in the group clause, where $P_G = \{P_1, P_2, \ldots, P_n\}$, $OPER \in \{\epsilon, SUM, UNION, DIFF, INTERSECT, POWER\ \alpha\ \}$, x and α are two non-negative integers. Note that the input WUML query expression is assumed to be syntactically valid. If $OPER = \epsilon$ or $POWER\ \alpha$, then $m = 1$. Figure 1 depicts a query tree which illustrates the basic idea.

Algorithm 2 TRANS Algorithm

Input: A WUML expression q
LET q = <selectClause><fromClause>[<conditionClause>][<groupClause>]
<selectClause> := "SELECT $\mathcal{P}$ | ∗"
<fromClause> := "FROM $OPER\ \mathcal{M}$"
<conditionClause> := "WHERE LINKWT θx"
<groupClause> := "GROUP BY P_G"
Procedure:
Step 1 : For the fromClause, **CASE** $OPER$ **OF**:
$\epsilon : TEMP :=$ "M_1"
$SUM : TEMP :=$ "$M_1 + M_2 + \cdots + M_m$"
$UNION : TEMP :=$ "$M_1 \bigcup M_2 \bigcup \cdots \bigcup M_m$"
$DIFF : TEMP :=$ "$M_1 - M_2 - \cdots - M_m$"
$INTERSECT : TEMP :=$ "$M_1 \bigcap M_2 \bigcap \cdots \bigcap M_m$"
$POWER\ \alpha : TEMP :=$ "$(M_1)^{\alpha}$"
Step 2 : For the selectClause:
IF "∗" **THEN** $TEMP := TEMP$
ELSE $TEMP :=$ "$\Pi_{\mathcal{P}}(TEMP)$"
Step 3 : **IF** there is a whereClause **THEN** $TEMP :=$ "$\sigma_{\theta x}(TEMP)$"
Step 4 : **IF** there is a groupClause **THEN** $TEMP :=$ "$G_{P_G}(TEMP)$"
Output: $TEMP$ expression

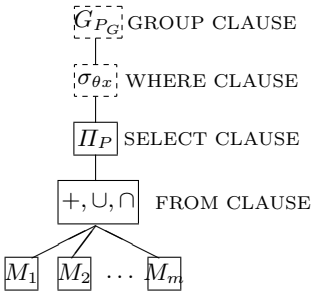


Fig. 1. WUML query tree

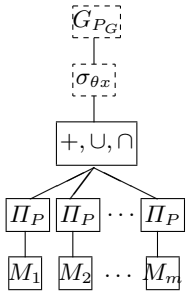


Fig. 2. Optimized WUML query tree

Now, we present a set of examples, which illustrates the usage of the WUML expressions and the translation into the corresponding sequence of NLA operations. Let M, M_1, M_2 and M_3 be navigation matrices.

Q_1: We want to know how frequently the pages P_1 and P_2 were visited.

WUML expression: SELECT P_1, P_2 FROM SUM M_1, M_2.

NLA operation: $\Pi_{\{P_1,P_2\}}(M_1 + M_2)$.

Q_2: We want to find out the essential difference of preferences between the two groups of users in M_1 and M_2. We consider those links having the weight > 3.

WUML expression: SELECT * FROM DIFF M_1, M_2 WHERE LINKWT > 3. *NLA operation*: $\sigma_{>3}(M_1 - M_2)$.

Q_3: We want to get the navigation details in an *information unit* [9] consisting of the pages P_1, P_2, and P_3. We may gain insight to decide whether it is better to combine these three Web pages or not. So we consider them as a group.

WUML expression: SELECT * FROM SUM M_1, M_2 GROUP BY P_1, P_2, P_3.

NLA operation: $G_{\{P_1,P_2,P_3\}}(M_1 + M_2)$.

Q_4: We want to know whether some pages were visited by users after 3 clicks. If they were seldom visited or lately visited in a user session, we may decide to remove or update them to make them more popular.

WUML expression: SELECT P_1, P_2, P_3 FROM POWER 3 M.

NLA operation: $\Pi_{\{P_1,P_2,P_3\}}(M)^3$.

Q_5: Now we want to get the specific information of a particular set of Web pages.

