CHAPTER 4

PREPROCESSING ALGORITHMS

The first step of the proposed next web page prediction system is preprocessing, where the main focus is to retain only useful data from the raw web log. Due to large amount of irrelevant information in the web log, the original log cannot be directly used in the web log mining procedure, hence in the preprocessing phase, raw Web logs need to be cleaned, analyzed and converted for further use.

According to Srivastava *et al.* (2000) preprocessing "consists of converting the usage, content and structure information contained in the various available data sources into the data abstractions necessary for pattern discovery". Preprocessing of web data is essential in order to make it suitable for mining. It is identified as one of the key issues for web mining.

The purpose of preprocessing is to transit various input such as content, structure and usage information into the format which data mining algorithms can handle easily (Han *et al.*, 2001). A significant amount of work has been done in this area for web usage data, including user identification and session creation (Cooley *et al.*, 1999), robot detection and filtering (Tan and Kumar 2002) and extracting usage path patterns (Spiliopoulou 1999).

Cooley's Ph.D. dissertation (Cooley, 2000) provides a comprehensive overview of the work in web usage data preprocessing. Preprocessing of web structure data, especially link information, has been carried out for some applications, the most notable being Google style web search (Brin and Page, 1998). An up-to-date survey of structure preprocessing is provided by Desikan *et al.* (2002).

Phase I of the study performs preprocessing in five steps. They are listed below and the methods used in each step are described in this chapter.

(i) Cleaning,
(ii) User identification,
(iii) Session identification,
(iv) Formatting, and
(v) Clustering.

4.1. CLEANING WEB LOG DATA

In the first step, that is, the task of cleaning raw web log data is considered. The data removed during cleaning are not required for user navigation and hence can be deleted safely from the log file. This step carries out the following tasks:-

(i) Removal of unwanted and redundant data,
(ii) Removal of non-human accesses, and
(iii) Removal of erroneous references.

Examples of unwanted data include requests for images, javascripts, flash animations, video, etc. If the filename has gif, jpg, JPEG, CSS and so on, they are pruned out from the web log file. Redundant data are records having similar values in each attribute of the record. Example of such data includes entries made by web crawlers and Spider accesses (tools that scan website to extract its content).

Search engines normally use network robots to crawl through the web pages to collect information. The number of records created by these robots in a log file is extremely high and has a negative impact while discovering navigation pattern. This problem is solved in this paper by identifying the robot entries first before segmenting the user groups into potential and not-potential users. According to Yu *et al.* (2005) Pp. 55-59, entries in web log made by network robots can be identified by their IP address and agents. But this might require

knowledge on all type of agents and search engines, which is difficult to obtain. An alternative way is to study the robots.txt file (located at the website's root directory), as a network robot must always read this file before accessing the website. This is because the robots.txt has the access details of the website and each robot is request to know its access right before scrawling. But this cannot be always relied on since compliance to robot exclusion standard is voluntary and most of the robots do not follow the proposed standard. Thus, to delete robot entries, the following procedure is used.

1. Detect and remove all entries which has accessed robots.txt file
2. Detect and remove all entries with visiting time of access as midnight (commonly used as the network activity at that time is light)
3. Remove entry when access mode is HEAD instead of GET or POST
4. Compute browsing speed and remove all entries whose speed less than two seconds. The browsing speed is calculated as the number of viewed pages / session time.

In general, all entries having robots.txt and having time less than 2 seconds are considered as non-human accesses. As IP address is important for user identification, all entries with blank IP addresses are also removed. Further, all entries having more a depth level more than five are also removed, as they do not provide any additional information during prediction.

Erroneous references are failed page requests due to client / server errors. These entries are identified using the status attributes in the web log data. In practice, the status attribute consists of 3-digit number, where the first digit indicates the class of status code. The status code is used to represent the status of the web page request in a session. There are five classes of status code:-

1. Informational (1xx Series),
2. Success (2xx Series),
3. Redirect (3xx Series),
4. Failure (4xx Series), and
5. Server Error (SOD Series).

