Data & Knowledge Engineering 68 (2009) 867–885

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

COWES: Web user clustering based on evolutionary web sessions

Ling Chen [b], Sourav S. Bhowmick [a,*], Wolfgang Nejdl [b]

[a] School of Computer Engineering, Division of Information Systems, Nanyang Technological University, Singapore 639798, Singapore
[b] L3S Research Center, University of Hannover, Germany

ARTICLE INFO

Article history:
Available online 19 May 2009

Keywords:
Web Usage Mining
Web user clustering
Historical web session
Evolutionary pattern mining

ABSTRACT

As one of the most important tasks of Web Usage Mining (WUM), web user clustering, which establishes groups of users exhibiting similar browsing patterns, provides useful knowledge to personalized web services and motivates long term research interests in the web community. Most of the existing approaches cluster web users based on the snapshots of web usage data, although web usage data are evolutionary in the nature. Consequently, the usefulness of the knowledge discovered by existing web user clustering approaches might be limited. In this paper, we address this problem by clustering web users based on the evolution of web usage data. Given a set of web users and their associated historical web usage data, we study how their usage data change over time and mine evolutionary patterns from each user's usage history. The discovered patterns capture the characteristics of changes to a web user's information needs. We can then cluster web users by analyzing common and similar evolutionary patterns shared by users. Web user clusters generated in this way provide novel and useful knowledge for various personalized web applications, including web advertisement and web caching.

1. Introduction

Web Usage Mining (WUM) is an active area of research and commercialization. The goal of WUM is to leverage the data collected as a result of user interactions with the web to learn user models which are beneficial for web personalization. Existing web usage data mining techniques include statistical analysis [18], association rules [14,9], sequential patterns [24], classification [12], and clustering [13,11]. An important topic in Web Usage Mining is clustering web users – discovering clusters of users that exhibit similar information needs, e.g., users that access similar pages. By analyzing the characteristics of the clusters, web users can be understood better and thus can be provided with more suitable and customized services [23].

There are quite a few methods for clustering web users proposed in the literature [8,23,21]. In general, web user clustering consists of three phases: *data preparation*, *cluster discovery*, and *cluster analysis*. Since the last phase is application-dependent, let us briefly describe the first two. In the first phase, *web sessions* of users are extracted from the web server log by using some user identification and session identification techniques [7]. A *web session*, which is an episode of interaction between a web user and the web server, consists of pages visited by a user in the episode [8]. For example, Fig. 1a shows four requests from one session.

The first line means that the user at *foo.ntu.edu* accessed the page www.uow.edu/sce/Jeffrey/pub.html at 10:30:05 on January 01, 2005. In the second phase, clustering techniques are applied to generate clusters of users. For example, consider the three users, u_1, u_2 and u_3, and their web session data about visiting the web site rooted at the home page a in a particular time period p_1, as shown in Fig. 2a. Note that, only accessed pages are presented in the figure, where other information in web sessions are ignored. Existing web user clustering methods which use commonly accessed web pages as the clustering feature, such as the work in [8], will group the three users together as their web sessions share common pages.

* Corresponding author.
E-mail addresses: lchen@l3s.de (L. Chen), assourav@ntu.edu.sg (S.S. Bhowmick), nejdl@l3s.de (W. Nejdl).

0169-023X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2009.05.002

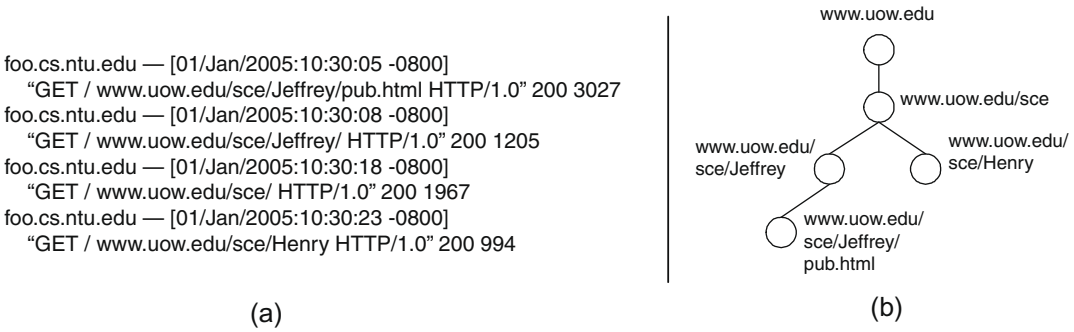

Fig. 1. Web session and web session tree.

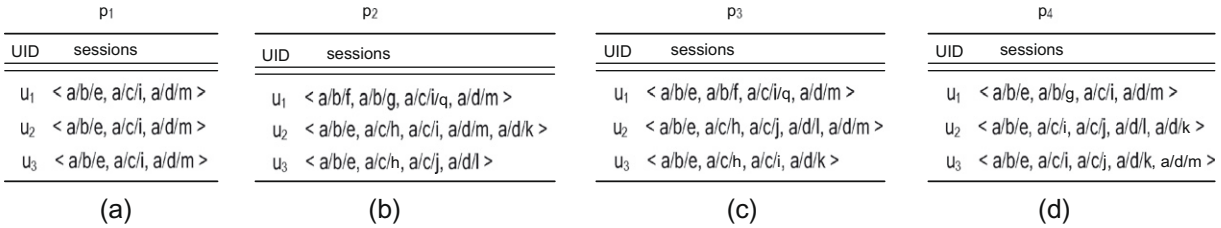

Fig. 2. Historical web sessions of users u_1, u_2 and u_3.

1.1. Motivating example

Existing web user clustering methods cluster users based on web sessions collected in a certain time period. However, web usage data is *evolutionary* in nature. The information needs of users may vary over time, consequently, pages accessed by users in different time periods may be different as well. For example, Fig. 2b–d show the web sessions of users u_1, u_2 and u_3 visiting the web site at subsequent time periods p_2, p_3 and p_4, where users accessed different pages in different periods. Such evolutionary nature of the web usage data poses both challenges and opportunities to web user clustering. In particular, the evolutionary nature of web usage data leads to the following two challenging problems:

- **Maintenance of web user clustering results**: Take the web sessions in Fig. 2 as an example. Although users u_1, u_2 and u_3 accessed similar web pages in the period p_1, they visited different pages in p_2 and beyond. Hence, the clustering results, generated by existing algorithms in a particular time period, have to be updated with the changes to web usage data. This requires development of efficient incremental web user clustering techniques.
- **Discovery of novel web user clusters**: Considering the evolutionary characteristics of web usage data, interesting and novel web user clusters may be discovered. For example, clusters of users that exhibit similar characteristics in the evolution of their usage data may be discovered. Examples of such clusters will be illustrated in the following paragraphs.

The first problem has been addressed by some incremental clustering algorithms proposed in the literature [22,17]. Our focus here is the second problem. Particularly, we propose a new web user clustering method, COWES (**C**lustering **O**f **W**eb users based on historical web session), which discovers novel knowledge of web users sharing common evolutionary characteristics of their usage data.

Let us look at novel clusters which can be discovered based on evolutionary characteristics of web usage data in our motivating example in Fig. 2. Pages accessed in a web session can be organized into a hierarchical structure, called *web session tree*, based on the URLs of the pages [8]. For example, Fig. 1b is the web session tree constructed for the pages in the web session shown in Fig. 1a. A web session tree represents the information needs of a user. Similarly, the sequences of historical web sessions of web users u_1, u_2 and u_3 in Fig. 2 are represented as sequences of web session trees as depicted in Fig. 3. Each gray node in the figure represents a page that was not accessed in the subsequent web session and each black node denotes a page that was not accessed in the previous session. Obviously, changes to the structures of web session trees of a user, i.e., gray and black nodes, reflect varying information needs of the user. Observing the sequence of web session trees of user u_1, we notice that u_1 frequently varies his information needs in the web session subtree a/b (we use the root path arriving at the root of a web session subtree to denote the subtree), which is highlighted by dotted lines in Fig. 3. Similarly, users u_2 and u_3 frequently vary their information needs in the web session subtrees a/c and a/d. In this paper, we extract such subtrees as a type of evolutionary frequent pattern called FRACTURE (FRequently And Concurrently muTating substructUREs), which we proposed in the context of XML documents in our previous work [5]. FRACTURES are mined from the evolution history of tree-structured

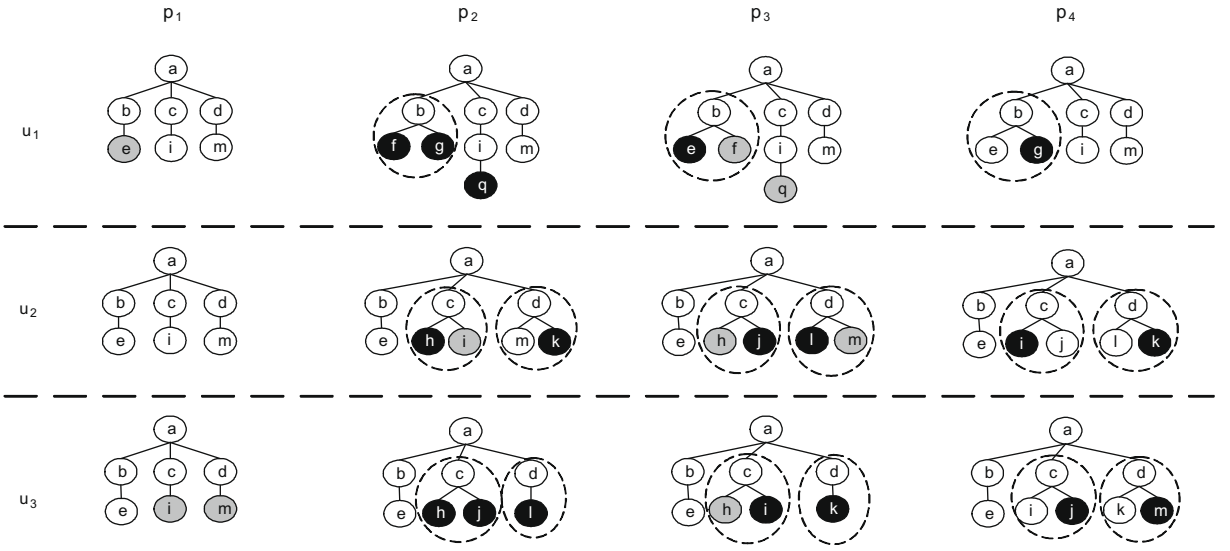

Fig. 3. Historical page hierarchies of users u_1, u_2 and u_3.

data (e.g., XML document). A discovered FRACTURE is a set of substructures that frequently change together. Since they capture the evolutionary characteristics of each user's usage data, we propose to use the FRACTURES discovered from each user's historical web sessions as the clustering feature to group web users. For example, given the web users and their historical web sessions in Fig. 2, our proposed clustering method COWES will cluster u_2 and u_3 together while leaving u_1 in another cluster. Knowledge can be inferred from the clustering result that users in the same cluster exhibit similar variation patterns in their information needs. Note that such clusters cannot be discovered by existing approaches based on snapshot web usage data.

1.2. Applications

Similar to existing web user clustering, the results of COWES provide knowledge on users' information needs and can be used in various web personalization applications. In contrast to existing methods, COWES discovers novel knowledge on the evolutionary characteristics of users' information needs. Thus, the results of COWES are useful for various web personalization scenarios. We elaborate two of the applications of COWES in the following paragraphs.

