# FRECLE Mining: Discovering Frequent Semantic Tree Cluster Sequences from Historical Tree Structured Data

Ling Chen and Sourav S Bhowmick

**Abstract**

Mining frequent trees is very useful in domains like bioinformatics, web mining, mining semi-structured data, and so on. Existing techniques focus on finding "structural" patterns and ignores the "semantics" that may be associated with the subtrees. In this paper we proposal an algorithm to mine a novel pattern called ***fr**equent **s**emantic tree **cl**uster **s**equences* (FRECLE), which captures the frequent sequential association between different semantics of tree-structured data. Given a *semantic tree* sequence database, the algorithm first categorizes each *semantic tree* to a *semantic cluster*. Next, FRECLE patterns are discovered from the semantic cluster sequences by adopting an existing frequent sequential pattern mining algorithm. FRECLE patterns are beneficial in applications where the knowledge of semantic association is significant, such as XML query caching, prefetching XML data, and web users clustering. Specifically, we show how our proposed FRECLE mining framework can be used for designing optimal XML query cache replacement strategy. Finally, by reporting the performance of our algorithm and caching strategy through extensive experiments with both synthetic and real datasets, we show the effectiveness and usefulness of FRECLE mining.

**Index Terms**

Semantic trees, semantic clusters, frequent cluster sequences mining, XML, cache replacement strategy, association rules.

L. Chen is with the L3S Research Center, University of Hannover, Germany.

S. S. Bhowmick is with the School of Computer Engineering, University of Nanyang Technological University, Singapore

# I. INTRODUCTION

Mining tree-structured data has gained tremendous interest in recent times due to the widespread occurrence of tree patterns in applications like bioinformatics, web mining, semi-structured data mining, and so on. Particularly, the *frequent tree mining* is the most well researched topic due to the importance of finding common subtree patterns. Several algorithms for mining frequent trees have been proposed recently, which include TreeMiner [30], FreqT [2], TreeFinder [22], PathJoin [27], Chopper [23], and CMTreeMiner [8]. The basic idea is to extract subtrees which occur frequently among a set of trees or within an individual tree. For example, consider a database of XML queries as shown in Figure 1. Suppose each user $i$ issues a sequence of queries over time as shown in Figure 1(a). One might like to mine frequently occurring "structural" query patterns, i.e., subtrees, that appear in this collection [28]. It has been reported that such patterns are useful in designing cache replacement strategy for XML queries [29].

While the above frequent tree pattern mining techniques have been innovative and powerful, our initial investigation revealed that majority of the existing approaches focus on finding "structural" patterns ignoring the "semantics" that may be associated with the substructures. For example, the query trees in Figure 1 have certain semantics. $S_1$ and $S_3$ are about book title, and $S_2$ and $S_4$ are about authors. We refer to such trees that are semantically meaningful as *semantic trees*. Existing frequent tree mining techniques typically focus on finding frequent structure but not frequent "semantics". Knowledge of frequent semantics and association between them can be useful in several applications. For example, if we know that whenever a user queries about the information of book title, he/she is very likely to issue another query about the authors, then we can use such information to design optimal XML cache replacement strategy. We can also use such semantically-enriched knowledge to prefetch data while processing XML queries.

Observe that this is a challenging problem as two semantic trees may *not* be structurally identical but they may have similar semantics. For example, although $S_2$ and $S_4$ in Figure 1(b) have similar semantics, they are not structurally identical. Furthermore, there may exist interesting sequential associations between two different semantics as highlighted above. Consequently, existing frequent tree mining algorithms cannot be directly applied on the collection of semantic trees to discover frequent semantics and associations between them.

Given a database of semantic tree sequences (e.g., XML query sequences in Figure 1), in this

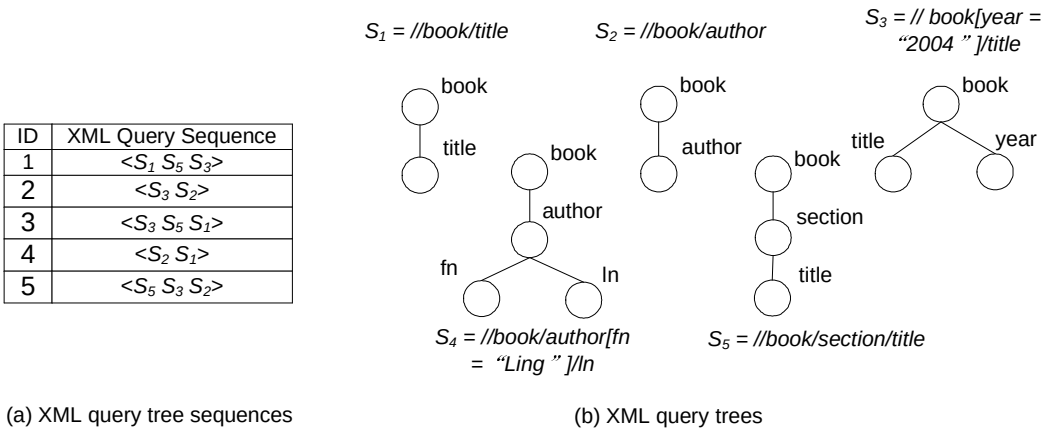


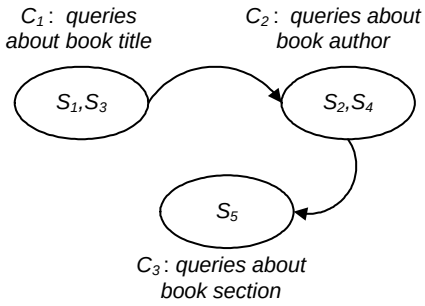

Fig. 1. Example of XML query tree sequences.

paper, we introduce a novel pattern called ***fr**equent **se**mantic tree **cl**uster **se**quences* (FRECLE). A *semantic tree cluster* contains a set of semantic trees that share similar semantics. Hence, each tree in a semantic tree sequence can be assigned to a semantic cluster that is "closest" to the semantics of the tree. In this paper, we present an efficient algorithm to discover FRECLE patterns by assigning each semantic tree to a cluster.

### A. Overview

Our proposed FRECLE mining algorithm consists of two major phases: the *semantic tree clustering* phase and the FRECLE *pattern discover* phase. In the first phase, a novel *cluster-centered* strategy is proposed to construct semantic clusters from a set of semantic trees by measuring the *semantic cohesiveness* of clusters. At the end of this phase, the semantic tree sequences are transformed to corresponding semantic cluster sequences. The goal of the second phase is to extract frequent cluster sequences by scanning the transformed database. We illustrate these two phases informally with an example.

Given the database of XML query tree sequences in Figure 1, we categorize each query tree to certain semantic cluster by grouping query trees representing similar semantics. For example, the query trees of $S_1$ and $S_3$ will be grouped together as they represent the information on *book title*. Suppose the clustering results of the five query trees in Figure 1 are shown in Figure $2(a)$, where the cluster $C_1$ represents the information of *book title*, the cluster $C_2$ represents the information about *book author*, and the cluster $C_3$ represents the information about *book section*. Then, the




(a)

| ID | Semantic Cluster Sequence |
|---|---|
| 1 | <$C_1$ $C_3$ $C_1$> |
| 2 | <$C_1$ $C_2$> |
| 3 | <$C_1$ $C_3$ $C_1$> |
| 4 | <$C_2$ $C_1$> |
| 5 | <$C_3$ $C_1$ $C_2$> |

(b)

Fig. 2. Historical XML query trees.

database of XML query tree sequences can be transformed into a set of cluster sequences as depicted in Figure $2(b)$. FRECLE patterns can then be discovered from the transformed database as frequent subsequences of tree clusters by adopting a traditional frequent sequential pattern mining technique [13]. For example, suppose the threshold of *support* be $0.4$. Then, the sequence $\langle C_1, C_2 \rangle$ in Figure $2(b)$ will be discovered as a FRECLE pattern because its support is $2/5 \geq 0.4$.

Observe that FRECLEs capture the frequent association between different semantics represented by different semantic tree clusters. Hence, such patterns are beneficial in applications where the knowledge of semantic association is significant, such as XML query cache replacement strategy, prefetching XML data, and web users clustering. For instance, the discovered FRECLE pattern in the above example indicates that when a user formulates query about the information of book title, he/she is very likely to issue another query about the author of the book. As we shall see in Section V, such knowledge can be used in designing optimal XML query caching replacement strategies. For example, we can delay the eviction of answers to queries about book author, if they exist in cache already, once the queries about book title are issued.