Evidently, all entries in the second class are important for web usage mining and are retained, while removing the rest of the record having status code entries in 1xx, 3xx, 4xx and SOD series.

4.2. USER IDENTIFICATION

The second step is the user identification step, consists of several subtasks that identify cookies, identd, through IP address and username. Cookies are defined as data sent by the server to the client; data locally stored in cookies and is sent to the server with each request (WCA, 2010). Successful user identification through cookies was done by Eirinaki and Vazirgiannis (2003) and Huysmans *et al.* (2003). The disadvantages of using cookies for user identification are twofold:-

(i) Some users block cookies, thus their information is not stored by the server, and

(ii) Users, for want of disk space, often delete cookies.

Identd is a protocol defined in RFC 1413 (2010), which can be used to identify users. This is performed by connecting the users with a unique TCP connection. The problem with this is that users should configure this protocol, if not is ignored by server and are not recorded in the log file.

Another method is to use user names and this again poses a problem, as it is often empty, as indicated by a hyphen ('-') in web log file.

The fourth method is the identification of users through the IP address. This is the most frequently used method as it is simple, easy to capture and is never empty. This has a problem of same users with multiple IP addresses. In the present research work, session identification is used to solve this problem and the method of identification is given below.

- If two records have dissimilar IP address they are differentiated as two different users else if both IP address are similar then User agent field is verified,
- If the browser and operating system information in user agent field is dissimilar in two records then they are recognized as different users else if both are identical then referrer URL field is checked, and
- If URL in the referrer URL field in present record is not accessed before or if URL field is blank then it is considered as a new user.

4.3. SESSION IDENTIFICATION

The third step, Session Identification, is generally performed using session timeout value. Session timeout is a process that automatically prevents user access to a system or application after a period of inactivity. The purpose of timeouts is to lock out unauthorized users when a system is unattended or when someone forgets to log out of an application. Typical session timeouts are between 25-30 minutes (Catledge and Pitkow, 1995). However, as mentioned in Chapter 3 (Methodology), usage of time alone is not efficient for session construction. In this research, to accommodate this issue, four acyclic graph-based algorithms are analyzed. Details of these four algorithms are presented in this section.

A user session is defined as a sequence of requests made by a single user over a certain navigation period and a user may have a single or multiple sessions during a period of time. The objective of session identification is to segregate the

page accesses of each user into individual sessions. Reconstruction of precise user sessions from server access logs is a difficult task because the access log protocol (HTTP protocol) is status less and connectionless. There are two simple methods for session identification. One is based on total session time and other based on single page stay time. The set of pages visited by a specific user at a specific time is called page viewing time. It varies from 25.5 minutes (Li *et al.*, 2008) to 24 hours (Spilipoulou *et al.*, 2003) at the same time as default time is 30 minutes by Cooley (1997). The second method depends on page stay time which is calculated with the difference between two timestamps. If it goes over 10 minutes the second entry is understood as a new session. The third method based on navigation of users through web pages. But this is accomplished by using site topology which is not used in our method.

In this research work, user sessions are constructed using acyclic graph based algorithms. The general procedure of each of these algorithms is given below:-

- Calculate the browsing time of each page and then discretized,
- Represent a users' page transition in each session as Acyclic Graph (AG) structure,
- Access patterns are extracted from the set of the AGs,
- Mine the patterns using the Closed Frequent Embedded AG mining algorithm (Termier *et al.*, 2007), and
- Cluster the result. By clustering the set of extracted access patterns, similar patterns are grouped for the ease of later analysis.

In the proposed method, sessions are modeled as a graph. Graph mining extracts users' access patterns as a graph structure like the web sites link structure. To make efficient analysis when users handle more pages at the same time using tab browsers graph mining gives excellent results. Vertices are represented as web

pages and edges are represented as hyperlink between pages. A graph is represented as a tuple of vertices, edges which connect the vertices (Li and Feng, 2009). User navigations are given as traversals in a graph. Each traversal can be represented as a sequence of vertices, or equivalently as a sequence of edges. Four types of AGs are analyzed in this study. They are, Directed Acyclic Graph (DAG), Hierarchical Directed Acyclic Graph-based (HDAG), Partial Ancestral Graph-based (PAG) and Mixed Ancestral Graph-based (MAG).