- **Intelligent web advertisement**: 99% of all web sites offer standard banner advertisements [2], underlying the importance of this form of on-line advertising. For many web-based organizations, revenue from advertisements is often the only or the major source of income (e.g., Yahoo.com, Google.com) [1]. One of the ways to maximize revenues for the party who owns the advertising space is to design intelligent techniques for the selection of an appropriate set of advertisements to display in appropriate web pages. Web user clusters generated by COWES can be beneficial for designing intelligent advertisement placement strategies. For example, after clustering users in Fig. 2 based on the evolution of historical web sessions, we know that the variation of information needs of u_1 is different from that of users u_2 as well as u_3. Suppose an advertisement is targeting all the three users. Although all of them accessed the page $a/b/e$ in the current period p_4, the advertisement should be put in the page a/b instead of $a/b/e$ for user u_1 because he frequently changes his information needs under a/b.
- **Proxy cache management**: Web caching is another interesting problem [3,24] as web caches can reduce not only network traffic and but also download latency. Because of the limited size of cache regions, it is important to design effective replacement strategies to maximize hit rates. One of the frequently used replacement strategies is LRU (Least Recently Used), which assigns priorities to the most recently accessed pages. Web user clusters generated by COWES can be used with LRU to manage the caching region more optimally. For example, analyzing the clusters generated by COWES in Fig. 2, we know that users u_2 and u_3 frequently vary their information needs under a/c and a/d together. In other words, once user $u_2(u_3)$ varies information needs under a/c, he probably will vary his information needs under a/d as well. Thus, if $u_2(u_3)$ accesses different pages under a/c in a subsequent time period, such as p_5, we can degrade the priority of cached pages under a/d so that their eviction from the cache becomes more probable.

1.3. Overview of COWES

The overview of COWES system is presented in Fig. 4. The input is a collection of web users $\{u_1, u_2, \ldots, u_n\}$. Each user is associated with a sequence of historical web sessions that are collected at a sequence of time periods $\langle p_1, p_2, \ldots, p_m \rangle$ with some

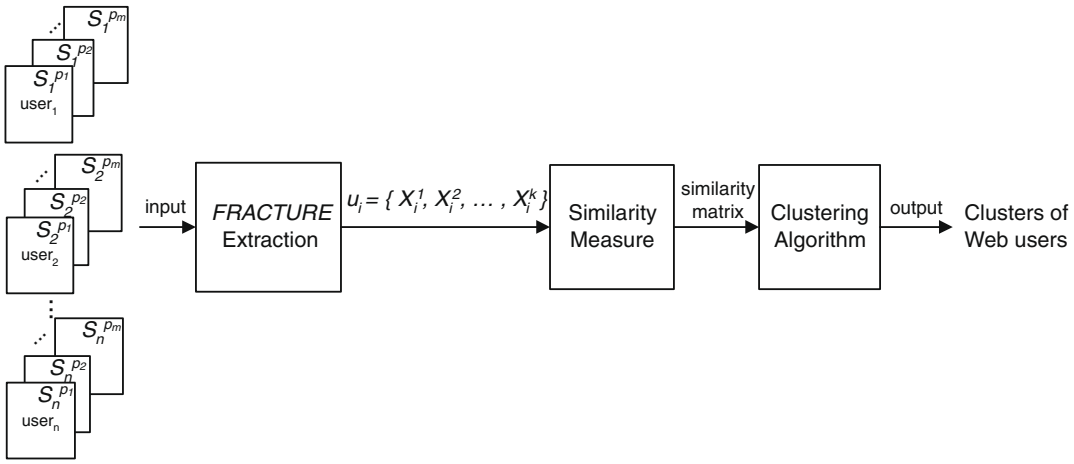

Fig. 4. Overview of COWES.

particular time granularity. Each sequence of web sessions of user u_i can be represented as a sequence of web session trees $\langle S_i^{p_1}, S_i^{p_2}, \ldots, S_i^{p_m} \rangle$.

The first step extracts FRACTUREs from web session trees. As discussed before, we aim to cluster users based on the evolutionary characteristics of their usage data. In particular, we mine FRACTUREs, where each FRACTURE is a set of subtrees that frequently undergo significant changes together, from each user's historical web session trees and use FRACTUREs as the clustering feature. Then, each web user can be represented as a set of FRACTUREs $u_i = \{X_i^1, X_i^2, \ldots, X_i^k\}$.

Given a set of users represented by the FRACTUREs mined from their historical web session trees, the second step is to measure the *similarity* between users based on their FRACTUREs. We define the User Similarity in terms of their shared FRACTUREs by considering not only their elements (e.g., subtrees) but also their *strength* (e.g., how frequently the subtrees undergo significant changes together). After computing the similarity between each pairs of users, a similarity matrix of users can be generated.

Finally, given a similarity matrix of web users, existing appropriate clustering algorithms, such as partitional clustering algorithms or agglomerative clustering algorithms, can be performed on the similarity matrix to generate the clusters of web users.

1.4. Contributions

The main contributions of this paper are summarized as follows:

- We propose an approach that is, to the best of our knowledge, the first one to discover novel knowledge by clustering web users based on the evolutionary features of historical web sessions.
- We capture the evolution characteristics web usage data with an interesting *evolutionary pattern* and show that user clusters generated based on this pattern are useful in real-life applications.
- We define *similarity* metrics which measure the proximity of the evolutionary patterns and the proximity of web users in terms of their evolutionary patterns, respectively.
- We present the results of extensive experiments with both synthetic and real datasets that we have conducted to demonstrate the performance of COWES and the novelty of generated clusters.

The rest of the paper is organized as follows. In Section 2, we explain the notion of FRACTURE that can be mined from historical web session trees. We define the similarity metrics in Section 3. In Section 4, we present the framework of our clustering approach. We evaluate the performance of clustering web users based on historical web sessions in Section 5 and review related works in Section 6. Finally, we present some conclusive remarks in Section 7. A preliminary version of this paper appeared in [6]. Table 1 summarizes the notations used in this paper.

2. FRequently And Concurrently muTating substructUREs (FRACTURE)

In this section, we briefly introduce the evolutionary pattern FRACTURE, which is used as our clustering feature. Readers can refer to our previous work [5] for details.

As in [8], pages in a web session can be organized into a page hierarchy based on their URLs. Hereafter, we refer to a page hierarchy of a web session as a *web session tree*. Formally, a web session tree is an unordered tree $S = \langle N, E \rangle$, where N is the set of nodes where a leaf node represents a web page corresponding to a file in the web server and a non-leaf node represents

Table 1
A summary of symbols in this paper.

Symbol	Description
u	A web user
p	A time period with certain granularity
S	A web session tree
s_i	A web session subtree
N	A set of web session tree (subtree) nodes
E	A set of web session tree (subtree) edges
X	A FRACTURE pattern
DoC	Degree of Change
SoC	Significance of Change
FoC	Frequency of Change
α	User-defined threshold of DoC
β	User-defined threshold of FoC
γ	User-defined threshold of SoC
$d(X_1, X_2)$	The distance between a pair of FRACTURES X_1 and X_2
$FS(u_1, u_2)$	FRACTURE similarity of users u_1 and u_2
$US(u_1, u_2)$	User Similarity of users u_1 and u_2
$X_1 = X_2$	A pair of identical FRACTURES
$X_1 \approx X_2$	A pair of Approximate FRACTURES
$L(s)$	Root path arriving at the root of web session subtree s
$PL(s_i, s_j)$	Prefix Level of web session subtrees s_i and s_j
$\omega(X_1, X_2)$	Weight assigned to FRACTURE pair X_1 and X_2
IS	Average internal similarity of clusters
ES	Average external similarity of clusters

a web page corresponding to a directory in the server, E is the set of edges where each edge from a parent node to a child node represents the consisting-of relationship between the corresponding pages. In particular, a node $r, r \in N$, is the root of the tree which represents the home page of a web site. An example web session tree is shown in Fig. 1. Accordingly, a tree $s_i = \langle N_i, E_i \rangle$ is a web session subtree, denoted as $s_i \prec S$, if and only if (1) $N_i \subseteq N$, (2)$E_i \subseteq E$, (3) the labeling of N_i is preserved in s_i, (4) for a node $x \in N$, if $x \in N_i$ then all descendants of x (if any) must be in N_i.

Given a sequence of historical web session trees of a web user, we are interested in how the structures of the trees change, which reflects the variation of the user's information needs. We define two basic operations that change the structure of a tree as follows.

- $Insert(x(name, value), y)$: This operation creates a new node x, with node name "name" and node value "value", as a child node of node y in a web session tree.
- $Delete(x)$: This operation is the inverse of the insertion one. It removes node x from a web session tree.

A web session subtree is considered as a *changed subtree* once a change operation, i.e., insertion or deletion, occurs to it.

For example, the sequence of web session trees of user u_2 in Fig. 3 is redrawn in Fig. 5, where the black nodes depict the newly inserted nodes in the current session and the grey nodes depict the nodes that will be deleted in the next session. Consider the first two versions of the web session tree. A new node h is inserted in the subtree a/c. Thus, the subtree a/c is a changed subtree in the two versions.

Changed web session subtrees are elements of FRACTURE. In order to measure the interestingness of a FRACTURE, we defined three metrics: *Degree of Change* (*DoC*), *Frequency of Change* (*FoC*) and *Significance of Change* (*SoC*). Given a sequence of web session trees, the metric *DoC* measures how significantly a subtree changed in two sessions, the metric *FoC* measures how frequently a set of subtrees change together in the sequence and *SoC* measures how frequently a set of subtrees undergo significant changes together.

Definition 1 (*DoC*). Given two versions of a web session subtree s^{p_i} and $s^{p_{i+1}}$, let $s^{p_i} = \langle N^i, E^i \rangle$ and $s^{p_{i+1}} = \langle N^{i+1}, E^{i+1} \rangle$. The *Degree of Change* (*DoC*) of web session subtree s in the two versions is[1]:

$$DoC(s, p_i, p_{i+1}) = \frac{|\{x | x \in \{N^i \cup N^{i+1}\} \& x \notin \{N^i \cap N^{i+1}\}\}|}{|\{x | x \in \{N^i \cup N^{i+1}\}\}|}$$

That is, the *DoC* of a subtree in two versions is the ratio of the number of inserted/deleted nodes to the total number of unique nodes of the subtree in the two versions. The value of *DoC* ranges from 0 to 1. The higher the *DoC* value, the more significantly the subtree changes. For example, in Fig. 5, the *DoC* of the subtree a/c in the first two sessions is 1/3.

While the metric *DoC* is defined for a single subtree, the metrics *FoC* and *SoC* are defined for a set of subtrees.

[1] For simplicity, our definition of *DoC* is based on the assumption that no two nodes bear same labels in a tree structure. Our more refined definition in [5] also considers the situation of duplicate labels.

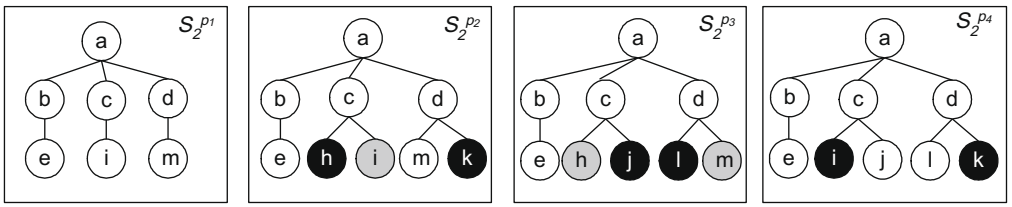

Fig. 5. Historical web session trees of u_2.