### *B. Contributions*

The major contributions of this paper can be summarized as follows.

- We introduce an approach that, to the best of our knowledge, is the first one to discover novel knowledge from temporal sequences of semantic trees. Specifically, in this paper we focus on discovering frequent semantic tree cluster sequences (FRECLE).
- We propose an algorithm to discover FRECLE pattern from a collection of semantic tree sequences.

- We show with illustrative examples that FRECLE patterns are useful for several real life applications. Specifically, we elaborate on how FRECLE patterns can be used as the framework for generating *positive* and *negative* FRECLE *rules* that can be used for more efficient cache replacement strategy in the context of XML query processing.
- We present the results of extensive experiments with both synthetic and real datasets that we have conducted to demonstrate the efficiency and scalability of the proposed algorithms, quality of mining results, and usefulness of XML cache replacement strategy.

### C. Paper Organization

The rest of this paper is organized as follows. In Section II, we formally introduce the notion of semantic tree clusters and the problem of *frequent semantic cluster sequences* (FRECLE) mining. In Section III, we present our proposed algorithm for FRECLE mining. We highlight several representative applications of FRECLE patterns in Section IV and elaborate on a specific application (XML query cache replacement strategy) in Section V. Performances of the FRECLE mining algorithm and XML query caching are evaluated using in Section VI. Section VII reviews the related works. Finally, the last section concludes this paper.

## II. PRELIMINARIES

This section formally defines the problem of mining frequent semantic tree cluster sequences (FRECLE) from tree-structured data. We begin by introducing some basic concepts and formalism essential for defining the FRECLE mining problem.

### A. Semantic Trees

The hierarchical relationships of nodes in a tree often reflects semantic relationship. Hence, a tree structure may represent a set of objects such that the hierarchy of the tree is *semantically meaningful*. We say a tree is *semantically meaningful* if the semantic hierarchy of labels of nodes complies with the structural hierarchy of nodes. For example, a tree with a parent-child node pairs which are labeled as “wheel” and “car”, respectively, is not semantically correct as car cannot be part of wheel. However, it is meaningful if the pair of nodes are labeled as “car” and “wheel”, respectively. We refer to such trees as *semantic trees*.

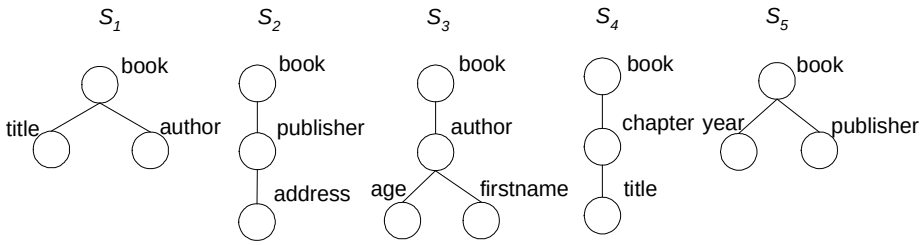



Fig. 3. Sequence of semantic trees.

*Definition 1:* **[Semantic Tree]** *A semantic tree is a node-labeled rooted tree, which is a* 3*-tuple* $S = \langle N, E, L \rangle$*, where* $N$ *is a set of nodes,* $E \subseteq N \times N$ *is the set of edges and* $L : N \rightarrow \Lambda$ *is a node labeling function which assign each node* $n \in N$ *a label* $L(n) \in \Lambda$*. For each edge* $(n_1, n_2) \in E$*,* $L(n_1)$ *semantically contains* $L(n_2)$*, i.e.,* $L(n_2)$ *is either an attribute or a subobject of* $L(n_1)$*.* □

That is, a node-labeled tree is a semantic tree if the label of each node semantically contains the label of any of its child nodes. For instance, a tree modeling an XML query is semantically meaningful and it reflects the semantics of the query. Figure $1(a)$ shows a set of XML query sequences, where each query is expressed in XPath language and modeled as a tree structure in Figure $1(b)$. Each sequence records a sequence of queries issued by some user. Observe that each query tree in Figure $1(b)$ is a semantic tree. Similarly, Figure 3 shows five semantic trees.

A *semantic tree database* (STD) can be defined as a set of semantic tree sequences. Formally,

*Definition 2:* **[Semantic Tree Database (STD)]** *Let* $\Sigma = \{S_1, S_2, \ldots, S_n\}$ *be a set of semantic trees. A semantic tree sequence* $Q$ *is an ordered list of semantic trees, denoted as* $Q = \langle S_1 S_2 \ldots S_l \rangle$*, where* $S_i \in \Sigma$ $(1 \leq i \leq l)$*.* $\mathcal{D}$ *is a semantic tree database on* $\Sigma$ *if* $\mathcal{D} = \{Q = \langle S_1 S_2 \ldots S_l \rangle | \forall S_i \in Q, S_i \in \Sigma\}$*.* □

For example, let $\Sigma = \{S_1, S_2, S_3, S_4, S_5\}$ be the set of semantic trees as shown in Figure 3. Then, Table I is a semantic tree database on $\Sigma$, where the first column $ID$ contains the identities of semantic tree sequences. Similarly, Figure $1(a)$ represents an STD where each semantic tree is an XML query. Note that semantic trees in each record in STD can be ordered temporally. Hence, each record may represent historical collection of semantic trees. For instance, queries $S_1$, $S_5$, and $S_3$ in record 1 in Figure $1(a)$ may represent sequence of queries formulated by a user at times $t_1$, $t_2$, and $t_3$, respectively where $t_1 < t_2 < t_3$.

| *ID* | *Semantic tree sequence* |
|---|---|
| $Q_1$ | $\langle S_1 S_4 S_5 \rangle$ |
| $Q_2$ | $\langle S_2 S_4 S_5 \rangle$ |
| $Q_3$ | $\langle S_2 S_4 \rangle$ |
| $Q_4$ | $\langle S_1 S_3 \rangle$ |
| $Q_5$ | $\langle S_1 S_4 S_2 \rangle$ |

TABLE I

A SEMANTIC TREE DATABASE.

### *B. Semantic Clusters*

Different semantic trees may have similar semantics. For example, semantic trees $S_1$ and $S_3$ in Figure 3 and queries $S_2$ and $S_4$ in Figure 1 are related to the information of book author. On the other hand, $S_2$ and $S_5$ in Figure 3 are about the information of book publisher. Then, given a collection of semantic trees, we can group them based on their semantics to generate a set of representative *semantic clusters*. After categorizing each tree to a semantic cluster, a semantic tree sequence can be transformed to be a *semantic cluster sequence*.

*Definition 3:* **[Semantic Cluster Sequence]** *Let $\Sigma = \{S_1, S_2, \ldots, S_n\}$ be a set of semantic trees. Suppose there exists a function $f : \Sigma \rightarrow \Upsilon$ that categorizes each tree in $\Sigma$ to a semantic cluster in $\Upsilon = \{C_1, C_2, \ldots, C_m\}$ (e.g., for any $i \in [1, n]$, there exists one and only $j \in [1, m]$ $s.t.$ $f(S_i) = C_j$ ). Then, given any semantic tree sequence $Q = \langle S_1 S_2 \ldots S_l \rangle$ where $S_i \in \Sigma$, it can be transformed into a semantic cluster sequence $C(Q) = \langle C_1 C_2 \ldots C_l \rangle$, where $C_1 = f(S_1), \ldots, C_l = f(S_l)$.* □

*Example 1:* Consider the set of semantic trees in Figure 3. Suppose that they can be categorized into three semantic clusters as follows: $f(S_1) = C_1$, $f(S_2) = C_2$, $f(S_3) = C_1$, $f(S_4) = C_3$, $f(S_5) = C_2$, where $C_1$ represents the information of book author, $C_2$ represents the information of book publisher and $C_3$ represents the information of book chapter. Thus, for the first semantic tree sequence $Q_1$ in Table I, it can be transformed to be the semantic cluster sequence $C(Q_1) = \langle f(S_1) f(S_4) f(S_5) \rangle = \langle C_1 C_3 C_2 \rangle$. ∎

### C. FRECLE Pattern

Given a set of semantic trees $\Sigma$ and a function $f : \Sigma \rightarrow \Upsilon$ that categorizes each tree in $\Sigma$ to a semantic clusters in $\Upsilon$, a semantic tree sequence $Q = \langle S_1 S_2 \dots S_l \rangle$ is said to *support* a semantic cluster sequence $C = \langle C_1 C_2 \dots C_k \rangle$, denoted as $Q \sqsubseteq C$, if there exists an integer $i (1 \leq i \leq l-k+1)$ such that $f(S_i) = C_1, f(S_{i+1}) = C_2, \dots, f(S_{i+k-1}) = C_k$. Then the *support* of a semantic cluster sequence in a STD can be defined as follows.