4.3.1. Calculation of Browsing Time

The first task is to calculate browsing time of each page. For this the timestamp fields of the records are considered. Real Browsing time is very difficult to calculate since it depends on network transfer rate, user's actions and computer specifications and so on. Browsing Time and Request Time recorded in log are abbreviated as BT and RT. Browsing time BT_p of page 'p' is equal to the period of time with the time difference between the RT_p of the request which include 'p' as a reference and another RT of the request which include 'p' as a requested page. In the log record, one of the fields is bytes_sent which is the size of the web page. 'c' is the data transfer rate. So the real browsing time is assumed as

$$BT_p = BT_p' - bytes\_sent / c \quad (4.1)$$

where BT_p' is the difference between reference and request page of 'p'.

4.3.2. Discretization

In this step, the calculated browsing time is discretized according to the length and given to each page as the weight which denotes the length of browsing time. The second task in this method is to fix minimum and maximum browsing time for each page as BT_{min} and BT_{max} is used to calculate the weighing function

which is to be used as a label in the graph. They are assumed by the administrators. The next step is to discretise the browsing time and given to each page as the weight which denotes the length of browsing time. Weighting function is calculated using Equations (4.2) to (4.5).

$$Wt\,(p, BT_p) = 0 \text{ when } BT_p \neq \text{null and } BT_p < BT_{min} \tag{4.2}$$

$$Wt\,(p, BT_p) = 1 \text{ when } BT_p \neq \text{null} \tag{4.3}$$

$$BT_{min} \leq BT_p \leq BT_{max} \tag{4.4}$$

$$Wt\,(p, BT_p) = \begin{cases} 2 & BT_p \neq \text{null and } BT_{max} < BT_p \\ 3 & BT_p = \text{null} \end{cases} \tag{4.5}$$

If weight is '0' it is assumed as the time to browse is too short and the user simply passed the page. If weight is '1', administrators conclude it is a valid browsing time and user is interested in the content of the page. If weight is '2' the time is too long and it is assumed as if the user left the page and if the weight is '3' the page does not exists as reference page in that session. It is assumed as the end page and the user does not move from this page.

4.3.3. A cyclic Graph Construction

This section presents details regarding the four selected AGs. Initially the graph terminology is introduced. The concepts defined in this section are illustrated using the graph shown in Figure 4.1.

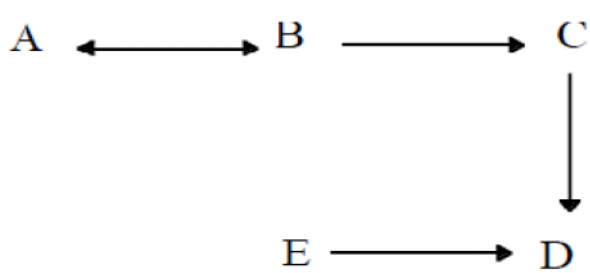

Figure 4.1 : Graph Example

A graph consists of two parts, a set of vertices V and a set of edges E. Each edge in E is between two distinct vertices in V. There are two kinds of edges in E, directed edges A $\rightarrow$ B or A $\leftarrow$ B and double-headed edges A $\leftrightarrow$ B. In either case A and B are endpoints of the edge. Further, A and B are said to be adjacent. In Figure 4.1 the set of vertices is {A, B, C, D, E} and the set of edges is {A $\leftrightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D, E $\rightarrow$ D}. For a directed edge A $\rightarrow$ B, A is the tail of the edge and B is the head of the edge, A is a parent of B and B is a child of A.

An undirected path U between X_1 and X_n is a sequence of edges $<E_1, ..., E_m>$ such that one endpoint of E_1 is X_1, one endpoint of E_m is X_n and for each pair of consecutive edges E_i, E_{i+1} in the sequence, $E_i \neq E_{i+1}$ and one endpoint of E_i equals one endpoint of E_{i+1}.