Definition 2 (*FoC*). Let $\langle S^{p_1}, S^{p_2}, \ldots, S^{p_n}\rangle$ be a sequence of n historical web session trees of a web user. Let X be a set of subtrees, $X = \{s_1, s_2, \ldots, s_m\}$, where $\forall j(1 \leqslant j \leqslant m), \exists i(1 \leqslant i \leqslant n)$ s.t. $s_j \prec S^{p_i}$. Let $DoC(s_j, p_i, p_{i+1})$ be the *DoC* of subtree s_j from the p_i version to the p_{i+1}th version. The *Frequency of Change* (*FoC*) of the set X is:

$$FoC(X) = \frac{\sum_{i=1}^{n-1} V_i}{n-1}$$
$$\text{where } V_i = \prod_{j=1}^{m} V_{j_i} \quad \text{and} \quad V_{j_i} = \begin{cases} 1, & \text{if } DoC(s_j, p_i, p_{i+1}) \neq 0 \\ 0, & \text{if } DoC(s_j, p_i, p_{i+1}) = 0 \end{cases} \quad 1 \leqslant j \leqslant m$$

Obviously, *FoC* of a set of subtrees X is the fraction of transitions between successive sessions where all subtrees in X changed. The value of *FoC* ranges from 0 to 1 as well. The more times the set of subtrees change together, the higher the *FoC*. For example, consider the sequence of web sessions trees in Fig. 5 again. Let X be two subtrees: a/c and a/d. Then, $FoC(X) = 3/3 = 1$ as both subtrees changed in the three transitions of successive versions.

Definition 3 (*SoC*). Let $\langle S^{p_1}, S^{p_2}, \ldots, S^{p_n}\rangle$ be a sequence of n historical web session trees of a web user. Let X be a set of subtrees, $X = \{s_1, s_2, \ldots, s_m\}$, where $\forall j(1 \leqslant j \leqslant m), \exists i(1 \leqslant i \leqslant n)$ s.t. $s_j \prec S^{p_i}$. Let $DoC(s_j, p_i, p_{i+1})$ be the degree of change of subtree s_j from the p_i version to the p_{i+1} version and $FoC(X)$ be the Frequency of Change of the set X. Given a user-defined threshold of *DoC* α, the *Significance of Change* (*SoC*) of the set X is defined as follows:

$$SoC_\alpha(X) = \frac{\sum_{i=1}^{n-1} D_i}{(n-1) * FoC(X)}$$
$$\text{where } D_i = \prod_{j=1}^{m} D_{j_i} \quad \text{and} \quad D_{j_i} = \begin{cases} 1, & \text{if } DoC(s_j, p_i, p_{i+1}) \geqslant \alpha \\ 0, & \text{otherwise} \end{cases} \quad 1 \leqslant j \leqslant m$$

That is, the *SoC* of a set of subtrees X is the ratio of the number of times all subtrees in X change significantly (compared with the user-defined threshold of *DoC*) to the number of times all subtrees in X changed together. The range of the value of *SoC* is [0,1]. For example, let X be the two subtrees of a/c and a/d in Fig. 5. Let the user-defined threshold of *DoC* be 0.5. $SoC_{0.5}(X) = 2/3$ because the two subtrees changed together for three times, while they changed significantly together in the last two times. Hereafter, we omit the subscript of *SoC* when the *DoC* threshold α is clear from the context.

Based on the above metrics, the *FRequently And Concurrently muTating substructUREs* (FRACTURE) can be defined as follows.

Definition 4 (FRACTURE). Let $\langle S^{p_1}, S^{p_2}, \ldots, S^{p_n}\rangle$ be a sequence of n historical web session trees of a web user. Let X be a set of subtrees, $X = \{s_1, s_2, \ldots, s_m\}$, where $\forall j(1 \leqslant j \leqslant m), \exists i(1 \leqslant i \leqslant n)$ s.t. $s_j \prec S^{p_i}$. Given the user-defined *DoC* threshold α, *FoC* threshold β and *SoC* threshold γ, X is a *FRequently And Concurrently muTating substructURE* (FRACTURE) if it satisfies the following two conditions:

(1) *FoC* of the set is no less than the user-defined *FoC* threshold $\beta, FoC(X) \geqslant \beta$.
(2) *SoC* of the set is no less than the user-defined *SoC* threshold $\gamma, SoC(X) \geqslant \gamma$.

That is, a FRACTURE, mined from a sequence of historical web session trees of a web user, is a set of web session subtrees that frequently change together and frequently change significantly together. For example, suppose the user-defined thresholds are: $\alpha = \beta = \gamma = 0.5$. Let X be the two subtrees a/b and a/d in Fig. 5. Then X is a FRACTURE as $FoC(X) = 1 \geqslant \beta$ and $SoC(X) \approx 0.67 \geqslant \gamma$.

For each web user, a set of FRACTUREs can be mined from his/her historical web session trees with respect to a set of specified thresholds. Consider the web users and their historical web session trees in Fig. 2. Suppose the thresholds are $\alpha = \beta = \gamma = 0.5$. Table 2 shows the discovered FRACTUREs of each user[2]. Algorithm to discover FRACTUREs is discussed in [5].

[2] In the second column of the table, we use the subscript to denote the user identity and the superscript to denote the pattern identity.

Table 2
FRACTURES discovered from web session trees.

User ID	FRACTURE *ID*	FRACTURE	*FoC*	*SoC*
u_1	X_1^1	{a/b}	1.00	1.00
u_1	X_1^2	{a/c/i}	0.66	1.00
u_1	X_1^3	{a/b, a/c/i}	0.66	1.00
u_2	X_2^1	{a/c}	1.00	0.66
u_2	X_2^2	{a/d}	1.00	0.66
u_2	X_2^3	{a/c, a/d}	1.00	0.66
u_3	X_3^1	{a/c}	1.00	1.00
u_3	X_3^2	{a/d}	1.00	0.66
u_3	X_3^3	{a/c, a/d}	1.00	0.66

3. Similarity measure

After representing each user as a set of FRACTURES, we need to measure the proximity of users. We measure the similarity between users in terms of their *shared* FRACTURES. In this section, we present two similarity metrics, *User Similarity* and FRACTURE *Similarity*.

Given two users represented as two sets of FRACTURES, they may share some of their FRACTURES. Then, the complete set of features is the combination of the two set of FRACTURES. For example, suppose user u_1 has a set of FRACTURES $\{X_1, X_2\}$ and user u_2 has a set of FRACTURES $\{X_1, X_3\}$. The complete set of features are $\{X_1, X_2, X_3\}$. Users u_1 and u_2 share the first FRACTURE only. Suppose a similarity measure FRACTURE *Similarity* (*FS*) can be computed on a pair of shared FRACTURES. According to the algorithm introduced in Section 2.4.6 in [19], the measure *User Similarity* (*US*) can be computed by the following three steps:

- For the kth FRACTURE, compute a similarity, $FS(u_1, u_2)$, in the range [0,1].
- Define an indicator variable δ_k, such that

$$\delta_k = \begin{cases} 0, & \text{if one of the users has a missing value for the } k\text{th fracture} \\ 1, & \text{otherwise} \end{cases}$$

- Compute the overall similarity between the two users using the following formula:

$$US(u_1, u_2) = \frac{\sum_{k=1}^{n} \delta_k FS_k(u_1, u_2)}{\sum_{k=1}^{n} \delta_k} \tag{1}$$

However, in our application, we observed that such a similarity measure has the following two limitations.

First, the denominator in Eq. (1) counts only the number of shared FRACTURES. Actually, the number of FRACTURES which are not shared by the two users affects the proximity of the users as well. Consider two users who have m and n FRACTURES, respectively ($m \gg 1$ and $n \gg 1$) and share only one FRACTURE with *FS* as 1. Then, according to Eq. (1), the two users are completely similar even if they have so many different FRACTURES. We address this problem by revising the denominator in Eq. (1) as $\frac{m+n}{2}$ so that the total number of FRACTURES of the two users are considered.

Second, the numerator in Eq. (1) treats all shared FRACTURES equally. However, if there exist different types of shared FRACTURES (e.g., we will define two types of shared FRACTURES, *Identical* FRACTURE and *Approximate* FRACTURE, in the next subsection), which contribute different importance to the proximity of the users, the numerator of Eq. (1) should be modified by introducing weights to different types of shared FRACTURES. Consequently, the numerator of Eq. (1) becomes: $\sum_{k=1}^{n} \omega_k \delta_k FS_k(u_1, u_2)$, where ω_k is the weight of the kth shared FRACTURE.

In the following, we first introduce the two types of shared FRACTURES and the way to weigh their contribution to the proximity of users (ω). Then, we define the similarity measure *FS*. Finally, the similarity measure *US* is formally defined.

3.1. Types of shared FRACTUREs

Recall that a FRACTURE discovered from historical web session trees is a set of web session subtrees (web directories). If two FRACTURES have the same web session subtrees, they certainly contribute to the similarity of the two users. We refer to such type of FRACTURES shared by two users as *Identical* FRACTURES.

Definition 5 (*Identical* FRACTURES). Let $X_1 = \{s_1, \ldots, s_m\}, X_2 = \{s_1, \ldots, s_n\}$ be two FRACTURES. Let $L(s)$ be the root path arriving at the root of web session subtree s. X_1 and X_2 is a pair of *Identical* FRACTURES, denoted as $X_1 = X_2$, if (1) $m = n$, and (2) $\forall i (1 \leqslant i \leqslant m), \exists j (1 \leqslant j \leqslant n)$ s.t. $L(s_i) = L(s_j)$ and (3) $\forall j (1 \leqslant j \leqslant n), \exists i (1 \leqslant i \leqslant m)$ s.t. $L(s_j) = L(s_i)$.

That is, two FRACTURES are *Identical* FRACTURES if there is a one-to-one mapping between the subtrees of the two FRACTURES and the corresponding subtrees have same labels. For example, consider the FRACTURES in Table 2. The two FRACTURES X_2^1 and X_3^1 is a pair of *Identical* FRACTURES. The two users u_2 and u_3 share another two pairs of *Identical* FRACTURES, $X_2^2 = X_3^2$ and $X_2^3 = X_3^3$.

It was observed in [8] that web directories (pages) are not randomly organized. Normally, a web directory (page) is stored in another web directory which semantically contains it. Thus, web session subtrees, whose labels have prefix relationships, are semantically related. For example, given two web session subtrees s_1 and s_2, suppose $L(s_1) = a/b$ and $L(s_2) = a/b/c$. Since $L(s_1)$ is a prefix of $L(s_2)$, the two web session trees are semantically related, e.g., s_1 semantically contains s_2. Then, a pair of FRACTURES whose corresponding subtrees are semantically related contribute to the similarity of users as well. Hence, besides *Identical* FRACTURES, we define another type of shared FRACTURES called *Approximate* FRACTURES as follows.

Definition 6 (*Approximate FRACTURES*). Let $X_1 = \{s_1, \ldots, s_m\}$ and $X_2 = \{s_1, \ldots, s_n\}$ be two FRACTURES. Let $L(s)$ be the root path arriving at the root of web session subtree s. X_1 and X_2 is a pair of *Approximate* FRACTURES, denoted as $X_1 \approx X_2$, if (1) $m = n$, and (2) $\forall i(1 \leqslant i \leqslant m)$, $\exists j(1 \leqslant j \leqslant n)$ s.t. $L(s_i)$ is a prefix of $L(s_j)$ or $L(s_j)$ is a prefix of $L(s_i)$ and (3) $\forall j(1 \leqslant j \leqslant n)$, $\exists i(1 \leqslant i \leqslant m)$ s.t. $L(s_j)$ is a prefix of $L(s_i)$ or $L(s_i)$ is a prefix of $L(s_j)$.