WUML expression: SELECT P_1, P_2, P_3, P_4, P_5 FROM INTERSECT M_1, M_2, M_3 WHERE LINKWT > 6.

NLA operation: $\sigma_{>6}(\Pi_{\{P_1,P_2,P_3,P_4,P_5\}}(M_1 \bigcap M_2 \bigcap M_3))$.

Let us again consider the query, Q_5. We will see in Section 6 that the running time of NLA operators are proportional to the number of non-zero elements in the executed matrix. Therefore, the optimal plan is to first execute the NLA operators which can minimize the number of non-zero elements in the matrix. For the sake of efficiency, the `projection` (Π) should be executed as early as possible. So a better NLA execution plan of Q_5 can be obtained as follows:

$Q_6 : \sigma_{>6}(\Pi_{\{P_1,P_2,P_3,P_4,P_5\}}(M_1) \bigcap \Pi_{\{P_1,P_2,P_3,P_4,P_5\}}(M_2) \bigcap \Pi_{\{P_1,P_2,P_3,P_4,P_5\}}(M_3))$.

We now summarize some optimization rules as depicted in Figure 2. First, the `projection` should be done as early as possible since it can eliminate some non-zero elements. Note that `projection` is not distributive under `difference` and `power`. Second, since the `selection` is not distributive under some binary operators such as `difference`, we do not change the execution order. Finally,

the `grouping` creates a view different from the underlying Web topology. Therefore, it should be done at the last step except some operators taking another navigation matrix whose structure is the same as the grouped one. Note that these rules are simple heuristics to sort NLA operations. We still need to find out a more systematic way to generate an optimized execution plan for a given WUML expression.

5 Storage Schemes for Navigation Matrices

As the navigation matrices generated from the Web log files are usually sparse, the storage scheme of a matrix greatly affects the performance of WUML. In this section we introduce three storage schemes, COO, CSR, and CSC, to study their impacts on the NLA operations.

In literature, the technique of storing sparse matrices has been intensively studied [3, 8]. In our WUML environment, we store the navigation matrix as three separate parts: the first row (i.e. the weights of the links starting from S), the last column(i.e. the weights of the links ending in F) and the square matrix despite the rows and columns of S and F. We employ two vectors, S_{vector} and F_{vector}, which contains an array for the non-zero values in the vector as well as the corresponding index, to store the first row and the last column. Table 2 and 3 show examples using the matrix given in Figure 1. As for the third part of the navigation matrix, we implement the storage schemes proposed in [8]. We illustrate the schemes using the matrix in Table 1.

	S	P_1	P_2	P_3	P_4	F
S	0	3	1	0	0	0
P_1	0	0	2	2	0	0
P_2	0	1	0	0	0	3
P_3	0	0	0	0	3	0
P_4	0	0	1	1	0	1
F	0	0	0	0	0	0

Table 1. Navigation Matrix

Non-zero	3	1
Column	1	2

Table 2. S_{vector}

Non-zero	3	1
Column	2	4

Table 3. F_{vector}

Non-zero	2	2	1	3	1	1
Column	2	3	1	4	2	3
Row	1	1	2	3	4	4

Table 4. COO Scheme

The Coordinate (COO) storage scheme is the most straightforward structure to represent a sparse matrix. As illustrated in Table 4, it records each nonzero entry together with its column and row index in three arrays. The Non-zero array holds the non-zero entries in row-first order. Similar to COO, the Compressed Sparse Row (CSR) storage scheme also consists of three arrays. CSR differs from COO in the Compressed Row array. In CSR, Compressed Row is an array of size n, where n is the number of pages in Web topology. It stores the location of the first non-zero entry in that row. Table 5 shows the structure of CSR. The Compressed Sparse Column (CSC) storage scheme, as shown in Table 6, is similar to CSR. It has three arrays: Nonzero array to hold the non-zero values in column-first order, Compressed Column array to hold the location of the first non-zero entry of that column, Row array for the row indices. CSC is the *transpose* of CSR. There are also other sparse matrix storage schemes, such as Compressed Diagonal Storage (CDS) and Jagged Diagonal Storage (JDS) [12]. However, they are used for storing a *banded sparse matrix*. In reality, the navigation matrix should not be banded. Therefore, these schemes are not studied in our experiments.