In Figure 4.1, A $\leftrightarrow$ B $\rightarrow$ C $\leftarrow$ D is an example of an undirected path between A and D. A directed path P between X_1 and X_n is a sequence of directed edges $<E_1, ..., E_m>$ such that the tail of E_1 is X_1, the head of E_m is X_n and for each pair of edges E_i, E_{i+1} adjacent in the sequence, $E_i \neq E_{i+1}$ and the head of E_i is the tail of E_{i+1}. For example, B $\rightarrow$ C $\rightarrow$ D is a directed path. A vertex occurs on a path if it is an endpoint of one of the edges in the path. The set of vertices on A $\leftrightarrow$ B $\rightarrow$ C $\rightarrow$ D is {A, B, C, D}. A path is acyclic if no vertex occurs more than once on the path. The following is a list of all the acyclic directed paths in Figure 4.1.

$$B \rightarrow C,\ C \rightarrow D,\ E \rightarrow D,\ B \rightarrow C \rightarrow D.$$

A graph is a directed graph if it contains no double-headed edges.A graph is a directed acyclic graph (DAG) if it contains no double-headed edges and no directed cycles. A vertex A is an ancestor of B (and B is a descendant of A) if and only if either there is a directed path from A to B or A = B. Thus the ancestor relation is the transitive, reflexive closure of the parent relation. The following table (Table 4.1) lists the child, parent, descendant and ancestor relations in Figure 4.1.

A vertex X is a collider on undirected path U if and only if U contains a subpath $Y \leftrightarrow X \leftrightarrow Z$, or $Y \rightarrow X \leftrightarrow Z$, or $Y \rightarrow X \leftarrow Z$, or $Y \leftrightarrow X \leftarrow Z$; otherwise if X is on U it is a non-collider on U. For example, D is a collider on $C \rightarrow D \leftarrow E$ and C is a non-collider on $B \rightarrow C \rightarrow D$. X is an ancestor of a set of vertices Z if X is an ancestor of some member of Z.

TABLE 4.1

LIST OF GRAPH RELATIONS

Vertex	Children	Parents	Descendants	Ancestors
A	∅	∅	{A}	{A}
B	{C}	∅	{B, C, D}	{B}
C	{D}	{B}	{C, D}	{B, C}
D	∅	{C, E}	{D}	{B, C, D, E}
E	{D}	∅	{D, E}	{E}

For disjoint sets of vertices, X, Y and Z, X is d-connected to Y given Z if and only if there is an acyclic undirected path U between some member X of X and some member Y of Y, such that every collider on U is an ancestor of Z and every non-collider on U is not in Z. For disjoint sets of vertices, X, Y and Z, X is d-separated from Y given Z if and only if X is not d-connected to Y given Z.

For example, the path $A \leftrightarrow B \rightarrow C$ d-connects A and C given ∅; it also d-connects A and C given {D}, {E}, or {D, E}. $E \rightarrow D \leftarrow C$ d-connects E and C given {D}, given {D, B}, {D, A}, or {D, A, B}. The following is a list of all the pairwise d-separation relations (where each pair is followed by a list of all of the sets that d-separate them):

- {A} and {C} are d-separated given: {B}, {B, D}, {B, E}, {B, D, E}
- {A} and {D} are d-separated given: {B}, {C}, {B, C}, {B, E}, {C, E}, {B, C, E}
- {A} and {E} are d-separated given: ∅, {B}, {C}, {B, C}, {B, D}, {C,D}, {B, C, D}
- {B} and {E} are d-separated given: ∅, {A}, {C}, {A, C}, {C, D}, {A, C, D}.

In a graph G, with a set of vertices V containing O, if A and B are in O, then there is an inducing path between A and B given O if and only if there is a path U between A and B such that every member of O that is on U is a collider and every collider is an ancestor of A or B. (If V = O, we will simply say that there is an inducing path between A and B.) It has been shown in Verma and Pearl(1990) that in a DAG G, A and B are d-separated given some subset of O\{A,B} if and only if there is no inducing path between A and B given O.