That is, two FRACTURES are *Approximate* FRACTURES if there is a one-to-one mapping between the subtrees of the two FRACTURES and labels of corresponding subtrees have prefix relationships. For example, the two FRACTURES X_1^2 and X_2^1 in Table 2 is a pair of *Approximate* FRACTURES.

We now explain how to assign different weights to different types of shared FRACTURES. Given two web session subtrees whose labels have prefix relationship, we define *Prefix Level* to measure the distance between the two subtrees.

Definition 7 (*Prefix Level*). Let s_i and s_j be two web session subtrees s.t. $L(s_i)$ is a prefix of $L(s_j)$. The length of the path $L(s_i)$, denoted as $|L(s_i)|$, is the number of edges traversed by the path. Then, the Prefix Level between s_i and s_j, denoted as $PL(s_i, s_j)$, can be computed as

$$PL(s_i, s_j) = |L(s_j)| - |L(s_i)|$$

For example, consider the two subtrees in the pair of *Approximate* FRACTURES, $X_1^2 \approx X_2^1$, in Table 2 again. Since the labels of the subtrees are $a/c/i$ and a/c, the *Prefix Level* between the two subtrees is 1. Note that, *Identical* FRACTURES can be viewed as a special case of *Approximate* FRACTURES from the perspective that the *Prefix Level* between each pair of corresponding subtrees of *Identical* FRACTURES is 0.

According to the observation in [8], the less the *Prefix Level* between two web session subtrees, the more semantically related the two subtrees. Given a pair of *Identical/Approximate* FRACTURES, the sum of the *Prefix Levels* between corresponding subtrees can be computed, which takes value in $[0, +\infty)$. Particularly, the sum is zero for a pair of *Identical* FRACTURES. Obviously, the less the sum, the more importance the shared FRACTURES contribute to the similarity of users. Thus, we use some monotonic decreasing function[3] to transform the sum to some weight value, ranging between (0,1]. Then, the weight assigned to a pair of shared FRACTURES can be computed as follows.

Definition 8 (*Weight*). Given a pair of shared FRACTURES $X_1 = \{s_1^1, \ldots, s_1^m\}$ and $X_2 = \{s_2^1, \ldots, s_2^m\}$ s.t. $PL(s_1^i, s_2^i) \geqslant 0 (1 \leqslant i \leqslant m)$. The weight assigned to the pair of FRACTURES, denoted as $\omega(X_1, X_2)$, is defined as,

$$\omega(X_1, X_2) = e^{-\sum_{i=1}^{m} PL(s_1^i, s_2^i)}$$

Thus, when the pair of shared FRACTURES are *Identical* FRACTURES, they are assigned the weight 1. The more similar the two FRACTURES in terms of their subtrees, the more weight will be assigned. For example, the weight assigned to the *Approximate* FRACTURES, $X_1^2 \approx X_2^1$, in Table 2 can be computed as follows. $\omega(X_1^2, X_2^1) = e^{-1} \approx 0.37$.

3.2. FRACTURE Similarity (FS)

A pair of shared FRACTURES may have different *FoC* and *SoC* values. Hence, we define the metric FRACTURE *Similarity* (*FS*) to measure the similarity of shared FRACTURES in terms of their *FoC* and *SoC* values.

Note that, although *FoC* is involved in the definition of *SoC*, the correlation between *FoC* and *SoC* is uncertain. That is, for different data set, the correlation between the two variables is different as well. Hence, we adopt the *Mahalanobis distance* to measure the distance between the *FoC* and *SoC* values of a pair of *Identical/Approximate* FRACTURES X_1 and X_2.

$$d(X_1, X_2) = (X_1 - X_2)^T \Sigma^{-1} (X_1 - X_2) = (FoC(X_1) - FoC(X_2) SoC(X_1) - SoC(X_2)) \Sigma^{-1} \begin{pmatrix} FoC(X_1) - FoC(X_2) \\ SoC(X_1) - SoC(X_2) \end{pmatrix}$$

[3] A variety of monotonic decreasing functions can be used here. For example, let Σ be the overall Prefix Levels between a pair of shared FRACTURES and ω be the weight. Then functions $\omega = \frac{1}{a\Sigma+1}$ and $\omega = a^{-\Sigma}$ can be used, where $a > 0$. The goodness of a function depends on the data. We evaluate the performance of different functions in Section 5.

where Σ^{-1} is the inverse of the covariance matrix of the complete set of *FoC* and *SoC* values of all users. Particularly, when *FoC* and *SoC* are uncorrelated for the data set, the *Mahalanobis distance* is actually the *Euclidean distance* between the *FoC* and *SoC* values of a pair of shared FRACTURES X_1 and X_2.

$$d(X_1,X_2)=\sqrt{(FoC(X_1)-FoC(X_2))^2+(SoC(X_1)-SoC(X_2))^2} \tag{2}$$

In order to define the FRACTURE *Similarity* based on the distance between *FoC* and *SoC* values of *shared* FRACTURES, we use a monotonic decreasing function to convert the distance to a similarity[4].

Definition 9 (*FRACTURE Similarity*). Let X_1 and X_2 be a pair of *Identical/Approximate* FRACTURES. Suppose $d(X_1,X_2)$ is the distance between the *FoC* and *SoC* values of X_1 and X_2. The FRACTURE *Similarity* (FS) of the pair of FRACTURES, denoted as $FS(X_1,X_2)$, is defined as,

$$FS(X_1,X_2)=\frac{1}{1+d(X_1,X_2)}$$

Since the *Mahalanobis distance* takes values in $[0,+\infty)$, *FS* ranges in $(0,1]$. The closer the *FoC* and *SoC* values of the two FRACTURES, the higher the *FS*. *FS* have the following two properties.

Property 1 (*Positivity Property*). $FS(X_1,X_2)=1$ *only if* $X_1=X_2$ $(0<FS\leqslant 1)$.

That is, given two FRACTURES X_1 and X_2, $FS(X_1,X_2)=1$ only if $FoC(X_1)=FoC(X_2)$ and $SoC(X_1)=SoC(X_2)$. Since *Mahalanobis distance* is a metric, which has the Positivity Property, $d(X_1,X_2)=0$ only if $FoC(X_1)=FoC(X_2)$ and $SoC(X_1)=SoC(X_2)$. As $FS=\frac{1}{1+d(X_1,X_2)}$ is a monotonic function, $FS=1$ only if $d=0$. Hence, the property holds for *FS*.

Property 2 (*Symmetry Property*). $FS(X_1,X_2)=FS(X_2,X_1)$ *for all* X_1 *and* X_2.

Again, since *Mahalanobis distance* is a metric, $d(X_1,X_2)=d(X_2,X_1)$. Thus, $FS(X_1,X_2)=\frac{1}{1+d(X_1,X_2)}=\frac{1}{1+d(X_2,X_1)}=FS(X_2,X_1)$.

Example 1. Consider the *FoC* and *SoC* of the FRACTURES in Table 2. The covariance matrix between the complete set of *FoC* and *SoC* is

$$\Sigma=\begin{pmatrix}0.0225 & -0.0161\\ -0.0161 & 0.0321\end{pmatrix}$$

For the pair of *Identical* FRACTURES, $X_2^3=X_3^3, d(X_2^3,X_3^3)=0$ and $FS(X_2^3,X_3^3)=1$, since the two FRACTURES have the same *FoC* and *SoC* values. While, for the pair of *Approximate* FRACTURES, $X_1^2\approx X_2^1$,

$$d(X_1^2,X_2^1)=\begin{pmatrix}-0.34 & 0.34\end{pmatrix}\Sigma^{-1}\begin{pmatrix}-0.34\\ 0.34\end{pmatrix}=5.6$$

Hence, $FS(X_1^2,X_2^1)=\frac{1}{1+5.6}\approx 0.15$.

3.3. User Similarity

After discussing the two limitations of the Eq. (1), the similarity measure *FS*, and the way to assign different weights to different types of shared FRACTURES, the *User Similarity* can be formally defined as follows.

Definition 10 (*User Similarity*). Let $u_1=\{X_1^1,X_1^2,\ldots,X_1^m\}$ and $u_2=\{X_2^1,X_2^2,\ldots,X_2^n\}$ be two web users that are represented as two sets of FRACTURES. Suppose there exists $k(0\leqslant k\leqslant m\leqslant n)$ s.t. $X_1^1=X_2^1$ or $X_1^1\approx X_2^1,\ldots,X_1^k=X_2^k$ or $X_1^k\approx X_2^k$. The *Similarity of Users*, denoted as $US(u_1,u_2)$, is defined as,

$$US(u_1,u_2)=\frac{2\sum_{i=1}^{k}\omega(X_1^i,X_2^i)FS(X_1^i,X_2^i)}{m+n}$$

If each pair of corresponding FRACTURES of the two users are *Identical* FRACTURES with the *FS* value 1, then *US* has the maximum value of 1. Otherwise, if the two users share no FRACTURE, *US* is 0. The Positivity Property and the Symmetry Property hold for *US* as well.

Property 3 (*Positivity Property*). $US(u_1,u_2)=1$ *only if* $u_1=u_2$. $(0\leqslant US\leqslant 1)$.

First, if $u_1=u_2$, which means u_1 and u_2 share all of their FRACTURES and all shared FRACTURES are *Identical* FRACTURES with *FS* as 1, then it is obvious $US(u_1,u_2)=1$. Second, if $u_1\neq u_2$, there exists at least one FRACTURE of one user which cannot be shared by the other user. Thus, for k,m,n in Definition 10, $k<m$ or $k<n$. Let us suppose $k<m$. k is either less than n or equal to n. If $k<n, k+k<m+n, k<\frac{m+n}{2}, US<1$, that is $US\neq 1$. If $k=n$, similarly, $k+k<m+k, 2k<m+n$, and $US\neq 1$. Thus, the Positivity Property holds for *US*.

[4] Again, other monotonic decreasing functions can be used as well. The performance of some alternative functions is evaluated in Section 5.

Property 4 (*Symmetry Property*). $US(u_1, u_2) = US(u_2, u_1)$ *for all* u_1 *and* u_2.

According to Definition 8, $\omega(X_1^i, X_2^i) = \omega(X_2^i, X_1^i)$. Moreover, it is shown above that *FS* has the Symmetry Property. Hence, $US(u_1, u_2) = US(u_2, u_1)$.

Example 2. Consider the users and their FRACTURES in Table 2 again. Since the two users u_2 and u_3 share three pairs of *Identical* FRACTURES, where each pair has the *FS* value $1, US(u_2, u_3) = \frac{2\times 3}{3+3} = 1$. On the other hand, users u_1 and u_2 share a pair of *Approximate* FRACTURES with FRACTURE similarity equal to 0.15 and assigned weight being 0.37. Then $US(u_1, u_2) = \frac{2\times 0.37\times 0.15}{3+3} = 0.0185$.

4. Framework of COWES

In this section, we describe the framework of COWES. Given a collection of web users, where each user is associated with a sequence of his historical web sessions, COWES generates the clusters of users in the follows phases:

- Phase I. From the historical web sessions of each user, we extract a set of FRACTURES, which will be treated as a vector of features for clustering.
- Phase II. Compute the similarity matrix between pairs of web users based on the defined similarity measure *US*.
- Phase III. Perform clustering on the generated similarity matrix of web users.

In [5], we have proposed two algorithms of mining FRACTURE patterns from a sequence of historical tree structures: *Apriori*-FRACTURE and *FPG*-FRACTURE. We discuss the Phases II and III in the following subsections.