Non-zero	2	2	1	3	1	1
Column	2	3	1	4	2	3
Compressed Row	0	2	3	4		

Table 5. CSR Scheme

Non-zero	1	2	1	2	1	3
Compressed Column	0	1	3	5		
Row	2	1	4	1	4	3

Table 6. CSC Scheme

6 Experimental Results and Analysis

We carry out a set of experiments to compare the performances of the three storage schemes introduced in Section 5. We also study the usability and efficiency of WUML on different data sets. The data set we used in the experiments is a set of synthetic Web logs on different Web topology, which are generated by a log generator designed in [10]. The parameters used to generate the log files are described in Table 7. Among these four parameters, PageNum and MeanLink

LogSize	The number of log records in a log file.
UserNum	The number of users traversing the Web site.
PageNum	The number of pages in the Web topology.
MeanLink	The average number of links per page in the Web topology.

Table 7. Parameters for Data Set

are dependent on the underlying Web topology while the other two are not. These experiments are run on Pentium 4, 2.5GHz, and 1G of RAM machine configuration.

6.1 Construction Time of Storage Schemes

We choose three data sets: $D_1 = (2500, 1500, 1500, 3)$, $D_2 = (5000, 3000, 3000, 5)$ and $D_3 = (10000, 6000, 6000, 10)$, in which the components represent the parameters LogSize, UserNum, PageNum and MeanLink, respectively. Then we construct three storage schemes based on the generated log files from D_1 to D_3. Our measurement of the system response time includes I/O processing time and CPU processing time. As shown in Figure 3, the response time grows significantly as the parameters increase. Since most of the time is consumed in reading the log files, the construction time for the same given data set varies slightly among the three storage schemes. But it still takes more time to construct COO than the other two schemes, since there is no compressed array for COO. CSC needs more time than CSR because the storage order in CSC is column-first while reading in the file is in row-first order.

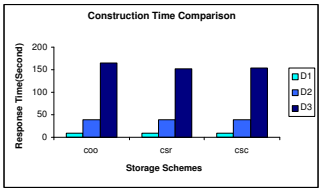


Fig. 3. Construction Time

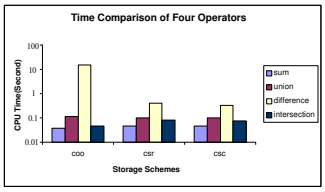


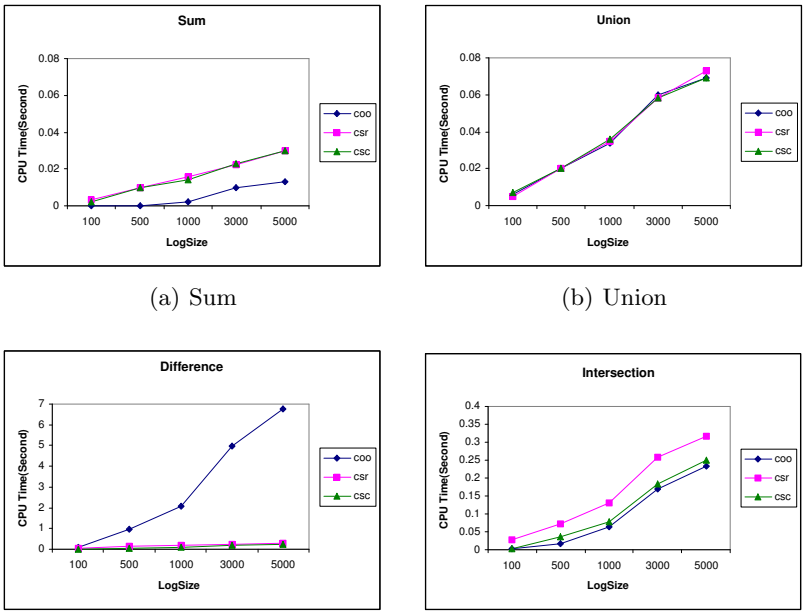
Fig. 4. Running Four Operators

6.2 Processing Time of Binary Operators

We present the CPU processing time results of four binary operators: `sum`, `union`, `difference` and `intersection`. Each time we tune one of the four parameters to see how the processing time changes on COO, CSR and CSC storage schemes. For each parameter, we carry out experiments on ten different sets of Web logs. We first compare the processing time of each single operator under different storage schemes. Then we present the processing time of each storage scheme under different operators.