*Definition 4:* **[Support]** *Let $\mathcal{D}$ be a* STD *on $\Sigma$. Suppose there exists $f : \Sigma \rightarrow \Upsilon$. Then, given a semantic cluster sequence $C$ on $\Upsilon$, the support of $C$ in $\mathcal{D}$, denoted as $supp_{\mathcal{D}}(C)$, is*

$$supp_{\mathcal{D}}(C) = \frac{|\{Q | Q \sqsubseteq C, Q \in \mathcal{D}\}|}{|\{Q | Q \in \mathcal{D}\}|}$$

□

*Example 2:* Suppose the semantic trees in Figure 3 are clustered as in Example 1. Then, given a semantic cluster sequence, $C = \langle C_1 C_3 \rangle$, its support in the STD $\mathcal{D}$ in Table I is $2/5$, since two semantic tree sequences, $Q_1 = \langle S_1 S_4 S_5 \rangle$ and $Q_5 = \langle S_1 S_4 S_2 \rangle$, support it. ∎

*Definition 5:* **[Frequent Semantic Tree Cluster Sequence (FRECLE)]** *Given a* STD *$\mathcal{D}$ on $\Sigma$, $f : \Sigma \rightarrow \Upsilon$, and a real number $\xi$ $(0 \leq \xi \leq 1)$ as the threshold of support, a semantic cluster sequence $C = \langle C_1 C_2 \dots C_k \rangle$ $(C_i \in \Upsilon)$ is called a* FRECLE *pattern with respect to $\mathcal{D}$ if $supp_{\mathcal{D}}(C) \geq \xi$.* □

*Example 3:* Again, suppose the semantic trees in Figure 3 are clustered as in Example 1. Given the STD in Table I and the support threshold $\xi = 0.2$, the semantic cluster sequence $\langle C_1 C_3 \rangle$ is a FRECLE pattern as its support is $2/5 = 0.4 > \xi$. ∎

Observe that FRECLE pattern in Example 3 reflects the frequent association between the two kinds of semantics: book author $(C_1)$ and book chapter $(C_3)$. In Section V, we shall highlight the usefulness of such knowledge in the context of XML query caching strategy.

### D. Problem Statement

Let $\mathcal{D}$ be a STD on a set of semantic trees $\Sigma$. Given a support threshold $\xi$ $(0 \leq \xi \leq 1)$, the problem of FRECLE mining is first to find the function $f : \Sigma \rightarrow \Upsilon$ that categorizes trees in $\Sigma$ to semantic clusters in $\Upsilon = \{C_1, C_2, \dots, C_m\}$ such that the semantics of trees within a cluster are similar to one another and different from the semantics of trees in other clusters, and then to find the set of semantic cluster sequences $\{C = \langle C_1 C_2 \dots C_k \rangle | \forall C_i \in \Upsilon, supp_{\mathcal{D}}(C) \geq \xi\}$. Note

that different sets of FRECLE patterns will be found if the function $f : \Sigma \rightarrow \Upsilon$ is implemented differently. The *goodness* of different sets of FRECLE patterns depends on the applications (e.g., a set of patterns is more beneficial than the others in some particular application).

The problem of FRECLE mining is different from traditional frequent sequential pattern mining [1], [20] in the following two critical aspects. First, the data type we considered is different. Traditional frequent sequential pattern mining considers one-dimensional data, such as items, while FRECLE mining considers tree structured data. Second, the definition of support is different. In traditional frequent sequential pattern mining, a sequence supports another sequence if the latter is embedded in the former. While, in FRECLE mining, a semantic tree sequence supports a semantic cluster sequence only if the latter is embedded in a semantic cluster sequence that is transformed from the former. Consequently, existing frequent sequential pattern mining algorithms cannot be applied here directly to discover FRECLEs. We propose a new data mining algorithm for FRECLE mining in the following section.

## III. FRECLE MINING ALGORITHM

In this section, we first present the overview of FRECLE mining algorithm. Then, we discuss the details of respective mining phases.

### A. Overview

Given a STD $\mathcal{D}$ and some user-specified support threshold $\xi$, the discovery of FRECLE patterns consists of the following two phases.

- **Phase I: Semantic Tree Clustering**. In order to transform semantic tree sequences to calculate the support of semantic cluster sequences, we need to cluster the set of semantic trees in $\mathcal{D}$ first. Hence, a clustering algorithm which clusters tree structures based on semantics is needed in this phase.
- **Phase II: FRECLE Pattern Discovery**. After getting the results of Phase I, each semantic tree sequence can be transformed to a semantic cluster sequence. Then, the second phase mines FRECLEs from the transformed database with respect to the support threshold $\xi$.

### B. Phase 1: Semantic Tree Clustering

Given a STD $\mathcal{D}$ on a set of semantic trees $\Sigma$, the objective of this phase is to implement the function $f : \Sigma \rightarrow \Upsilon$ which categorizes each semantic tree in $\Sigma$ to a semantic cluster in $\Upsilon$.

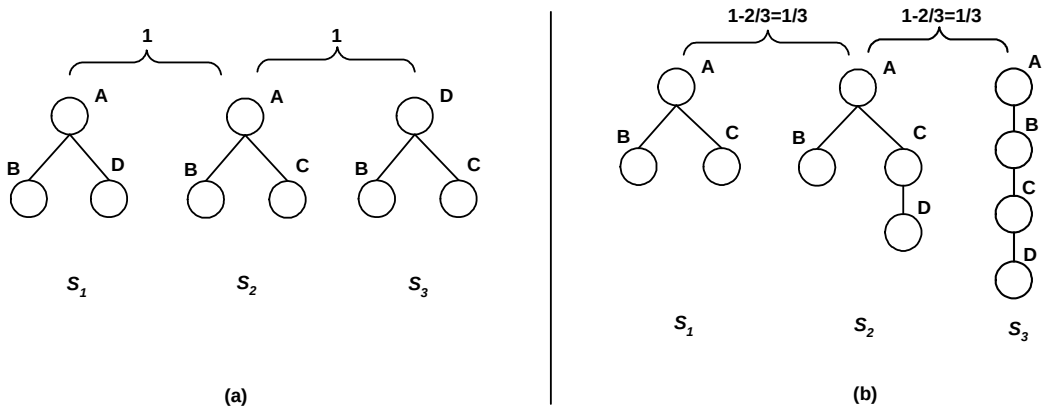



Fig. 4. Node-based and edge-based similarity measures.

The function $f$ can be implemented by clustering semantic trees in $\Sigma$ based on semantics. In other words, given a collection of semantic trees $\{S_1, S_2, \ldots, S_n\}$, we aim to find a clustering function $f$ that forms a partition $\{C_1, C_2, \ldots, C_m\}$ of $\{S_1, S_2, \ldots, S_n\}$ such that trees in each $C_i$ are semantically as close as possible.

*1) Similarity Measure for Clustering:* A key task for clustering semantic trees is to define an appropriate similarity measure. In recent years, there have been several tree structure clustering methods [24], [10] proposed in the literature. Hence, at first glance it may seem that we can adopt a similarity metric proposed in the literature for clustering semantic trees. However, our initial investigation revealed that such strategy may not work for certain cases of semantic tree clustering. Let us elaborate on this further.

Similarity measures defined on tree structures can broadly be classified into two basic categories. *Node-based* similarity measures the proximity of two trees based on shared nodes. For example, the work in [10] proposed to measure the similarity between two trees using *tree distance*, which is a sequence of node insertion, node deletion and node relabeling operations etc. Then lesser the number of operations in the tree distance, the closer the two trees are. However, as observed by [24], tree distance based similarity may not be able to distinguish trees of different semantics. For example, as shown in Figure $4(a)$, the tree distance between $S_1$ and $S_2$ is one (a node relabeling), while the tree distance between $S_2$ and $S_3$ is also one. Although $S_1$ and $S_2$ should be semantically closer as they represent the information of the same object $A$, the tree distance-based similarity fail to differentiate between such semantic structures.