- **Directed Acyclic Graph (DAG)**

A DAG, is a directed graph with no directed cycles, that is, it is formed by a collection of vertices and directed edges, each edge connecting one vertex to another, such that there is no way to start at some vertex v and follow a sequence of edges that eventually loops back to v again (Nicos, 1975; Thulasiraman and Swamy, 1992; Jørgen, 2008)

The corresponding concept for undirected graphs is a forest, an undirected graph without cycles. Choosing an orientation for a forest produces a special kind of directed acyclic graph called a polytree. However there are many other kinds of directed acyclic graph that are not formed by orienting the edges of an undirected acyclic graph and every undirected graph has an acyclic orientation, an assignment of a direction for its edges that makes it into a directed acyclic graph. For this

reason it may be more accurate to call directed acyclic graphs acyclic directed graphs or acyclic digraphs.

Thus, the DAG consists of the following elements and an example is shown in Figure 4.2.

- Nodes: Each node represents some object or piece of data.
- Directed edges: A directed edge (or "arrow") from one node to another represents some kind of relationship between those two nodes.
- A root node: At least one of the nodes will have no parents. This is the root of the DAG.
- Leaf nodes: One or more of the nodes will have no children. These are called leaves or leaf nodes.

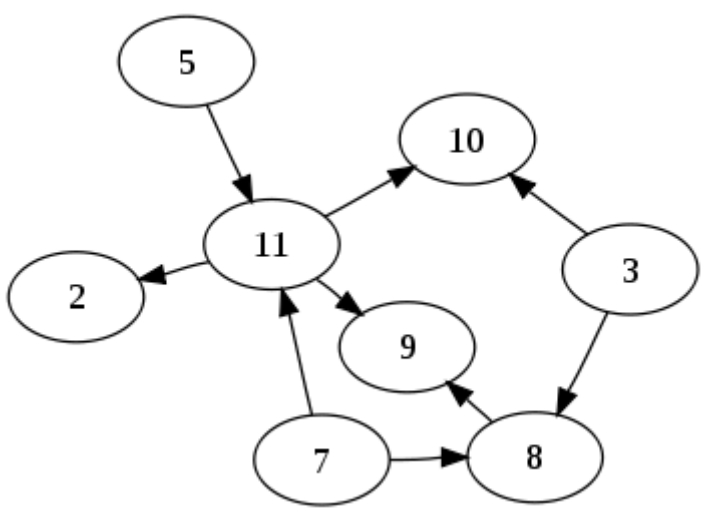

Figure 4.2 : DAG Example

In connection to using DAG for constructing user sessions the following procedure is used. After weighting all pages based on the browsing time a DAG structure is built for each user session. Vertex is labeled by a page and its weight. Each vertex is represented by a set of page and it's weight as (p, wt (p, BTp). Edge

connects reference page to request page for each request. Edges show users page transition and only the direction is considered.

If any cyclic structure exists a new vertex is created and the graph structure is converted to acyclic. In this method DAGs which give user session information for mining is constructed. The advantage over other graph methods is use of numerical values like browsing time is considered. A simplest form of the weighting function is used depending on the browsing time which is longer or shorter than the threshold. The threshold is based on the content of the page.

- **HDAG Construction**

This section describes HDAG as a Directed Acyclic Graph (DAG) with hierarchical structures (Suzuki *et al.*, 2003). That is, certain nodes contain DAGs within themselves. The Kernel Function is described below.

$x \in X$ is a composite structure and $x_1, \ldots, x_D$ where $x_d \in X_d$. R is relation on the set $X_1 \times \cdots X_D \times X$ such that $R(x, x)$ is true if x are the parts of x. $R^{-1}(x)$ is defined as $R^{-1}(x) = \{x: R(x, x)\}$.