Tuning LogSize. We set UserNum and PageNum to be 3000, MeanLink to be 5. The results are shown in Figure 5. When LogSize increases, the processing time of the same operator on each storage scheme also increases. The reason is that the number of non-zero elements in the navigation matrix grows with the increase of LogSize, and therefore it needs more time to do the operations.

Tuning PageNum. We set LogSize to be 5000, UserNum to be 3000, and MeanLink to be 5. The results are presented in Figure 7. With the growth of PageNum, the CPU time for each operator on specified storage scheme grows quickly. It is because PageNum is a significant parameter when constructing the navigation matrix. The more pages in the Web site, the larger dimension of a navigation matrix, and consequently, the more time needed to construct the navigation matrix.

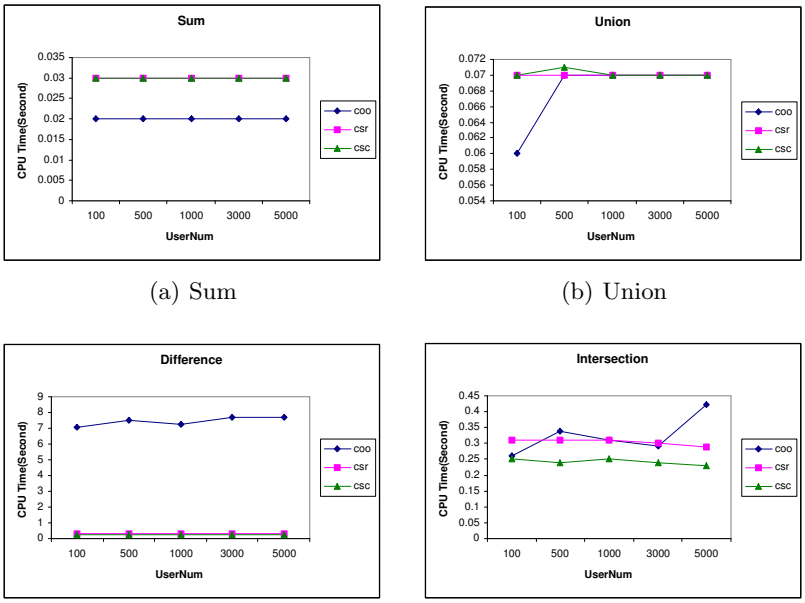


(a) Sum (b) Union

(c) Difference (d) Intersection

Fig. 5. The CPU time by tuning LogSize

Tuning UseNum. Figure 6 shows the results when LogSize is 5000, PageNum is 3000 and MeanLink is 5. We observe that the processing time remains almost unchanged when UserNum grows. The main reason is that, although different user may have different behavior when traversing the Web site, the number of non-zero elements in navigation matrix is roughly the same due to the fixed LogSize.



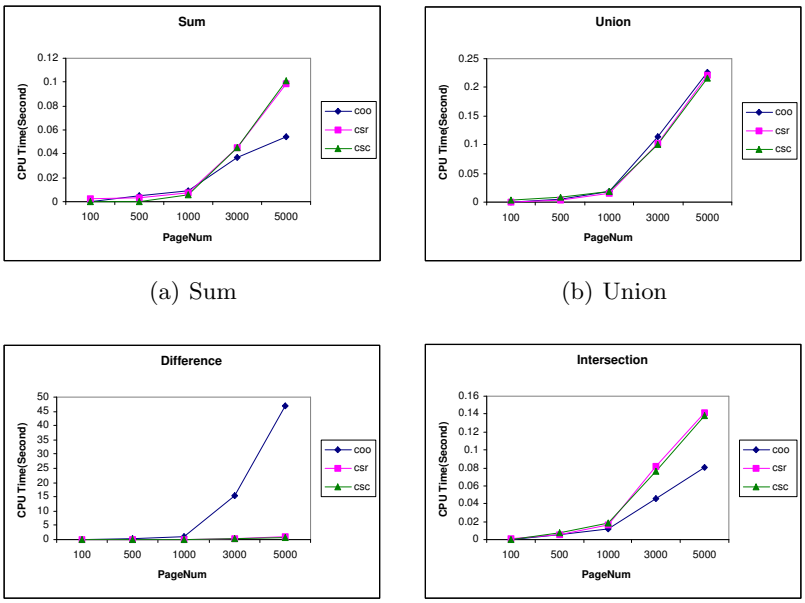
(a) Sum (b) Union

(c) Difference (d) Intersection

Fig. 6. The CPU time by tuning UserNum

Tuning MeanLink. We use the log files with LogSize of 5000, UserNum of 3000 and PageNum of 3000. The results are shown in Figure 8 which indicates that, with the increase of MeanLink, the processing time decreases.