*Edge-based* similarity measures the proximity of two trees based on shared edges. For example,

the work in [24] proposed a modified *Jaccard coefficient* defined on edges, $dist(S_1, S_2) = 1 - \frac{|sg(S_1) \cap sg(S_2)|}{max\{|sg(S_1)|, |sg(S_1)|\}}$, where $sg(S_i)$ is the set of edges in tree $S_i$. Thus, the more edges the two trees share, the closer the two trees are. Edge-based similarity measure is more accurate than node-based measure as it considers not only the nodes but also the edges connecting the nodes. For example, based on the modified Jaccard coefficient, the distance between $S_1$ and $S_2$ in Figure $4(a)$ is $0.5$, while the distance between $S_2$ and $S_3$ is $1$. However, we observed that edge-based measure also fails to distinguish trees of different semantics in some cases. For example, as shown in Figure $4(b)$, the distance between $S_1$ and $S_2$ is the same as the distance between $S_2$ and $S_3$ based on modified Jaccard coefficient. But $S_1$ and $S_2$ should be semantically closer as they represent the information of an object $A$ which has two sub-objects $B$ and $C$.

Based on the above discussion, it is obvious that node-based and edge-based similarities cannot achieve good accuracy in clustering semantic trees. Hence, we propose a *rooted subtree-based* similarity measure to cluster semantic trees. Formally, a *rooted subtree* of a semantic tree is defined as follows.

*Definition 6:* **[Rooted Subtree]** *Given a semantic tree $S = \langle N, E, L \rangle$, let $Root(S)$ be the root of $S$, a rooted subtree $RS = \langle N', E', L' \rangle$ is a subtree of $S$ if it satisfies the following conditions:* 1) $Root(RS) = Root(S)$; *2)* $N' \subseteq N$, $E' \subseteq E$. □

For example, in the Figure 4 $(b)$, the tree $S_1$ is a rooted subtree of $S_2$, while $S_3$ is not a rooted subtree of $S_2$.

Rooted-subtree-based similarity scheme has certain advantages over the node-based and edge-based ones. For example, consider the trees in Figure $4(a)$ again. $S_1$ and $S_2$ share the common rooted subtree $RS$ with a root node $A$ and a leaf node $B$, while $S_3$ does not share any rooted subtree with $S_1$ or $S_2$. Hence, $S_1$ and $S_2$ should be more similar. Consider the trees in Figure $4(b)$ again. $S_1$ and $S_2$ share the rooted subtree $RS$ with the root $A$ and two child nodes $B$ and $C$. $S_3$ share the rooted subtree $RS$ with the root $A$ and a leaf node $B$. Since the size of the rooted subtree shared by $S_1$ and $S_2$ is larger, $S_1$ and $S_2$ should be closer.

*2) Clustering Method:* We now discuss the clustering method. Existing tree structure clustering approaches usually employ the agglomerative clustering technique [24], [10]. They compute the similarity between all pairs of clusters and then merge the most similar pair. Recently, *cluster-centered* approach was proposed in the area of document clustering [26], [15]. A *clustered-centered* method constructs clusters by measuring the *cohesiveness* of clusters directly using

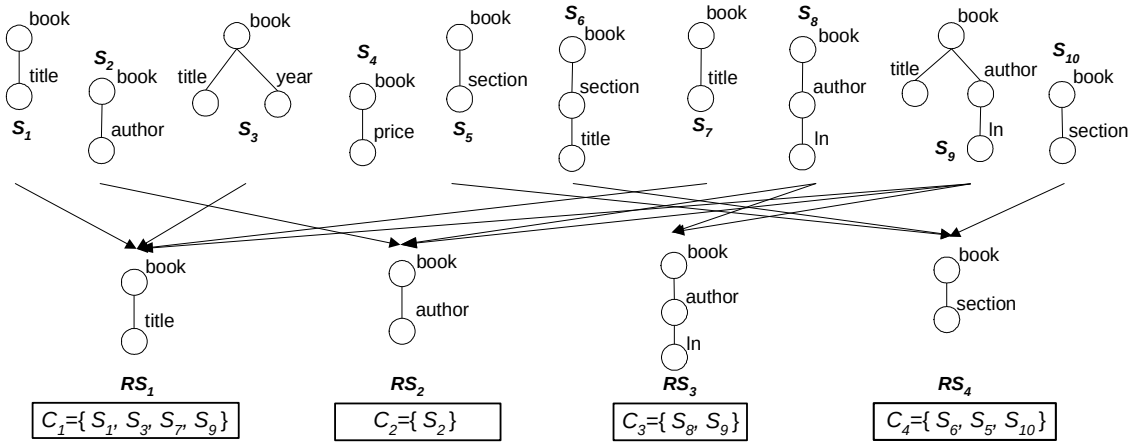



Fig. 5. Initial clusters.

frequent patterns. For example, FIHC [15] mined frequent terms of a collection of documents first and then clustered the documents according to the frequent terms they contain. The motivation is that there are some frequent terms for each cluster (topic) in the document set, and different clusters share few frequent terms. We employ such a clustering strategy here to cluster semantic trees as it is revealed in [15] that the cluster-centered method can distinguish documents of different semantics better and achieve higher clustering accuracy.

In order to cluster semantic trees based on cluster-centered strategy, we need to discover frequent rooted subtrees from a collection of semantic trees first. We use the algorithm FastXMiner [29] to discover *frequent rooted subtrees* from semantic trees. A *frequent rooted subtrees* can be defined as follows.

*Definition 7:* **[Frequent Rooted Subtree]** *Given a set of semantic trees* $\Sigma = \{S_1, S_2, \ldots, S_n\}$*, and a real number* $\delta$ $(0 \leq \delta \leq 1)$*, which is called minimum rooted subtree support, the support of a rooted subtree* $RS$*, denoted as* $supp(RS)$*, is the fraction of semantic trees that include it.* $RS$ *is a frequent rooted subtree if* $supp(RS) \geq \delta$*.* □

*Example 4:* In order to illustrate our clustering method clearly, we use another set of semantic trees, in the upper part of Figure 5, as the running example. If the threshold $\delta$ is $0.2$, then four frequent rooted subtrees, $RS_1$, $RS_2$, $RS_3$ and $RS_4$, will be discovered as depicted in the lower part of Figure 5. Each arrowed line from a semantic tree to a frequent rooted subtree indicates that the semantic tree supports the rooted subtree. ∎

After discovering frequent rooted subtrees from a collection of semantic trees, the clustering

method constructs clusters in the following three steps: *initializing clusters*, *disjointing clusters*, and *pruning clusters*. We elaborate on these steps in turn.

**Initializing Clusters**. In this step, we construct initial clusters for each mined frequent rooted subtree. We use the frequent rooted subtrees as the labels of the initial clusters. A semantic tree is assigned to an initial cluster if the label of the cluster is the *maximal* frequent rooted subtree supported by the semantic tree [1]. For example, consider the semantic tree $S_8$ in Figure 5. It is supported by two frequent rooted subtrees $RS_2$ and $RS_3$. We assign $S_8$ to the initial cluster $RS_3$ since the label of $RS_2$ is not a maximal frequent rooted subtree supported by $S_8$. Note that if a semantic tree does not support any frequent rooted subtree, e.g., the semantic tree $S_4$ in Figure 5, then it will be treated as an outlier because the semantics of the tree is not close to any cluster. Given the set of semantic trees and discovered frequent rooted subtrees in Figure 5, the results of the initializing step are shown in the bottom of Figure 5, where four initial clusters, $C_1$, $C_2$, $C_3$ and $C_4$, are created.

Initial clusters may not be disjoint because a semantic tree may support more than one maximal frequent rooted subtrees. For example, the semantic tree $S_9$ supports two maximal frequent rooted subtrees, $RS_1$ and $RS_3$. Thus, $S_9$ is assigned to two corresponding initial clusters $C_1$ and $C_3$. We discuss how to make the initial clusters disjoint in the next step.

**Disjointing Clusters**. For each semantic tree, we identify the best initial cluster and keep the tree only in the best cluster. We define the *goodness* of a cluster for a semantic tree based on the intra-cluster dissimilarity. That is, we remove a semantic tree from all the clusters but the one to which adding the tree results in the minimal intra-cluster dissimilarity. We measure the intra-cluster dissimilarity based on the number of infrequent edges in the cluster [2]. That is, we merge all semantic trees in a cluster into a tree structure. For the merged tree, each edge $e$ is associated with a *cluster support*, denoted as $supp_C(e)$, which is the fraction of semantic trees containing it. For example, Figure $6(a)$ shows the merged tree structure for the initial cluster $C_1$ in Figure 5. The numbers on the edges are their absolute cluster support values. Given a

[1]A frequent rooted subtree is not maximal w.r.t. a semantic tree if it is included by another frequent rooted subtree supported by the semantic tree.