Consider that $x, y \in X$, x be the parts of x with $x = x_1, \ldots, x_D$ and y be the parts of y with $y = y_1, \ldots, y_D$. Then, the similarity $K(x, y)$ between x and y is defined as the following generalized convolution:

$$K(x, y) = \sum_{x \in R^{-1}(x)} \sum_{y \in R^{-1}(y)} \prod_{d=1}^{D} K_d(x_d, y_d) \qquad (4.6)$$

Convolution Kernels are abstract concepts and that instances of them are determined by the definition of sub-kernel $K_d(x_d, y_d)$. An explicit definition of both the Tree Kernel (Collins and Duffy, 2001) and String Sub-sequence Kernel (SSK) $K(x, y)$ is written as in Equation (4.7).

$$K(x, y) = \sum_{x \in R^{-1}(x)} \sum_{y \in R^{-1}(y)} \prod_{d=1}^{D} K_d(x_d, y_d) \qquad (4.7)$$

All sub-structures occurring in x and y are listed, where M represents the total number of possible sub-structures in the objects. ϕ, the feature mapping from the sample space to the feature space, is given as $\phi(x) = (\phi_1(x), \dots, \phi_M(x))$.

There are several levels of chunks in the web usage mining such as web pages, web links, named entities and these are bound by relation structures. HDAG is designed to enable the representation of all of the structures in the web usage mining, hierarchical structures for chunks and DAG structures for the relations of chunks. This richer representation is extremely useful to enhance the performance of similarity measure between web pages, moreover, learning and clustering tasks in the application areas of web usage mining. Here, the nodes are allowed to have more than zero attributes, because nodes in texts usually have several kinds of attributes. For example, attributes include web pages, web links, etc.

The set of nodes in HDAGs G_1 and G_2 as P and Q, respectively, p and q represent nodes in the graph that are defined as $\{p|p_i \in P, i = 1, \dots, |P|\}$ and $\{q|q_i \in Q, j = 1, \dots, |Q|\}$ respectively. The expression $p_1 \rightarrow p_4 \rightarrow p_7$ to represent the path from p_1 to p_7 through p_4.

"Attribute sequence" is defined as a sequence of attributes extracted from nodes included in a subpath. This framework makes similarity evaluation robust; the similar sub-structures can be computed in the value of similarity, on the contrary to exact matching that never evaluate the similar substructure. The similarity between HDAGs, which is the definition of the HDAG Kernel, follows Equation (4.7) where input objects x and y are $G_1$and G_2 respectively.

- **Partial Ancestral Graph (PAG)**

Variables in a DAG are divided into types observed(measurable) and latent(non-measurable). A PAG (Partial Ancestral Graph) is used to represent any

subset of Equiv (G (O, L))(Equivalence graph). A PAG is a extended graph consist of set of vertices O and a set of edges between vertices were they may be following kind of edges: A ↔ B, A o─o B, A o→ B, A ←o B, A→ B or A←B. we say that the A endpoint of A→ B is "-" ; A endpoint of an A ↔ B is "<"; and then A endpoint of A o→ B is "o". The conventions of B endpoint are analogous. In addition, pair of edge points may be connected by underlining. A partial Ancestral Graph for the set of directed graph G each carrying the same set of Observed variables O, contain partial information about the ancestral relation in G, namely only those ancestral relations is common to all members of G.

If G is a set of directed graph included in Equiv (G (O,L)), ψ (with vertices O) is a PAG for G if and only if

1. There is the edge between A and B in ψ if and only if every graph in G does not entail that A and B are the independent to the subset of O\{A,B}.If there is an edge in ψ out of A, i.e. A → B then A is the ancestor of B in every graph in G,
2. If there is an edge in ψ into B, then in every PAG in G, B is not the ancestor of A,
3. If there is any underlining A*-*<u>B</u>*-*C in ψ then B is the ancestor of (atleast one of) A or C in every graph of G, and
4. Any edge endpoint not marked in one of the above ways is left with a small circle thus o-*.

Only Condition (1) gives necessary and sufficient condition about the features of PAG. All the other conditions are merely necessary conditions. That means that there can be more than one PAG representing a given set G. Thus the PAG can be used to represent both the ancestor relation among the members of O common to the members of G and the set conditional independence relation among the members of O in G.