4.1. Similarity computation

As the output of Phase I, each web user is represented as a set of FRACTURES, where each FRACTURE is a set of web session subtrees and associated with two values: *FoC* and *SoC*. We need to compute the similarity between each pair of users in the second phase.

Given two web users represented as two sets of FRACTURES, we first compute an optimal assignment of their FRACTURES so that the total weight between shared FRACTURES is maximized (i.e., the total *Prefix Level* between matched subtrees of the two FRACTURES is minimized). For example, suppose $u_1 = \{X_1^1\}$ where $X_1^1 = \{a/b, a/c\}$, and $u_2 = \{X_2^1, X_2^2\}$ where $X_2^1 = \{a/b, a/c/i\}$ and $X_2^2 = \{a/b/e, a/c/i\}$. Although X_1^1 is approximate with both X_2^1 and X_2^2, we assign X_1^1 to X_2^1 so that the total weight between shared FRACTURES of the two users is maximized. This problem is referred to as *Assignment Problem*, in the branch of optimization or operations research in mathematics, which can be defined as follows [25]:

> *Any problem involving minimizing the sum of* $C(a, b)$ *over a set P of pairs* (a, b) *where a is an element of some set A and b is an element of set B, and C is some function, under constraints such as "each element of A must appear exactly once in P" or similarly for B, or both.*

In our problem, *A* is the set of FRACTURES of one user, *B* is the set of FRACTURES of the other user, *C* is the total Prefix Level between matched subtrees from a pair of shared FRACTURES[5], and *P* is the assignment solution. Many algorithms have been developed to handle the assignment problem. Here, we use the Hungarian algorithm, which solves assignment problems in $O(n^3)$ time[6]. An implemented version of Hungarian algorithm can be downloaded from [26]. After getting the optimal assignment, the *FS* of shared FRACTURES and the *US* of the two users can be computed according to Definitions 9 and 10.

4.2. Cluster generation

After Phase II, we get a similarity matrix of web users. Many appropriate clustering methods can be used to generate the clusters. In general, the suitability of different methods are application-dependent. In our work, we compare the performance of 8 clustering methods provided by CLUTO (CLUstering TOolkit), which is a freely distributed clustering package created by Karypis [27]. In particular, we experimented with four partitional methods using different clustering criterion functions, three agglomerative methods using different merging schemes and one graph-based clustering algorithm.

We briefly describe the employed clustering methods in the following. For more details, please refer to the document provided by CLUTO [27]. The partitional clustering methods we used generate the desired *k*-way clustering solutions by performing a sequence of $k-1$ repeated bisections. During each step, the cluster is bisected so that the resulting 2-way clustering solution optimizes a particular clustering criterion function. The mathematical definitions of the four criterion

[5] Finding matched subtrees from a pair of *Approximate* FRACTURES is an assignment problem, too. For example, given a pair of *Approximate* FRACTURES, $\{a/b, a/b/c\}$ and $\{a/b, a/b/c/d\}$, the minimum sum of *Prefix Level* should be 1. The sum of *Prefix Level* can be 3 if assigning subtrees differently.

[6] Although the complexity is polynomial, the number *n* (the number of FRACTURES of a user or the number of subtrees of a FRACTURE) in our problem is usually not high.

functions used by us are shown in Eqs. (4)–(7), where k is the number of clusters, n_i is the number of users in the ith cluster, u and v are two individual users, C_i is the ith cluster and C represents clusters other than C_i.

$$I_1: \ maximize \ \sum_{i=1}^{k} \frac{1}{n_i} \left(\sum_{v,u \in C_i} sim(u,v) \right) \tag{3}$$

$$I_2: \ maximize \ \sum_{i=1}^{k} \sqrt{\sum_{v,u \in C_i} sim(u,v)} \tag{4}$$

$$E_1: \ minimize \ \sum_{i=1}^{k} n_i \frac{\sum_{v \in C_i, u \in C} sim(v,u)}{\sqrt{\sum_{v,u \in C_i} sim(u,v)}} \tag{5}$$

$$G_1: \ minimize \ \sum_{i=1}^{k} \frac{\sum_{v \in C_i, u \in C} sim(v,u)}{\sum_{v,u \in C_i} sim(u,v)} \tag{6}$$

The agglomerative clustering methods we used generate the desired k-way clustering solution based on the agglomerative paradigm, which starts with points (users) as individual clusters and, at each step, merge the closest pair of clusters. Three different merging schemes are used: *single-link*, *complete-link* and *UPGMA* (Unweighted Pair Group Method with Arithmetic mean) [10]. The three merging schemes define the cluster proximity based on the closest two points in different clusters, the farthest two points in different clusters, and the average pairwise proximities of all pairs of points from different clusters.

The graph-based clustering method computes the desired k-way clustering solution by first modeling the points (users) using a nearest-neighbor graph, and then splitting the graph into k-clusters using a min-cut graph partitioning algorithm.

5. Experimental results

In this section, we evaluate the performance of COWES via experiments on both synthetic and real data sets. We point out here that the novelty of COWES is not the clustering method, but the extraction of appropriate information from historical web sessions as a base for clustering and the similarity measures we defined to measure the proximity of web users in terms of their characteristics in usage data evolution. The first two steps of COWES are implemented in the Java programming language. All experiments are carried out on a Pentium IV 2.8 GHz PC with 512 MB memory. The operating system is windows 2000 professional.

5.1. Experiments on synthetic data

We conduct two experiments on the synthetic data. The first experiment is carried out to evaluate alternative similarity measures. The second experiment is used to study the performance of different clustering methods in our application.

5.1.1. Synthetic data generator

We implemented a synthetic data generator which generates FRACTURES for users with the following steps. Parameters used by the generator are shown in Table 3(a), where the third column shows the default values of the parameters.

Table 3
Parameters and datasets.

(a) Parameter list		
U	Number of web users	1000
w	Average number of FRACTURES per user	3
G	Number of FRACTURE groups	6
F	Average number of FRACTURES of each group	6
P	Number of FRACTURES	36
S	Average number of subtrees of each FRACTURE	3
N	Number of nodes of web session tree	1000
R	Range of overall Prefix Levels	0–5

Data	Feature parameters
(b) Dataset list	
D_1	$R = 0 - 1$
D_2	$R = 0 - 5$
D_3	$R = 0 - 10$
D_4	$w = 5, F = 4$
D_5	$w = 4, F = 5$
D_6	$w = 3, F = 6$
D_7	$w = 2, F = 7$

- First, we generate a general web session tree with the given number of nodes *N*. At a given node, the number of children is sampled uniformly at random from the range 0–10. For each child, the process is performed recursively if the depth of the tree is less than or equal to *R* or the total number of nodes reaches *N*.
- Then, we select subtrees from the general web session tree to compose *P* FRACTURES. The size of each FRACTURE is picked from a Poisson distribution with mean equal to *S*. The subtrees are sampled uniformly at random from the general web session tree.
- Then, we organize FRACTURES into FRACTURE groups. We sample the size of a FRACTURE group from a uniform distribution in the range of 0–*F*. The group to which a FRACTURE will be assigned is sampled uniformly at random from the range 0–*G*.
- Finally, we generate FRACTURES for web users. For each web user, we decide the number of FRACTURES from a Poisson distribution with mean equal to *w*. Each FRACTURE group is associated with a weight, which corresponds to the probability that it will be selected for a web user. The weight is picked from an exponential distribution with unit mean. Weights of each group is normalized so that the sum of weights is 1. The next group to be used for a user is chosen by tossing an *G*-sided weighted coin, where the weight of a side is the probability of picking the corresponding group. To model the phenomenon that all FRACTURES in a group do not always occur together, we associate to each group a corruption level, which is obtained from a normal distribution with mean 0.5 and variance 0.1. Then, when adding FRACTURES for a user, we keep dropping a FRACTURE from the group as long as a uniformly distributed random number between 0 and 1 is less than the corruption level of the group. To model the phenomenon of *Approximate* FRACTURE, for each FRACTURE, we replace some of its subtrees with its ancestor/descendant subtrees. The number of replaced subtrees is decided randomly from a uniform distribution in the range of 0 and the size of the FRACTURE. The Prefix Level between the original subtree and the replacing subtree is also sampled randomly from a uniform distribution in the range of 0 and *R*. Finally, we assign *FoC* and *SoC* to each selected FRACTURE from a uniform distribution in the range of [0,1].

In total, we use seven datasets in our experiments. As shown in Table 3(b), the first three datasets are generated with the default parameter values except that the value of parameter *R* is varied. We use the three datasets to conduct the first experiment which evaluates the alternative similarity measures. Hence, datasets generated with different *R* values will affect the weight of shared FRACTURES, which in turn affect the performance of alternative similarity measures. The last four datasets are generated by varying the parameters *w* and *F*. The four datasets are used in the second experiment which studies the performance of different clustering methods. Datasets generated with different *w* and *F* values exhibit different degrees of data dense, which affects the performance of clustering methods.

5.1.2. Result analysis

Before conducting any experiment to evaluate alternative similarity measures or clustering methods, we first carried out an initial experiment to see whether the natural number of clusters can be decided clearly. We then performed clustering on the dataset D_2 using the partitional methods with criterion functions I_1, I_2 and E_1. The plots of overall value of criterion functions versus the number of clusters are shown in Fig. 6. Since there exists no distinct knees or peaks in the overall value of criterion function, according to [19], the number of clusters cannot be decided clearly. Then, we simply generate 10 clusters in the remaining experiments.

As we mentioned in Section 3, different monotonic decreasing functions can be used in transforming a distance measure to a similarity measure. Hence, the similarity measure *US* can be defined alternatively. We now conduct experiments to study the performance of 6 alternative similarity measures as in Table 4. For example, for US_1, we use the function e^{-d} for both *FS* and ω. While, for US_2, we use the function e^{-d} for *FS* and $\frac{1}{1+d}$ for ω. In addition, we use similarity measure US_3 (and US_6) to consider *Identical* FRACTURES only. That is, ω is either 1 for *Identical* FRACTURES, or 0 for other situations.

We perform clustering with the six similarity measures on datasets D_1, D_2 and D_3. The results are shown in Fig. 7a–c, respectively. In Fig. 7, we use the two measures *IS* and *ES* to compare the clustering qualities. The *IS* refers to the average internal similarity of clusters, which can computed as follows.

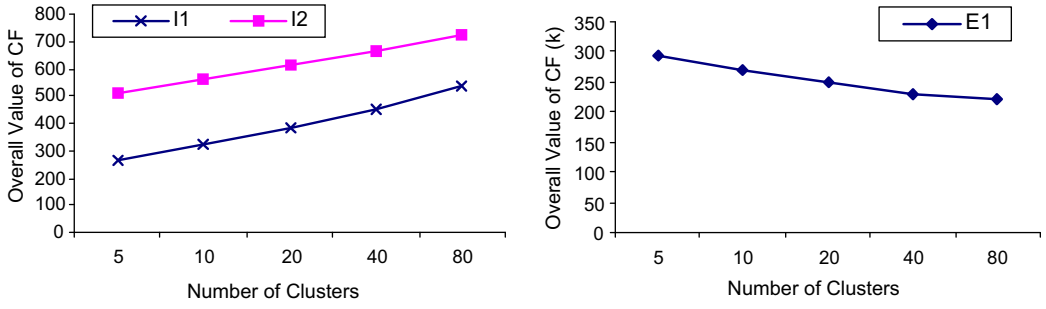

Fig. 6. Overall value of criterion functions versus number of clusters.

Table 4
Alternative similarity measures.