[2]We highlight that our fall-back on measuring dissimilarity based on edges in this step will not damage the clustering quality significantly since the initializing step based on rooted subtrees basically decides the assignment of semantic trees to clusters.

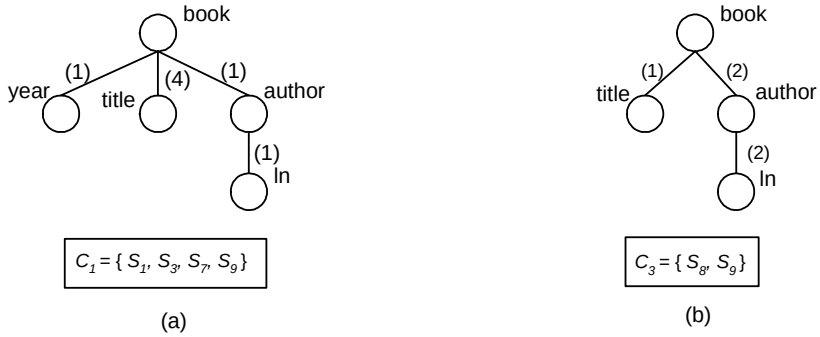



Fig. 6. Intra-cluster dissimilarity.

*minimum edge support* $\chi$ $(0 \leq \chi \leq 1)$, an edge in a merged tree is infrequent if its *cluster support* is less than $\chi$. Then, we define the intra-cluster dissimilarity as follows.

*Definition 8:* **[Intra-Cluster Dissimilarity]** *Given a merged tree $M_i$ of a cluster $C_i$ and a minimum edge support $\chi$, the intra-cluster dissimilarity of $C_i$, denoted as $Intra(C_i)$, is*

$$Intra(C_i) = \frac{|\{e \in E(M_i) | supp_{C_i}(e) < \chi\}|}{|\{e \in E(M_i)\}|}$$

*where $E(M_i)$ is the set of edges of $M_i$.* □

The value of $Intra(C_i)$ ranges from 0 to 1. The higher the $Intra(C_i)$, the more dissimilar the semantic trees in cluster $C_i$.

*Example 5:* Figure 6 shows the merged trees for initial clusters $C_1$ and $C_3$ in Figure 5. Let $\chi = 0.6$. In the merged tree of $C_1$, there are three infrequent edges: $(book,\ year)$, $(book, author)$ and $(author, ln)$. Hence, $Intra(C_1) = 3/4 = 0.75$. Similarly, the edge $(book,\ title)$ is an infrequent edge in the merged tree of $C_3$. Hence, $Intra(C_3) = 1/3 \approx 0.33$. ∎

We then make the clusters disjoint by assigning a semantic tree to a cluster which has the smallest intra-cluster dissimilarity value. That is, a semantic tree $S_i$ is kept in cluster $C_j$ if

$$C_j = argmin_{C_j \in C, S_i \in C_j} Intra(C_j)$$

*Example 6:* Given the set of semantic trees and initial clusters in Figure 5, $S_9$ is assigned to both $C_1$ and $C_3$. As shown in Example 4, grouping $S_9$ in $C_1$ generates the $Intra(C_1) = 0.75$, whereas grouping $S_9$ in $C_3$ results in the $Intra(C_3) = 0.33$. Hence, we remove $S_9$ from $C_1$. ∎

After this step, clusters are not overlapping any longer. The initial clusters in Figure 5 are adjusted as shown in Figure 7, where each cluster is represented as a merged tree of all semantic trees in it.

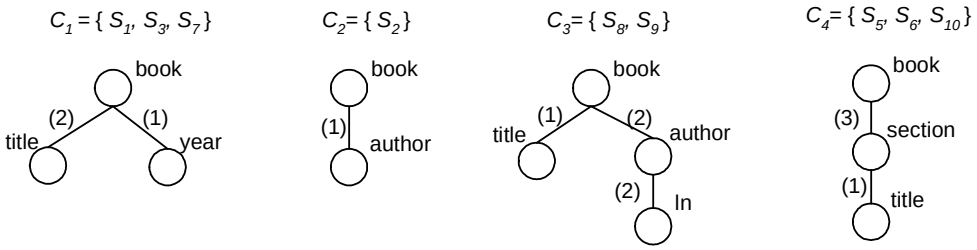



Fig. 7. Clusters after disjointing.

**Pruning Clusters**. If the *minimum rooted subtree support* $\delta$ is small, many frequent rooted subtrees will be mined from the collection of semantic trees. Then, some of the rooted subtrees are semantically close, e.g., $RS_2$ and $RS_3$ in Figure 5, which results in semantically close clusters, e.g., $C_2$ and $C_3$ in Figure 7. Hence, in this step, we perform cluster pruning to merge close clusters.

We measure the similarity of a cluster to another cluster based on the number of frequent edges shared by the two clusters. Given a *minimum edge support* $\chi$, the set of frequent edges of cluster $C_i$, denoted as $F_{C_i}$, are the edges in the merged tree of $C_i$ with their *cluster support* no less than $\chi$. That is, $F_{C_i} = \{e | e \in E(M_i)\ \&\ supp_{C_i}(e) \geq \chi\}$.

*Example 7:* Consider the cluster $C_3$ in Figure 7. Let $\chi = 0.6$. Then the two edges, $(book,\ author)$ and $(author,\ ln)$, are frequent because each edge has the cluster support $2/2 = 1 \geq \chi$. Hence, $F_{C_3}$ =$\{(book,\ author),\ (author,\ ln)\}$. ∎

Then, the similarity of one cluster to another cluster can be defined as follows.

*Definition 9:* **[Cluster Similarity]** *Given two clusters $C_i$ and $C_j$, the cluster similarity of $C_i$ to $C_j$, denoted as $Sim(C_i \rightarrow C_j)$, is,*

$$Sim(C_i \rightarrow C_j) = \frac{|\{e | e \in F_{C_i}, e \in F_{C_j}\}| - |\{e | e \in F_{C_i}, e \notin F_{C_j}\}|}{|\{e | e \in F_{C_i}\}|} + 1$$

*where $F_{C_i}$ is the set of frequent edges of $C_i$.* □

That is, the more frequent edges of $C_i$ are frequent in $C_j$, the closer $C_i$ is to $C_j$. The value of the first term, $\frac{|\{e|e\in F_{C_i}, e\in F_{C_j}\}| - |\{e|e\in F_{C_i}, e\notin F_{C_j}\}|}{|\{e|e\in F_{C_i}\}|}$, ranges from $-1$ to 1. If all frequent edges in cluster $C_i$ are frequent as well in cluster $C_j$, then the value of $Sim(C_i \rightarrow C_j)$ is 1. If none of the frequent edge in cluster $C_i$ is frequent in cluster $C_j$, then the value of $Sim(C_i \rightarrow C_j)$ is $-1$. To avoid negative similarity values, we add the term $+1$. As a result, the range of $Sim(C_i \rightarrow C_j)$ is $[0, 2]$. If $Sim(C_i \rightarrow C_j)$ is greater than 1, cluster $C_i$ is semantically close to cluster $C_j$. Note

that, the cluster similarity is asymmetrical.

The inter-cluster similarity between $C_i$ and $C_j$ is defined as the geometric mean of the two similarities: $Sim(C_i \rightarrow C_j)$ and $Sim(C_j \rightarrow C_i)$. Formally,

*Definition 10:* **[Inter-Cluster Similarity]** *Given two clusters $C_i$ and $C_j$, the inter-cluster similarity between $C_i$ and $C_j$, denoted as $Inter(C_i, C_j)$, is,*

$$Inter(C_i, C_j) = \sqrt{Sim(C_i \rightarrow C_j) \times Sim(C_j \rightarrow C_i)}$$

□

According to [15], the advantage of the geometric mean is that the inter-cluster similarity will be high only if both values of $Sim(C_i \rightarrow C_j)$ and $Sim(C_j \rightarrow C_i)$ are high. Since the range of cluster similarity is $[0, 2]$, the range of $Inter(C_i, C_j)$ is also $[0, 2]$. Obviously, the inter-cluster similarity has the symmetry property.