- **Mixed Ancestral Graph (MAG)**

A MAG (or mixed ancestral graph) is a graph with two kinds of edges: directed edges (e.g. A → B) and bi-directed edges, (e.g. C ↔ D). The MAG that represents a DAG G (also called MAG(G,O) with a set of observed variables O) can be constructed in the following way:

- Place the edge A→B in MAG(G,O) if and only if A is an ancestor of B in G and there is an inducing path between A and B given O in G.
- Place the edge A ↔ B in MAG(G,O) if and only if A is not an ancestor of B in G, B is not an ancestor of A in G and there is an inducing path between A and B given O in G.

Some examples of MAGs are shown in Figure 4.3, where O = {A,B,C,D}. In cases where the distinction between latent variables and measured variables is important, the latent variables are enclosed in ovals.

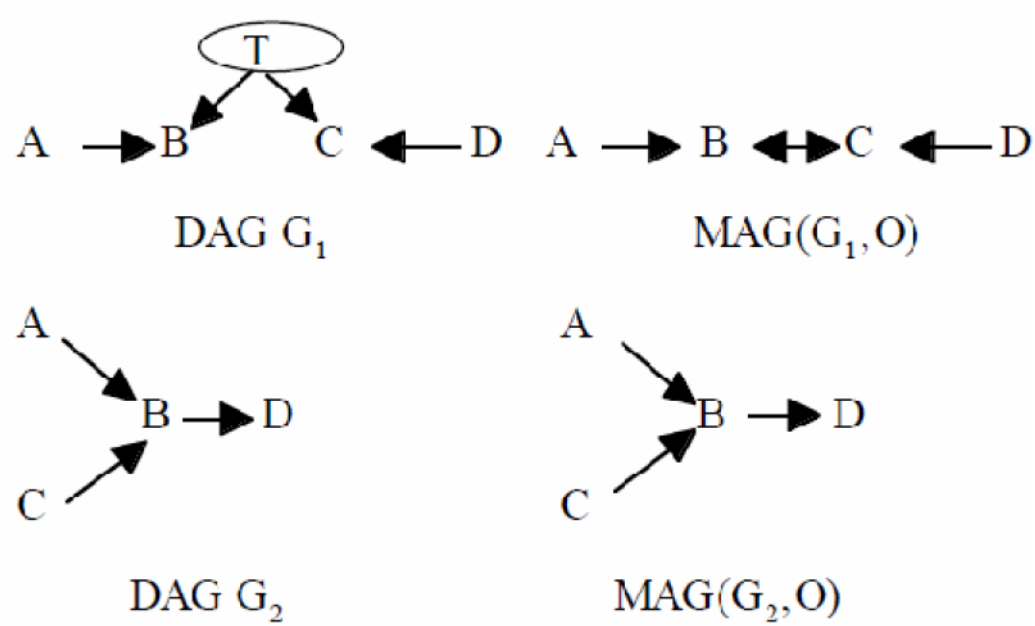

Figure 4.3 : MAG Example

It has been shown in Spirtes *et al.* (1993) that in a MAG G, A and B are d-separated given some subset of {A,B} if and only if there is no inducing path between A and B in G. Thus a MAG M may be considered to represent any DAG G such that M = MAG(G,O). MAG(G,O) represents the following features of a DAG G with latent variables:

- The ancestor relations among the members of O in G
- The d-separation relations among the members of O in G.

In general, each MAG represents many different latent variable models. A MAG can thus also be considered a representation of a set of conditional independence relations among variables in O (which in some cases cannot be represented by any DAG containing just variables in O.) A MAG imposes no restrictions on the set of distributions it represents other than the conditional independence relations that it entails. The class of MAGs is neither a subset nor a superset of other generalizations of DAGs such as chain graphs cyclic directed graphs, or cyclic chain graphs.