US	ω	*FS*
US_1	e^{-d}	e^{-d}
US_2	$\frac{1}{1+d}$	e^{-d}
US_3	0/1	e^{-d}
US_4	e^{-d}	$\frac{1}{1+d}$
US_5	$\frac{1}{1+d}$	$\frac{1}{1+d}$
US_6	0/1	$\frac{1}{1+d}$

		partitional				agglomerative			graph-based
		I1	I2	E1	G1	slink	clink	upgma	
US_1	IS	0.487	0.494	0.490	0.486	0.063	0.518	0.428	0.201
	ES	0.131	0.130	0.131	0.131	0.159	0.123	0.123	0.005
US_2	IS	0.516	0.506	0.511	0.502	0.067	0.535	0.445	0.205
	ES	0.139	0.139	0.139	0.139	0.169	0.131	0.133	0.005
US_3	IS	0.454	0.463	0.414	0.441	0.053	0.394	0.374	0.183
	ES	**0.104**	**0.104**	**0.116**	**0.107**	0.120	**0.125**	**0.101**	0.005
US_4	IS	0.514	0.514	0.517	0.514	0.068	0.534	0.452	0.209
	ES	0.142	0.142	0.123	0.141	0.172	0.133	0.134	0.006
US_5	IS	**0.537**	**0.538**	**0.533**	**0.536**	0.182	**0.553**	**0.529**	**0.215**
	ES	0.152	0.152	0.153	0.151	0.153	0.141	0.153	0.006
US_6	IS	0.456	0.481	0.445	0.459	0.056	0.434	0.396	0.187
	ES	0.112	0.110	0.117	0.114	0.132	0.127	0.107	0.005

(a) Performance of alternative similarity measures on D_1

		partitional				agglomerative			graph-based
		I1	I2	E1	G1	slink	clink	upgma	
US_1	IS	0.319	0.319	0.301	0.327	0.067	0.275	0.301	0.150
	ES	0.090	0.090	0.091	0.079	0.075	0.079	0.060	**0.004**
US_2	IS	0.373	0.369	0.358	0.368	0.015	0.352	0.353	0.149
	ES	0.108	0.111	0.112	0.114	0.102	0.096	0.077	**0.004**
US_3	IS	0.277	0.261	0.267	0.268	0.049	0.262	0.255	0.115
	ES	**0.064**	**0.062**	**0.075**	**0.062**	0.047	**0.073**	0.040	0.005
US_4	IS	0.323	0.322	0.319	0.332	0.067	0.276	0.266	0.133
	ES	0.092	0.090	0.103	0.081	0.076	0.085	0.064	**0.004**
US_5	IS	**0.381**	**0.381**	**0.371**	**0.381**	0.381	**0.376**	**0.376**	**0.152**
	ES	0.116	0.116	0.118	0.119	0.119	0.119	0.077	**0.004**
US_6	IS	0.266	0.263	0.268	0.270	0.049	0.265	0.262	0.116
	ES	0.065	**0.062**	**0.075**	**0.062**	0.047	**0.073**	**0.038**	0.005

(b) Performance of alternative similarity measures on D_2

		partitional				agglomerative			graph-based
		I1	I2	E1	G1	slink	clink	upgma	
US_1	IS	0.286	0.286	0.284	0.282	0.186	0.226	**0.311**	0.109
	ES	0.085	0.085	0.085	0.082	0.057	0.091	0.078	**0.004**
US_2	IS	0.345	0.286	0.344	0.341	0.114	0.326	0.300	0.127
	ES	0.112	0.085	0.112	0.108	0.090	0.110	0.105	**0.004**
US_3	IS	0.225	0.217	0.215	0.220	0.209	0.256	0.213	0.092
	ES	**0.051**	**0.051**	**0.051**	**0.051**	0.040	**0.062**	0.047	**0.004**
US_4	IS	0.290	0.290	0.287	0.286	0.186	0.229	0.286	0.108
	ES	0.087	0.087	0.086	0.084	0.059	0.089	0.073	**0.004**
US_5	IS	**0.356**	**0.356**	**0.355**	**0.352**	0.114	**0.342**	0.277	**0.129**
	ES	0.117	0.117	0.117	0.113	0.095	0.108	0.107	**0.004**
US_6	IS	0.226	0.218	0.213	0.221	0.209	0.312	0.220	0.092
	ES	0.052	**0.051**	**0.051**	0.052	0.037	**0.062**	**0.044**	**0.004**

(c) Performance of alternative similarity measures on D_3

Fig. 7. Similarity matrix ordered by clustering results.

$$IS = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{n_i}\left(\sum_{v,u\in C_i} sim(v,u)\right) \tag{7}$$

while, *ES* refers to the average external similarity of clusters, which is defined as Eq. (9).

$$ES = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{n_i}\sum_{v\in C_i, u\in C} sim(v,u) \tag{8}$$

For a good clustering, the *IS* should be large while the *ES* should be small. From the experimental results in Fig. 7, we have the following observations:

- There is a tradeoff between *IS* and *ES*. That is, similarity measures which works well in *IS* do not perform well in *ES*, vice versa. For example, US_5 can achieve high *IS* value, while its *ES* is quite high as well. Similarly, although US_3 (or US_6) generates low *ES*, it sacrifices too much in *IS*. Generally, US_4, which is the User Similarity measure defined in Definition 10, generates good *IS* without sacrificing too much in *ES*.
- From each table, it can be discovered that the performance of a similarity measure is independent of the clustering methods. That is, no matter what kind of clustering method is used, the performance of a similarity measure is consistent (note that, the agglomerative clustering method which uses the simple-link merging scheme is not a suitable method here because the generated *IS* is lower than *ES* in many cases).
- From the tables, it can be noticed that when the dataset changes so that the overall Prefix Levels between *Approximate* FRACTURES increases, the goodness of the similarity measures is consistent. For example, Fig. 8a and b, respectively plot the *IS* and *ES* values generated by the partitional clustering method with the I_1 function and the six similarity measures on the three datasets. We learned that US_2 and US_5 always achieve the highest *IS* (*ES*) while US_3 and US_6 persistently have the lowest *IS* (*ES*).

We now describe the second experiment conducted to evaluate the performance of different clustering methods. Different methods have different performance with respect to characteristics of the datasets. In our experiment, we use the four synthetic datasets D_4, D_5, D_6 and D_7, which are generated by using different values for parameters *w* and *F*. According to the way we generate synthetic datasets, the higher the *w* and the lower the *F*, the more overlapped the user groups. Consequently, the dataset D_4 is more overlapped than D_7. Fig. 9 shows the results of the experiments, where the defined similarity measure US_4 is used. It can be observed that the partitional clustering method using the criterion function I_1 usually achieves the highest *IS* value while the graph-based clustering method works best in *ES*. Figs. 10 and 11 show the gray scale images of the same similarity matrix ordered by the clusters generated by the eight algorithms. The shade of each point in the images represents the value of the corresponding entry in similarity matrix. In extreme cases, white and black correspond to the similarity values of 1 and 0, respectively. Hence, for a good clustering, the rectangles on the diagonal should be as white as possible as they represent the web users in the same clusters, while the remaining areas should be as black as possible. We observed that generally, the partitional clustering methods perform well not only in achieving the better accuracy but also in controlling the balance of the cardinality of the clusters.

5.2. Experiments on real data

We also conducted three experiments on real-life data. In the first experiment, we compare the accuracy of COWES and an existing algorithm [20] which clusters tree-structured data without considering its evolution. The second experiment is con-

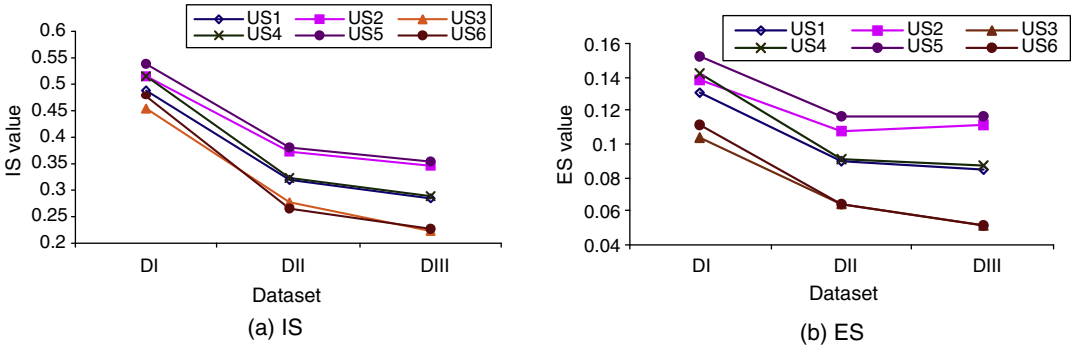

Fig. 8. *IS* and *ES* generated by partitional I_1 on different datasets.

		partitional				agglomerative			graph-based
		I1	I2	E1	G1	slink	clink	upgma	
D_4	IS	0.268	0.275	0.270	0.288	0.014	0.238	**0.311**	0.091
	ES	0.134	0.120	0.121	0.124	0.139	0.122	0.106	**0.007**
D_5	IS	**0.402**	0.379	0.371	0.385	0.182	0.298	0.252	0.156
	ES	0.201	0.183	0.184	0.142	0.023	0.158	0.099	**0.010**
D_6	IS	**0.363**	0.327	0.320	0.309	0.063	0.289	0.276	0.138
	ES	0.117	0.122	0.121	0.115	0.094	0.112	0.083	**0.006**
D_7	IS	**0.417**	0.411	0.364	0.408	0.054	0.327	0.382	0.229
	ES	0.079	0.079	0.076	0.079	0.107	0.071	0.068	**0.003**

Fig. 9. Performance of different clustering methods on different datasets.