Higher value of inter-cluster similarity implies higher similarity between two clusters. An $Inter(C_i, C_j)$ value below 1 implies that the weight of dissimilar item (e.g., $Sim(C_i \rightarrow C_j)$ or $Sim(C_j \rightarrow C_i)$) has exceeded the weight of similar item. Hence, we merge two clusters $C_i$ and $C_j$ if $Inter(C_i, C_j)$ is greater than 1.

*Example 8:* Consider the cluster $C_2$ and $C_3$ in Figure 7 again. Let $\chi = 0.6$, $F_{C_2}$ ={$(book,\ author)$} and $F_{C_3}$ ={$(book,\ author)$, $(author,\ ln)$}. Thus, $Sim(C_2 \rightarrow C_3)$ = $(1-0)/1+1 = 2$ because the frequent edge $(book,\ author)$ in $C_2$ is frequent as well in $C_3$. Whereas, $Sim(C_3 \rightarrow C_2) = (2-1)/2+1 = 1.5$ because the frequent edge $(author, ln)$ in $C_3$ is infrequent in $C_2$. Then, $Inter(C_2, C_3) = \sqrt{2 \times 1.5} \approx 1.73$ and we merge the two clusters $C_2$ and $C_3$. ∎

The final clustering result is shown in Figure 8. That is, given the set of ten semantic trees as shown in the upper part of Figure 5, we implement the following function $f$ to categorize each semantic tree to a collectively represented semantic cluster.

$$f(S_i) = \begin{cases} C_1, & if\ i = 1, 3, 7 \\ C_2, & if\ i = 2, 8, 9 \\ C_3, & if\ i = 5, 6, 10 \\ Null, & if\ i = 4 \end{cases}$$

### C. Phase 2: Frequent Cluster Sequence Discovery

After implementing the function $f : \Sigma \rightarrow \Upsilon$, the STD $\mathcal{D}$ can be transformed to be a database of semantic cluster sequences. Then, the objective of the second phase is to discover FRECLE

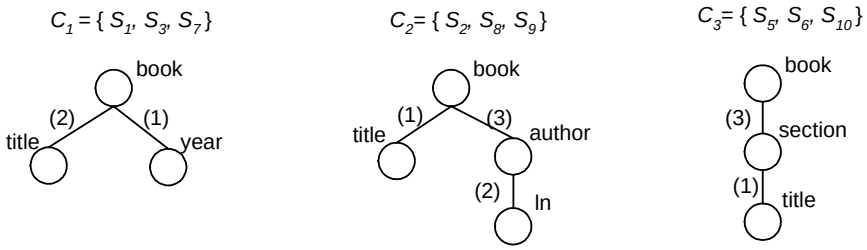



Fig. 8. Semantic clusters.

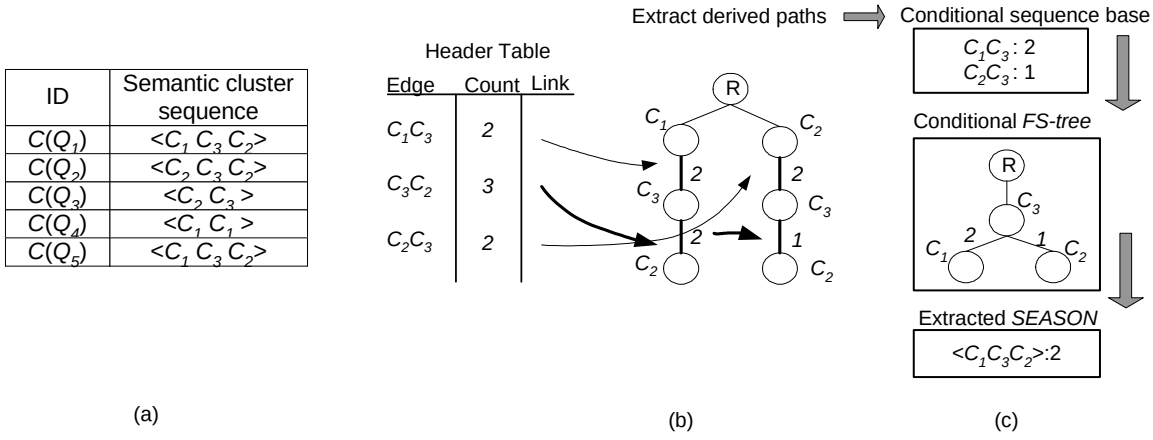



Fig. 9. Mining FRECLEs from semantic cluster database.

patterns from the transformed database with respect to some given support threshold $\xi$. Since FRECLE patterns are actually frequent semantic cluster sequences, the objective of this phase is similar to the problem of traditional frequent sequential pattern mining. Although many data mining approaches have been proposed in the literature for frequent sequential pattern mining [1], [20], we adopt the *FS-Miner* [13] algorithm because of the following reason. Most of existing frequent sequential pattern mining algorithms discover frequent *embedded* subsequences. For example, $\langle ace \rangle$ will be regarded as a subsequence of $\langle abcde \rangle$. While, both *FS-Miner* and our FRECLE mining are interested in frequent *induced* subsequences only, such as $\langle abc \rangle$ of $\langle abcde \rangle$.

We illustrate how *FS-Miner* discovers FRECLEs from the transformed semantic cluster database with an example. Readers can refer to [13] for the details of *FS-Miner*. Given the STD in Table I and the clustering in Example 1, Figure $9(a)$ shows the transformed semantic cluster database.

Given a semantic cluster database and a support threshold $\xi$, *FS-Miner* discovers FRECLEs from a constructed data structure called *FS-tree* as depicted in Figure $9(b)$. Similar to *FP-tree* [16], *FS-tree* contains a header table and a tree structure. Each entry in the header table has

three fields: *edge* (a frequent sequence of length 2), *count* (count of the edge), and *link* (pointing to the first occurrence of the edge in the tree structure). For example, suppose the threshold $\xi$ is $0.4$. Then edges are frequent if their counts are no less than 2. After scanning the table in Figure 9 for the first time, three frequent edges are discovered and created in the header table. Then, a second scan of the database is performed to construct the tree structure. For example, we construct a root path $\langle C_1C_3C_2\rangle$ in the tree structure for the first semantic cluster sequence. Similarly, another root path is created for the second semantic cluster sequence. For the third sequence, since it can share the edge $\langle C_2C_3\rangle$ with the second root path, we simply increment the count of the edge in the tree. Other sequences are inserted into the tree using the above strategy.

For each edge in the header table, *FS-Miner* extracts derived paths. For example, for the edge $\langle C_3C_2\rangle$, two derived paths, highlighted with bold lines in Figure 9, are extracted. Then, the conditional sequence base of $\langle C_3C_2\rangle$ can be obtained by setting the frequency count of each edge in the paths to the count of the removed $\langle C_3C_2\rangle$, as shown in Figure $9(c)$. We create the conditional *FS-tree* of $\langle C_3C_2\rangle$ by inserting each path of conditional sequence base into the tree in a backward manner. Finally, a depth first traversal is performed to discover all sequences satisfying the threshold $\xi$.

Lastly, we would like to highlight here that the novelty of the FRECLE mining algorithm lies in the first phase, where we developed a *cluster-centered* strategy to construct clusters from a set of trees. To the best of our knowledge, this is the first method that clusters tree structures by measuring the semantic cohesiveness of clusters directly using frequent substructures.

## IV. Application Scenarios

In this section, we first show the usefulness of FRECLE patterns with some potential applications. Since FRECLE patterns discover knowledge about semantic associations in tree-structured data, they can be useful in applications where tree structures are dynamic and semantically meaningful. We enumerate some of these applications. In the next section, we shall elaborate on one of these applications.

- **XML query cache replacement**. Efficient processing of XML queries is an important issue in the XML research community. Recently, caching XML queries has been recognized as an orthogonal approach to improve the performance of XML query engines. For example,

FastXMiner [29] proposed to mine frequent XML query patterns from the user queries and discard infrequent query patterns first once the cache is full. However, these frequent query pattern mining techniques are primarily designed for static collection of XML queries. Hence, they may not always be reliable in predicting the subsequent queries as these techniques ignore the evolutionary feature of users' queries. Instead, we can mine FRECLE patterns from XML queries to predict the subsequent information needs of users based on their current queries. The details are discussed in the next section.