It is not the case that an arbitrary graph is a MAG, i.e. there may be no DAG G such that M = MAG(G,O). The following theorem states necessary and sufficient conditions for M to be a MAG. Let the canonical graph G(M) for a MAG M be constructed in the following way: If O is the set of vertices in M, then the set of vertices in G(M) is $O \cup T$, where $T_{i,j} \in T$ if there is an edge $X_i \leftrightarrow X_j$ in M, for X_i, X_j in O, there are edges $X_i \leftarrow T_{i,j} \rightarrow X_j$ in G(M) if and only if there is an edge $X_i \leftrightarrow X_j$ in M and an edge $X_i \rightarrow X_j$ in G(M) only if there is an edge $X_i \rightarrow X_j$ in M.

Lemma 1 : If M is a MAG, then G(M) has the same ancestor relations among members of O as M does.

Lemma 2 : If M = MAG(G) has vertices O, then M and G have the same ancestor relations among members of O.

Theorem 1 : A graph M is a MAG if and only if:

1. If there is an inducing path between X_i and X_j in M, then X_i and X_j are adjacent in M.
2. If there is an edge $X_i \rightarrow X_j$ in M, then X_j is not an ancestor of X_i in M, but X_i is an ancestor of X_j in M.
3. If there is an edge $X_i \leftrightarrow X_j$ in M, then X_j is not an ancestor of X_i and X_i is not an ancestor of X_j in M.

The proofs of the lemmas and theorem can be found in Spirtes *et al.* (1997).

A MAG can also be considered a representation of a set of conditional independence relations among variables in O (which in some cases cannot be represented by any DAG containing just variables in O. A MAG imposes no restrictions on the set of distributions it represents other than the conditional independence relations that it entails. The MAG have two separate uses, they can be used in algorithm to perform fast condition and the other one is used to calculate the effect of any ideal intervention in the system.

4.3.4. Pattern Extraction

Once a graph and its traversals are specified, valuable information can be retrieved through graph mining. Normally they are in the form of patterns. Frequent patterns which are sub traversals occurred in a large ratio are considered for analysis. To discover AG's i.e., sub graphs mining algorithm is used which derive closed frequent sets. It replaces closed frequent AG mining problem with the problem of closed frequent item-set mining on edges with the restriction that all the labels of the vertices in an AG must be distinct. By the reconstruction of AG structures from the mined closed frequent edge set, closed frequent AG's are obtained. The algorithm extracts the embedded AGs based on not only on parent-child relationship but also ancestor-descendant relationship of vertices. The input

for the algorithm is the user sessions, AG set and the minimum support €(≥ 0) as inputs. Access patterns are obtained as frequent AGs.

The procedure used during preprocessing was described in this chapter and is summarized in Figure 4.4.

1. Perform cleaning and filtering
    a. Remove all image, video, audio entries
    b. Remove all unsuccessful HTTP entry codes (Status code ≤200 and ≥299)
    c. Retain only those entries have GET and POST in request method field
    d. Remove all entries with blank IP addresses
    e. Removal all entries which requests robots.txt along with entries whose time taken is less than two seconds (indicating they are automated program traversal entries)
    f. Remove all entries whose depth level is more than five
2. User Identification
    a. Identify unique users using IP address
3. Session Identification
    a. Construct user session using any one of the selected four acyclic graph based algorithms
4. Assign sequential unique codes to each individual URL
5. Reformat the log file, by assigning the URLs in each session with the unique code formulated in Step 4
6. Prepare a URL-Session Binary Table, which is created by assigning a value of 1, if a URL has been visited in a session, else a 0 value is inserted.
7. Calculate the total number of visits in made to each URL, session wise and construct the final data.
8. Perform clustering using H-NUC Algorithm

Figure 4.4 : Preprocessing Procedure

4.4. CHAPTER SUMMARY

Session Construction was successfully completed using various DAGs like Directed Acyclic Graph, Hierarchical Directed Acyclic Graph, Partial Ancestral Graph and Mixed Ancestral Graph. A comparative study indicates that the use of DAG is a time efficient strategy for session construction.

One issue faced by WEB PAGE PREDICTION SYSTEM at this stage is the web log size, which, when huge, affects the performance of prediction. In the next phase of the study, to solve this issue, a technique that can reduce the size of the preprocessed web log file data is proposed. Details regarding this technique are presented in the next chapter, **Potential User Identification**, (Chapter 5).