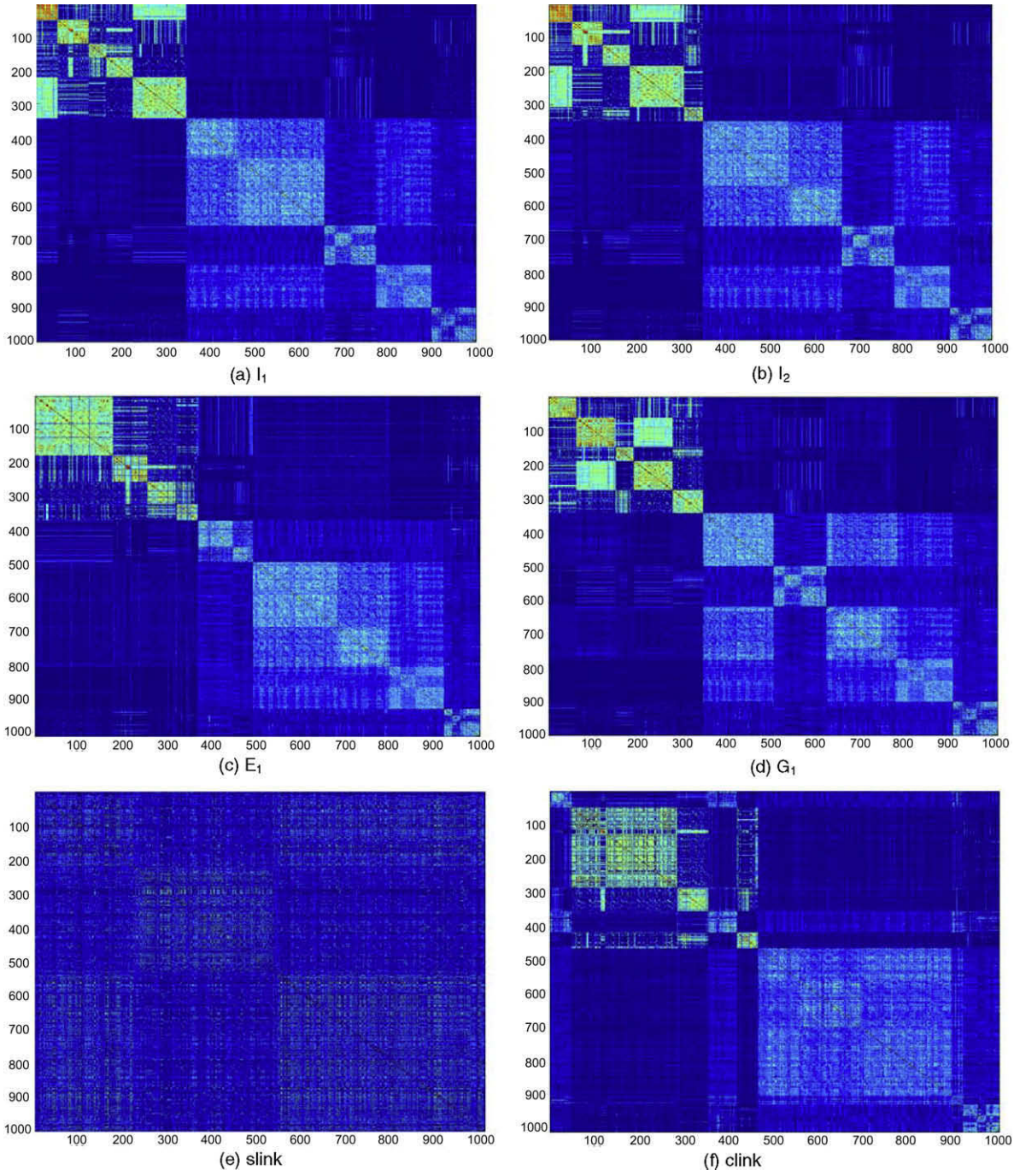

Fig. 10. Similarity matrix ordered by clustering results I.

ducted to compare the effectiveness of our similarity measure and alternative similarity measures. The third experiment shows the performance variation with respect to the parameters of COWES.

5.2.1. Datasets

The real-life datasets are collected from Internet Traffic Archive (http://ita.ee.lbl.gov), sponsored by ACM SIGCOMM. We use the trace that contains a day's worth of all HTTP requests to the EPA www server located at Research Triangle Park, NC. In considering the evolution of web usage data, the requests of a host are grouped with a time interval of one hour. All the requests of all 2333 hosts in the trace form the Dataset I. In order to study the novelty of the knowledge that can be discovered by COWES, we collect the requests of 57 hosts that browse the subtree with two paths, "/docs/whatsNew.html" and "/docs/

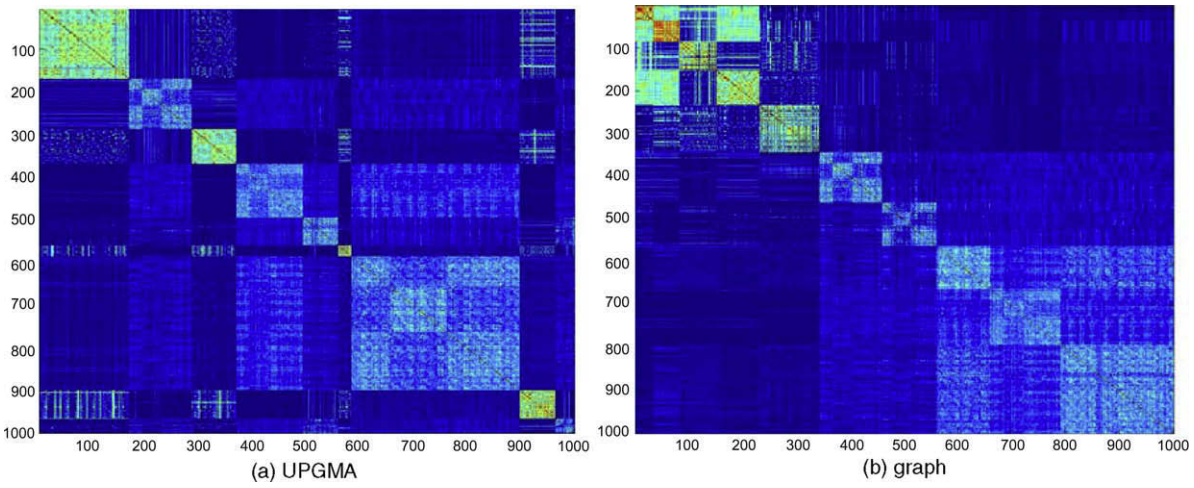

Fig. 11. Similarity matrix ordered by clustering results II.

Num of Clusters	Dataset I				Num of Clusters	Dataset II			
	COWES		*STRUCTURE*			*COWES*		*STRUCTURE*	
	IS	ES	IS	ES		IS	ES	IS	ES
5	**0.36**	0.013	0.09	**0.007**	3	**0.67**	**0.24**	0.35	**0.24**
6	**0.36**	0.014	0.08	**0.006**	4	**0.72**	0.39	0.37	0.24
7	**0.38**	0.017	0.21	**0.006**	5	**0.73**	0.34	0.38	0.23
8	**0.39**	0.019	0.18	**0.008**	6	**0.72**	0.32	0.40	0.22

Fig. 12. Comparison of clustering algorithms.

whatsHot.html" to form the Dataset II. Since hosts in the Dataset II are similar in their requests, without considering the evolutionary characteristics of the requests, they may not be distinguished by existing cluster algorithms. We study to see whether COWES can generate clusters of high quality based on evolutionary features of the requests.

5.2.2. Result analysis

We first conduct experiments to evaluate the accuracy of COWES on real-life data. The defined similarity measure US_4 and the partitional clustering method with criterion function I_1 are used. Similarly, we use *IS* and *ES* to evaluate the quality of the clustering results. We compare the results of COWES with those of an existing clustering method [20], referred to as STRUCTURE in Fig. 12, which clusters hosts based on their a day's worth all HTTP requests without considering the evolution of the requests. The reason we compare COWES against STRUCTURE instead of other classical web user clustering methods [23,21] is as follows. Classical web user clustering methods usually represent each web session as a bag pages. However, STRUCTURE considers web sessions as tree structures, which capture more semantic similarity between pages. For example, given two web sessions, $\{a/b, a/c\}$ and $\{a/b/d, a/c\}$, STRUCTURE defines similarity based on shared edges. Then, the Jaccard similarity between the two web sessions is 2/3. If we consider the two web sessions as bags of pages, the Jaccard similarity between the two sessions is 1/3. Since COWES represents web sessions as trees also, it is fair to compare COWES with STRUCTURE instead of other classical web user clustering algorithms.

The experimental results are shown in Fig. 12. The *DoC*, *FoC* and *SoC* thresholds for FRACTURE mining are fixed at 0.5, 0.2 and 0.6, respectively. On Dataset I, the number of generated clusters ranges from 5 to 8. Since there are fewer hosts in Dataset II, we generate 3 through 6 clusters. We observed that for Dataset I, the accuracy achieved by COWES is compatible with respect to that of STRUCTURE. However, for Dataset II, COWES achieves much better *IS* and competitive *ES*. The reason that STRUCTURE cannot perform as well as COWES on Dataset II is as follows. STRUCTURE clusters hosts based on all pages accessed by each host in one day. Hence, it cannot distinguish hosts which accessed similar pages (e.g., "/docs/whatsNew.html" and "/docs/whatsHot.html"). Nevertheless, COWES clusters hosts based on the evolutionary features of web sessions. Although hosts accessed similar pages, COWES can distinguish them if they share similar FRACTURES. For example, we observed that in all the clustering results on Dataset II when the cluster number is varied from 3 to 6, the two hosts, "e659229.boeing.com" and "keyhole.es.dupont.com", are consistently clustered together. After analyzing the historical web sessions of the two hosts, we observed that although they share similar pages with other hosts in Dataset II, they share a particular pair of FRACTURES $\{EPA - AIR/1995, docs/EPA - AIR/1995\}$[7], whose sequential *DoC* values are shown in Fig. 14a and b, respectively. For "e659229.boeing.com", *FoC* of the FRACTURE is 0.22 and *SoC* of the FRACTURE is 1.0. For "keyhole.es.dupont.com", the *FoC* and *SoC*

[7] The two hosts share some other FRACTURES which are shared by other hosts in the cluster also. For clarity, we do not show them all.

Num of Clusters	Dataset I				Num of Clusters	Dataset II			
	US_4		US_6			US_4		US_6	
	IS	ES	IS	ES		IS	ES	IS	ES
5	**0.36**	**0.013**	0.21	0.015	3	**0.67**	0.24	0.59	**0.21**
6	**0.36**	**0.014**	0.22	0.015	4	**0.72**	0.39	0.67	**0.34**
7	**0.38**	**0.017**	**0.38**	0.019	5	**0.73**	0.34	0.65	**0.31**
8	**0.39**	**0.019**	0.30	0.024	6	**0.72**	0.32	0.65	**0.29**

Fig. 13. Comparison of similarity measures.

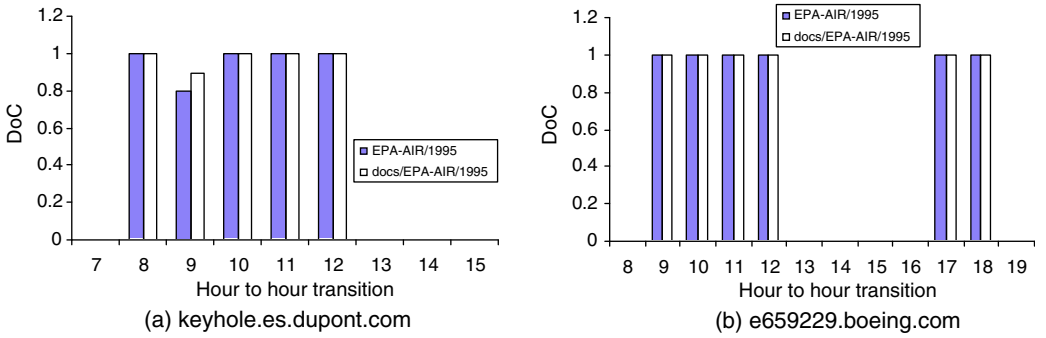

Fig. 14. Shared FRACTURES of clustered hosts.

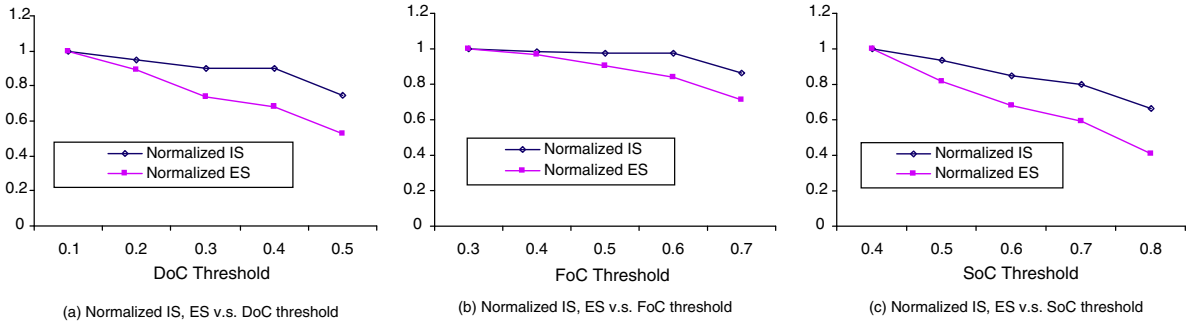

Fig. 15. Performance variation with respect to thresholds.

values of the FRACTURE are 0.26 and 1.0. COWES successfully distinguished them from other hosts in Dataset II because they frequently vary their information needs under the two web session subtrees together.