- **Prefetching XML data**. Besides caching, prefetching can be used for XML query performance improvement. For example, Ng et al. [19] proposed to mine association rules from XML query sequences and prefetch answers to queries which are predicated to be issued subsequently. Considering that users may not issue exactly same queries in sequence, they proposed the idea of *abstract rules*, which are association rules between semantically similar queries. Queries are semantically similar if they are only different in constraint values (predicate). However, this form of abstract rules is not flexible enough as many sequential associations between semantically similar queries may not be discovered. For example, two XPath queries, $//book[title = \text{“XML”}]/publisher/$ and $//book[title = \text{“WWW”}]/pubisher$, will be regarded as similar queries as they contain different predicates on *title* element only (e.g., XML and WWW). However, the queries $//book[title = \text{“XML”}]/publisher/$ and $//book[title = \text{“WWW”}]/\ pubisher/address$ will not be treated as similar queries although both inquire the information of book publisher. Since FRECLE mining cluster similar queries before mining sequential patterns, FRECLE-based prefetching system should be more effective in improving the hit ratio and precision of XML cache.
- **Web users clustering**. A *Web session tree*, which is constructed by organizing Web pages based on their URLs, is semantically meaningful as it represents the information needs of a user. We can mine FRECLEs from historical Web usage data for each user and cluster users based on discovered FRECLEs. Such a clustering method can identify clusters of users that exhibit similar sequential information needs. Then, Web requests can be served more efficiently by designing some user-cluster aware caching/prefetching strategies.

# V. Replacement Strategy for XML Query Cache

In this section, we elaborate on how FRECLE patterns can be used in designing optimal XML cache replacement strategies. First, we derive association rules called FRECLE *rules* from FRECLEs mined from sequential XML queries. Then, we discuss the design of cache replacement strategies based on the derived rules. We begin by giving an overview of the replacement strategy. A preliminary version of this section appeared in [5].

## A. Overview

Recently, several approaches that mine frequent XML query patterns and cache their results have been proposed to improve query response time [28], [29]. However, frequent XML query patterns mined by these approaches ignore the temporal and semantic association of queries. As a result caching strategies based on only frequency and recency have certain limitations in predicting future queries. In this section, we show how FRECLE patterns can be used to exploit temporal and semantic features of user queries by discovering association rules that are used as a foundation for optimal cache replacement strategies. The association rules indicate that when a user inquires some information from the XML document, he/she probably will (positive) or will not (negative) inquire some other information subsequently. Intuitively, as few users issue *exactly* the same queries sequentially and many users may inquire similar information, we cluster queries based on their semantics first using the FRECLE mining framework and then discover the *positive* and *negative associations* between them. The knowledge obtained from the discovered rules are incorporated in designing appropriate replacement strategies.

## B. FRECLE *Rules*

Suppose we have a database of XML query sequences where each sequence represents a set of XML queries sequentially issued by some user at different timepoints. Given such a database, a support threshold $\xi$, a minimum rooted subtree support $\delta$, and a minimum edge support $\chi$, we can mine FRECLEs of any length. That is, FRECLEs with any number of semantic clusters. Particularly, in the application of XML query caching, we consider FRECLEs of length 2 only (e.g., $\langle C_1 C_2 \rangle$), because this type of short patterns significantly reduce the complexity of the mining process and any FRECLE with more than 2 clusters can be decomposed into a set of FRECLEs of length 2.

Given a FRECLE $\langle C_1 C_2 \rangle$ mined from an XML query sequence database, we may derive an association rule of the form $C_1 \Rightarrow C_2$. The rule indicates that when a user issues a query that is semantically contained by or close to the semantics of cluster $C_1$, he/she will probably issue a subsequent query that is semantically related to the semantics of cluster $C_2$. Based on such kind of knowledge, we may optimally delay the eviction of answers to the predicated queries from the cache. We also observed the following type of knowledge is potentially useful as well. When a user issues a query semantically related to the cluster $C_1$, he/she will probably *not* issue a query semantically related to cluster $C_2$ subsequently. Thus, we can optimally hasten the purge of answers to queries related to the semantics of $C_2$. Thus, besides $\langle C_1 C_2 \rangle$, FRECLEs including negative clusters, such as $\langle C_1 \neg C_2 \rangle$, are beneficial (the symbol $\neg$ is used to represent the nonoccurrence of a semantic cluster). We call the former *positive* FRECLE and the latter *negative* FRECLE, respectively.

To determine the positive or negative association between variables, some correlation measures should be used. As we are equally interested whether a user inquires some information or not, the occurrence of a cluster $C_i$ is a symmetric binary value. Hence, correlation measures which are suitable for analyzing symmetric binary variables can be used, such as $\phi$-coefficient, odds ratio, and the Kappa statistic etc. [21]. In our analysis, we use the $\phi$-coefficient. Then, if we use $supp(\langle C_i, * \rangle)$ to denote the support of any 2-sequence starting with $C_i$ and $supp(\langle *, C_i \rangle)$ to denote the support of any 2-sequence ending with $C_i$, then the correlation of a FRECLE of length 2 can be defined as follows.

*Definition 11:* **[Correlation of FRECLE]** *Given a* FRECLE $\langle C_i C_j \rangle$*, the correlation of the pattern, denoted as* $corr(\langle C_i C_j \rangle)$*, is*

$$corr(\langle C_i C_j \rangle) = \frac{supp(\langle C_i C_j \rangle) - supp(\langle C_i, * \rangle) supp(\langle *, C_j \rangle)}{\sqrt{supp(\langle C_i, * \rangle)(1 - supp(\langle C_i, * \rangle)) supp(\langle *, C_j \rangle)(1 - supp(\langle *, C_j \rangle))}}$$

□

The strength and the meaning of the correlation of FRECLE are same as those of the $\phi$-coefficient. Hence, if $corr(\langle C_i C_j \rangle) > 0$ then it is a *positive* FRECLE pattern. Otherwise, it is a *negative* FRECLE pattern. The confidence of association rules derived from a *positive/negative* FRECLE can be defined accordingly.

*Definition 12:* **[Confidence of FRECLE rule]** *Given a* FRECLE $\langle C_i C_j^* \rangle$*, the derived association*

**Algorithm 1** Positive and Negative FRECLE Rule Generation

**Input:**

$\mathcal{D}, f : \Sigma \rightarrow \Upsilon, \xi, \mu, \theta$

**Output:**

$PR$: A set of positive FRECLE rules, $NR$: A set of negative FRECLE rules

**Description:**

```
1:  scan D and find the frequent 1-sequence (F1) /*support(⟨Ci,⟩) or support(⟨,Ci⟩)≥ ξ*/
2:  P2 = F1⋈F1 /*candidate frequent 2-cluster sequence*/
3:  for each < ⟨Ci,Cj⟩ ∈ P2 do
4:    if corr(< ⟨Ci,Cj⟩) ≥ μ then
5:      if (supp(< ⟨CiCj⟩) ≥ ξ)&& (conf(Ci ⇒ Cj) ≥ θ) then
6:        PR = PR ∪ {Ci ⇒ Cj}
7:      end if
8:    else
9:      if (corr(< ⟨Ci,Cj⟩) ≤ −μ)&&(conf(Ci ⇒ ¬Cj) ≥ θ) then
10:       NR = NR ∪ {Ci ⇒ ¬Cj}
11:     end if
12:   end if
13: end for
```

*rule has the form of* $C_i \Rightarrow C_j^*$. *The confidence of rule, denoted as* $conf(C_i \Rightarrow C_j^*)$, *is,*

$$conf(C_i \Rightarrow C_j^*) = \frac{support(\langle C_i C_j^* \rangle)}{support(\langle C_i, \rangle)}$$

*where* $C_j^*$ *represents either* $C_j$ *or* $\neg C_j$. □

Then, positive and negative FRECLE rules, which can be mined from XML query sequences, can be defined based on these metrics.

*Definition 13:* **[Positive and negative FRECLE rules]** *Let* $\mathcal{D}$ *be an* XML *query tree sequence database on* $\Sigma = \{S_1, S_2, \ldots, S_n\}$, *where each tree in* $\Sigma$ *represents an* XML *query. Let* $f : \Sigma \rightarrow \Upsilon$ *be the function that categorizes each tree in* $\Sigma$ *to a semantic cluster in* $\Upsilon = \{C_1, C_2, \ldots, C_m\}$. *Given the threshold of support* $\xi$, *the threshold of correlation* $\mu$, *and the threshold of confidence* $\theta$,

- $C_i \Rightarrow C_j$ *is a positive* FRECLE *rule if* 1) $corr(\langle C_i C_j \rangle) \geq \mu$; 2) $supp(\langle C_i C_j \rangle) \geq \xi$; 3) $conf(\langle C_i \Rightarrow C_j \rangle) \geq \theta$.
- $C_i \Rightarrow \neg C_j$ *is a negative* FRECLE *rule if* 1) $corr(\langle C_i C_j \rangle) \leq -\mu$; 2) $conf(\langle C_i \Rightarrow \neg C_j \rangle) \geq \theta$.