Note that when processing the `sum`, COO always outperforms others, while CSR and CSC perform almost the same (see Figures 5(a), 6(a), 7(a) and 8(a)). The similar phenomenon can be observed in Figures 5(d), 6(d), 7(d) and 8(d) for processing the `intersection`. As shown in Figures 5(b), 6(b), 7(b) and 8(b), we see that the processing time for `union` on three storage schemes has no significant difference. Finally, from Figures 5(c), 6(c), 7(c) and 8(c), the `difference` is opposite to `sum`: the performances of CSR and CSC are much better than COO when processing `difference`. Note also that from Figure 4, `difference` requires the most processing time, and `sum` needs the least. The Web logs used in this set of experiments has 5000 LogSize, 1000 UserNum, 3000 PageNum and 5 MeanLink. The reason for this result is as follows. As we have mentioned, when defining `sum`, we do not need to check the balance of Web pages and the valid-



(a) Sum (b) Union

(c) Difference (d) Intersection

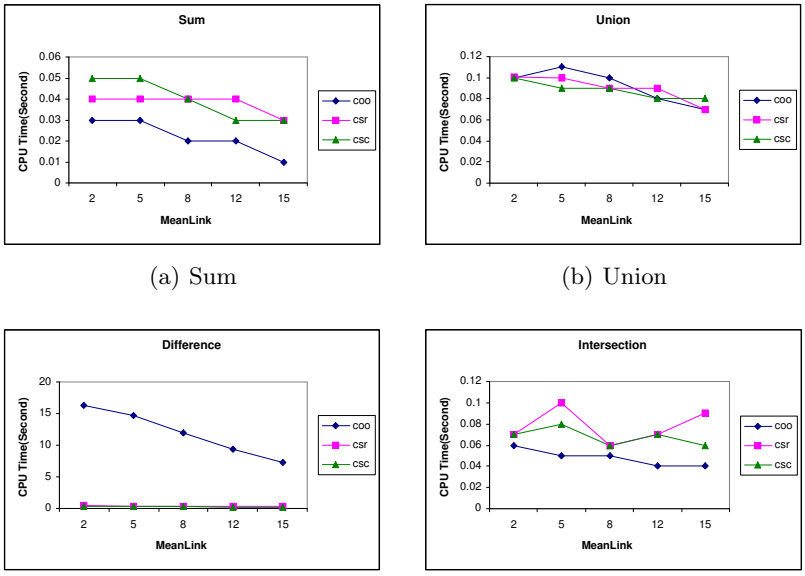
Fig. 7. The CPU time by tuning PageNum

ity of the navigation matrix. Therefore, it takes the least time to perform `sum`. For `union`, we only need to check the balance of Web pages without checking the validity of the output matrix. But for `difference` and `intersection`, we have to check both the page balance and matrix validity, which is rather time-consuming. It can be found that `intersection` does not need much time. This is because there are very few non-zero elements in the output matrix.

6.3 Performance of Unary Operators

Power. We study the performance of `power` using log files with 5000 LogSize, 3000 UserNum, 5 MeanLink, $(100, 500, 1000)$ PageNum. Each matrix multiplies twice (i.e. power $= 2$). We show the result in Figure 9. COO performs much worse than CSR and CSC in this situation. We also see that `power` is a rather time-consuming operator.

Projection and Selection. Since `projection` and `selection` are commutative, we study the time cost by swapping the two operators on the same navigation matrix. The matrix is constructed on a log file with 5000 LogSize, 5000 PageNum, 3000 UserNum and 5 MeanLink. From Figure 10, we conclude that doing `projection` before `selection` is more efficient than doing `selection` and then `projection`. According to this result, we can do some optimization when interpreting some queries. Moreover, COO outperforms CSR and CSC.