We conduct another experiment to evaluate the effectiveness of our similarity measure in the real-life data. In particular, we compare the two similarity measures US_4 and US_6 described in the previous subsection. The former is the defined similarity measure while the latter considers *Identical* FRACTURES only. The experimental results are shown in Fig. 13. We noticed that although both similarity measures have similar performance in *ES*, the defined similarity measure works much better in *IS*.

We then examine how the performance of COWES varies with respect to the variations of *DoC*, *FoC* and *SoC* thresholds. The experiments are conducted on the Dataset I. The cluster number is fixed at 5 (Note that, our objective in this experiment is to study the variation trend of the performance. Hence, the number of clusters will not affect the trend.). When varying one of the thresholds, the other two are set as default values. The default values of the three thresholds are 0.4, 0.6, and 0.7, respectively (see Fig. 14). The normalized *IS* and *ES* values with respect to each threshold are shown in Fig. 15a–c, respectively. We observed that when the threshold values are increased, both the *IS* and *ES* values decreased. The reason is that when the threshold values are low, more FRACTURE patterns can be discovered for each web user. Hence, users probably share more FRACTURES and both the average internal similarity and average external similarity increase. On the contrary, when the thresholds are high, fewer FRACTURES can be discovered and web users probably share fewer FRACTURES. Consequently, both *IS* and *ES* values decrease. We also observed that our approach on this dataset is more sensitive to the thresholds of *DoC* and *SoC* than the threshold of *FoC*. The reason is that even if the threshold of *FoC* is lowered, it is not necessary more FRACTURES can be found for each user, because the threshold of *SoC* is fixed. Generally, the performance of COWES depends on the thresholds. Users of COWES need to adjust the thresholds until acceptable *IS* and *ES* values are reached.

6. Related work

Clustering **O**f **W**eb users is an important task of Web Usage Mining. Existing work on web user clustering usually extract access patterns of users from web server log files and organize them into web sessions. Xiao and Zhang [23] clustered web user sessions based on various similarity measures, such as the number of shared web pages, the frequency of accessing the shared web pages, the visiting time spent on shared pages and the order of visiting shared paged. Rather than clustering web users based on web sessions directly, Fu et al. [8] first generalized the sessions so that pages representing the similar semantics are collapsed. This approach reduces the dimension of clustering feature significantly. Wang and Zaiane [21] also cluster web users based on snapshots of web sessions. They represented web sessions as vectors of encoded page IDs and then a clustering algorithm handling categorical data was employed. The critical difference between existing works on clustering web users and our effort is that we address the dynamic nature of web usage data. Existing approaches cluster web users by the snapshots of web sessions, whereas we cluster web users by their historical web sessions. Furthermore, our similarity measure is different from existing ones. We measure the proximity of web users based on the characteristics of their usage data evolution. Existing approaches measure the likeness between web users based on the information in snapshot web sessions.

Recently, the evolutionary feature of data was addressed by some data mining work which aim to maintain consistent mining results over time. Chakrabarti et al. [4] proposed to cluster timestamped data such that a clustering at each timestep is similar to the clustering at the previous timestep. Furthermore, each clustering should accurately reflect the data arrived during that timestep. The dynamic nature of web access patterns were also observed by Nasraoui et al. [16,15]. As an alternative to locking the state of the web access patterns in a frozen state depending on when the web log data was collected and preprocessed, they proposed to consider the web usage data as a reflection of a dynamic environment. These approaches are different in the way that they cope with the dynamic nature of web usage data by designing evolutionary approaches to update knowledge, whereas we mine the historical web usage data to obtain novel knowledge.

7. Conclusions

In this paper, we have presented a novel web user clustering method called COWES. To the best of our knowledge, this is the first work which clusters web users based on the evolution of their web usage data. In order to capture the evolutionary characteristics of web usage data, we mine evolutionary patterns, FRACTURE, from users' historical web sessions and use discovered patterns as the clustering features. We identified two types of FRACTURES that can be shared by users and assigned different weights to them to distinguish their contribution to the proximity of users. We then define the similarity between users in terms of their shared FRACTURES.

Various clustering methods are employed to generate clusters of web users. The experimental results show that (*i*) COWES is effective in discovering high quality clusters when users accessing similar pages exhibit different characteristics in usage evolution; (*ii*) the defined similarity measure works well in achieving high average internal similarity of clusters, without sacrificing much in average external similarity of clusters; (*iii*) partitional clustering methods are preferable to agglomerative clustering methods and graph-based methods in our application.

References

[1] A. Amiri, S. Menon, Efficient scheduling of internet banner advertisements, ACM TOIT 3 (4) (2003) 334–346.
[2] C. Buchwalter, M. Ryan, D. Martin, The state of online advertising: data covering 4th Q 2000, in TR Adrelevance, 2001.
[3] P. Cao, S. Irani, Cost-aware WWW proxy caching algorithms, in: Proc. of USENIX SITSY, 1997.
[4] D. Chakrabarti, R. Kumar, A. Tomkins, Evolutionary clustering, in: KDD, 2006, pp. 554–560.
[5] L. Chen, S.S. Bhowmick, L.T. Chia, FRACTURE mining: mining frequently and concurrently mutating structures from historical XML documents, in: Data and Knowl. Eng., vol. 59, 2006, pp. 320–347.
[6] L. Chen, S.S. Bhowmick, J. Li, COWES: clustering web users based on historical web sessions, in: Proc. of DASFAA, 2006.
[7] R. Cooley, B. Mobasher, J. Srivastava, Data preparation for mining world wide web browsing patterns, in: Knowledge and Information Systems, vol. 1, 1999.
[8] Y. Fu, K. Sandhu, M. Shih, A generalization-based approach to clustering of web usage sessions, in: Proc. of WEBKDD'99, 1999.
[9] X. Huang, A. An, N. Cercone, G. Promhouse, Discovery of interesting association rules from livelink web log data, in: Proc. of ICDM, 2002.
[10] Anil K. Jain, Richard C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
[11] N. Labroche, N. Monmarche, G. Venturini, Web sessions clustering with artificial ants colonies, in: Proc. of www, 2003.
[12] T. Li, Q. Yang, K. wang, Classification pruning for web-request prediction, in: Proc. of www, 2001.
[13] B. Mobasher, R. Cooley, J. Srivastava, Creating adaptive web sites through usage-based clustering of URLs, in: Proc. of IEEE KDEX workshop, 1999.
[14] B. Mobasher, H. Dai, T. Luo, M. Nakagawa, Effective personalization based on association rule discovery from web usage data, in: Proc. of wIDM, 2001.
[15] O. Nasraoui, C. Cardona-Uribe, C. Rojas-Coronel, Techno-streams: tracking evolving clusters in noisy data streams with a scalable immune system learning model, in: Proc. of ICDM, 2003.
[16] O. Nasraoui, R. Krishnapuram, H. Frigui, A. Joshi, One step evolutionary mining of context sensitive associations and web navigation patterns, in: Proc. of SDM, 2002.
[17] D. Simovici, N. Singla, M. Kuperberg, Metric incremental clustering of nominal data, in: ICDM, 2004, pp. 523–526.
[18] J. Srivastava, R. Cooley, M. Deshpande, P.-N. Tan, Web usage mining: discovery and applications of usage patterns from web data, in: SIGKDD Explorations, vol. 1 (2), 2000, pp. 12–23.
[19] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Addison wesley, 2006.
[20] L. Wang, D.w.-L. Cheung, N. Mamoulis, S.-M. Yiu, An efficient and scalable algorithm for clustering XML documents by structure, in: IEEE TKDE, vol. 16 (1), 2004, pp. 82–96.
[21] W. Wang, O.R. Zaiane, Clustering web sessions by sequence alignment, in: Proc. of DEXA, 2002.

[22] D. Widyantoro, T. Ioerger, J. Yen, An incremental approach to building a cluster hierarchy, in: ICDM, 2002, pp. 705–708.
[23] J. Xiao, Y. Zhang, Clustering of web users using session-based similarity measures, in: Proc. of ICCNMC'01, 2001.
[24] Q. Yang, H.H. Zhang, T. Li, Mining web logs for prediction models in WWW caching and prefetching, in: Proc. of ACM SIGKDD, 2001.
[25] Assignment Problem Definition. <http://www.definethat.com/define/3622.htm>.
[26] Munkres' (Hungarian) Algorithm. <http://www.spatial.maine.edu/~kostas/dev/soft/munkres.htm>.
[27] CLUTO 2.1.1: Software for Clustering High-Dimensional Datasets. <http://www.cs.umn.edu/~karypis>.

Ling Chen received her Ph.D in Computer Engineering at Nanyang Technological University, Singapore, in 2007. She is currently a post-doc researcher at L3S Research Center, University of Leibniz, Germany. Her research interest includes data mining, machine learning, social web mining, collaborative filtering and web personalization.

Sourav S. Bhowmick is an Associate Professor in the School of Computer Engineering, Nanyang Technological University and the Director of Centre for Advanced Information Systems (CAIS). He is currently Visiting Associate Professor at the Biological Engineering Division, Massachusetts Institute of Technology (MIT), USA. He also holds the position of Singapore-MIT Alliance (SMA) Fellow in Computation and Systems Biology program (2005–2008). He received his Ph.D. in computer engineering in 2001. His current research interests include XML data management, systems biology data management, web data management, and data mining. He has published more than 100 papers in major international database and data mining conferences and journals such as VLDB, IEEE ICDE, ACM WWW, ACM SIGMOD, ACM SIGKDD, ACM CIKM, ER, PAKDD, IEEE TKDE, ACM CS, Information Systems, and DKE. He is serving as a PC member of various database conferences and workshops and reviewer for various database journals. He is also serving as a program chair/co-chair of several international workshops in biological and XML data management. He is a member of the editorial boards of several international journals. He has been tutorial speaker in several international conferences such as ER 2006, APWeb 2008, and WAIM 2008. He has co-authored a book entitled Web Data Management: A Warehouse Approach" (Springers Verlag, October 2003). He is a member of ACM and an affiliate member of IEEE.

Wolfgang Nejdl (born 1960) has been full professor of computer science at the University of Hannover since 1995. He received his M.Sc. (1984) and Ph.D. degree (1988) at the Technical University of Vienna, was assistant professor in Vienna from 1988 to 1992, and associate professor at the RWTH Aachen from 1992 to 1995. He worked as visiting researcher/ professor at Xerox PARC, Stanford University, University of Illinois at Urbana-Champaign, EPFL Lausanne, and at PUC Rio.
He heads the Distributed Systems Institute/ Knowledge Based Systems (http://www.kbs.uni-hannover.de/) as well as the L3S Research Center (http://www.l3s.de/), and does research in the areas of technology-enhanced learning, semantic web technologies, peer-to-peer information systems, search and information retrieval, databases and artificial intelligence. Recent projects in the L3S context include the PHAROS Integrated Project on audio-visual search, the OKKAM IP focusing on entities on the Web, and the Digital Library EU project LiWA, coordinated by L3S, which investigates Web archive management and advanced search in such an archive.
He published more than 180 scientific articles, as listed at DBLP, and has been program chair, program committee and editorial board member of numerous international conferences and journals, see also http://www.kbs.uni-hannover.de/~nejdl/.