(a) Sum (b) Union

(c) Difference (d) Intersection

Fig. 8. The CPU time by tuning MeanLink

From the experimental results shown above, we have the following observations. First, from construction point of view, CSR is the best. Second, COO is the best for `sum` and `intersection`. Third, CSR and CSC perform almost the same for `difference` and `power`, and greatly outperform COO. Finally, COO, CSR and CSC perform the same for `union`. Taking these observations into consideration, CSR is the best for our WUML expressions. Although COO performs better in `sum` and `intersection`, it needs too much time for `difference` which is intolerant when the user issues a query associated to `difference`. Although the performance of CSC is the same as CSR with respect to the operations, CSC needs more time to be constructed. We also observe that the time growth for each operator is linear to the growth of parameters, which indicates that the usability and scalability of WUML is acceptable in practice.

7 Concluding Remarks

We presented NLA which consists of a set of operators on navigation matrices and proposed an efficient algorithm VALID ($O(n)$ space and $O(n^2)$ time complexities) to ensure the validity of an output matrix by NLA operators. Within NLA, we develop a query language WUML and study the mapping between the WUML statements and NLA expressions. To choose an efficient storage scheme for the sparse navigation matrix, we carried out a set of experiments on different

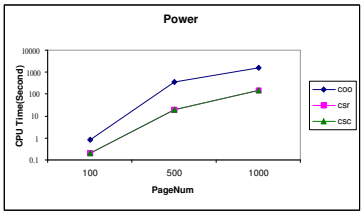


Fig. 9. Power (in log scale)

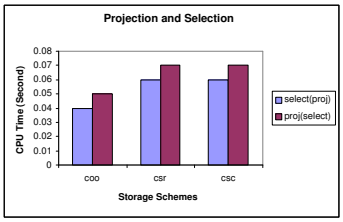


Fig. 10. Projection and Selection

synthetic Web log data sets, which are generated by tuning different parameters such as the number of pages, the number of mean links and the number of users. By the experimental results on three storage schemes of COO, CSC and CSR, we can see that the CSR scheme is relatively efficient for NLA. As for future work, we plan to develop a full-fledged WUML system to preform both analyzing and mining the real Web log data sets. We are also studying a more complete set of optimization heuristic rules for the NLA operators in order to generate a better execution plan for an input WUML expression.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB*, 1994.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of ICDE*, 1995.
3. Nawaaz Ahmed et al. A framework for sparse matrix code synthesis from high-level specifications. In *Proc. of the 2000 ACM/IEEE Conf. on Supercomputing*, 2000.
4. M. Baglioni et al. Preprocessing and mining web log data for web personalization. *The 8th Italian Conf. on AI*, 2829:237–249, September 2003.
5. B. Berendt and M. Spiliopoulou. Analyzing navigation behavior in web sites integrating multiple information systems. *The VLDB Journal*, 9(1), 2000.
6. L. D. Catledge and J. E. Pitkow. Characterizing browsing strategies in the world wide web. *Journal of Artificial Intelligence Research*, 27(6):1065–1073, 1995.
7. Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Web mining: Information and pattern discovery on the world wide web. *ICTAI*, 1997.
8. N. Goharian, A. Jain, and Q. Sun. Comparative analysis of sparse matrix algorithms for information retrieval. *Journal of Sys., Cyb. and Inf.*, 1(1), 2003.
9. Wen-Syan Li et al. Retrieving and organizing web pages by information unit. In *Proc. of WWW*, pages 230–244, 2001.
10. W. Lou. *loggen*: A generic random log generator: Design and implementation. Technical report, CS Department, HKUST, December 2001.
11. Wilfred Ng. Capturing the semantics of web log data by navigation matrices. *Semantic Issues in E-Commerce Systems. Kluwer Academic*, pages 155–170, 2003.
12. Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. of Sci. Stat. Comput.*, 10:1200–1232, 1989.
13. M. Spiliopoulou and L.C. Faulstich. WUM: A web utilization miner. *In Proc. of EDBT Workshop, WebDB98*, 1998.
14. J. Srivastava et al. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):12–23, 2000.