□

### C. FRECLE *Rules Mining Algorithm*

Given an XML query sequence database, the function $f : \Sigma \rightarrow \Upsilon$ can be implemented with the clustering algorithm discussed in the previous section. Then, the database can be transformed to be an evolutionary semantic cluster database. Rather than discovering FRECLEs first and then derive possible positive and negative rules as commonly done by traditional association rule mining algorithm, we can discover positive and negative FRECLE rules directly from the transformed evolutionary semantic cluster database.

The algorithm is presented in Algorithm 2. Initially, we scan the transformed database to find the set of frequent 1-cluster sequences (Line 1). Then we join the frequent 1-cluster sequences to generate candidate FRECLEs (Line 2). Note that only a pair of 1-cluster sequences in the form of $\langle C_i, \rangle$ and $\langle , C_j \rangle$ can be joined. After that, we verify the correlation of candidate FRECLEs. If the correlation is greater than the threshold $\mu$ (Line 4), we consider to derive a positive FRECLE rule and check the *support* and *confidence* of the possible rule (Line 5). If both support and confidence of the derived rule are greater than the thresholds $\xi$ and $\theta$, it is added into the set of positive FRECLE rules $PR$ (Line 6). Otherwise, if the correlation of a candidate FRECLE is not greater than the threshold $-\mu$, we check whether valid negative FRECLE rules can be derived (Line $9 - 11$) .

### D. Designing Replacement Strategies

After deriving positive and negative FRECLE rules, we design replacement strategies with discovered rules [3]. As we cluster XML query (trees) into semantic clusters, our replacement scheme has two levels. The upper level applies replacement functions on clusters while the lower level decides the replacement value for queries in each cluster.

We incorporate the discovered association rules between clusters with LRU, which is a classic cache replacement algorithm that discards the least recently used items first, in the upper level. Without loss of generality, we assume that "*the most recent value for clusters*", $V_{top}$, is incremented by one, each time a new query $S_x$ is issued. When a new query $S_x$ is issued,

[3]Although FRECLEs are defined on semantic trees with specific node labels, our strategy can be extended straightforwardly to handle XML queries with wildcards " $*$ " and relative paths "//", given the partial ordering as defined in [29]. That is, a specific node label is semantically contained by " $*$ ", which is semantically contained by "//".

**Algorithm 2** FRECLE Rule-based Cache Replacement Strategy

**Input:**

$PR$ - the set of *positive* FRECLE rules, $NR$ - the set of *negative* FRECLE rules, $S_x$ - the new XML query

**Description:**

```
let V_top be the most recent values for clusters
if Sim(S_x → C_i) > 1 and C_i ⇒ C_j ∈ PR then
    select the C_i ⇒ C_j with the highest Sim(S_x → C_i)
    if C_j exists in cache then
        V(C_j) = V(C_j) + (V_top − V(C_j) × Conf(C_i ⇒ C_j))
    end if
end if
if Sim(S_x → C_i) > 1 and C_i ⇒ ¬C_j ∈ NR then
    select the C_i ⇒ ¬C_j with the highest Sim(S_x → C_i)
    if C_j exists in cache then
        V(C_j) = V(C_j) + (V(C_j − V_top) × Conf(C_i ⇒ ¬C_j))
    end if
end if
for there is insufficient space for S_x do
    select the cluster C_i with lowest V(C_i)
    remove queries in C_i according to some existing query level caching strategy
end for
admit S_x
if there is a cluster C_i in the cache with the highest Sim(S_x → C_i) then
    V(C_i) = V_top + 1
else
    create a cluster C_x for S_x, V(C_x) = V_top + 1
end if
```

we need to justify whether $S_x$ is semantically contained by or close to an existing cluster $C_i$. Basically, we treat $S_x$ as a singular cluster and compute the cluster similarity $Sim(S_x \rightarrow C_i)$ according to Definition 9. If there's more than one cluster $C_i$ such that $Sim(S_x \rightarrow C_i)$ is greater than one, we select the cluster with the highest cluster similarity. Then, we examine whether a positive FRECLE rule $C_i \Rightarrow C_j$ was discovered and $C_j$ is cached. If yes, we calculate a new replacement value for $C_j$ as $V(C_j) = V(C_j) + (V_{top} - V(C_j)) \times conf(C_i \Rightarrow C_j)$. Obviously, $V(C_j)$ is increased and we delayed the eviction of queries in cluster $C_j$ based on the rule. It is similar for negative FRECLE rules. For example, with a negative rule $C_i \Rightarrow \neg C_j$, we update $V(C_j) = V(C_j) + (V(C_j) - V_{top}) \times conf(C_i \Rightarrow \neg C_j)$. Since $V(C_j)$ is decreased, we actually hasten the purge of queries in cluster $C_j$.

| | | |
|---|---|---|
| $N$ | Number of semantic trees | 1K |
| $L$ | Number of potential frequent rooted subtrees | 8 |
| $D$ | Maximum depth of rooted subtrees | 10 |
| $F$ | Maximum fanout of rooted subtrees | 10 |
| $M$ | Average number of nodes of rooted subtrees | 20 |
| $P$ | Maximum overlap between frequent rooted subtrees | 0.2 |
| $O$ | The ratio of outliers | 0.05 |

(a)

| Data | Feature parameters |
|---|---|
| $d_1$ | $P = 0.05$ |
| $d_2$ | $P = 0.1$ |
| $d_3$ | $P = 0.2$ |
| $d_4$ | $O = 0.01$ |
| $d_5$ | $O = 0.05$ |
| $d_6$ | $O = 0.08$ |
| $d_7$ | $N = 10K$ |

(b)

TABLE II

PARAMETERS AND DATA SETS.

For the replacement function for queries in the lower level, some other parameters, such as the processing cost of the query and the size of the query region, should be considered. These issues are out of the scope of our discussion on this application. Interested readers can refers to [29] for the details.

## VI. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the FRECLE mining algorithm and the FRECLE based cache replacement strategies. We have implemented our approach using Java. Experiments are carried out on a Pentium IV 2.8GHz PC with 512 MB memory. The operating system is Windows 2000 professional.

### A. *Performance of* FRECLE *Mining*

Firstly, we investigate the performance of the FRECLE mining algorithm. Recall that the mining algorithm consists of two phases. The first one clusters the semantic trees while the second one discovers FRECLEs from the transformed semantic cluster database. Since we adopt the *FS-Miner* [13] algorithm in the second phase, we focus on evaluating the performance of the first phase.

In order to evaluate our semantic tree clustering method in the first phase, we implemented a synthetic tree generator. The set of parameters used by the generator are presented in Table II.

Basically, synthetic trees are generated with the following steps [4]:

- We generate a set of $L$ potential frequent rooted subtrees by controlling the overlap $P$ between them. The structure of each potential frequent rooted subtree are decided by $D$, $F$ and $M$. For example, at a given node, the number of children is sampled uniformly at random from the range $0$ to $F$. For each child, the process is performed recursively if the depth of the tree is less than or equal to $D$ or the total number of nodes reaches a value sampled from a Poisson distribution with mean of $M$. After generating the first potential frequent rooted subtree, we generate subsequent subtrees based on the previous one with respect to the parameter $P$. For example, if the number of nodes in the previous tree is $M_i$ and the number of nodes in the current tree is $M_{i+1}$, then the number of changed nodes between trees $(1-p)*(M_i+M_{i+1})$. We randomly decide the positions of inserted and deleted nodes.
- We then generate the set of $N*(1-O)$ trees based on potential frequent rooted subtrees produced in the first step. We generate $N*O$ trees randomly with respect to the parameters $D$, $F$ and $M$. Each potential frequent rooted subtree is assigned a weight, which corresponds to the probability that it will be selected to generate a synthetic tree. The weight is picked from an exponential distribution with unit mean. Weights of each frequent rooted subtree is normalized so that the sum of weights is $1$. The next frequent rooted subtree to be used to generate a synthetic tree is chosen by tossing an $L$-sided weighted coin, where the weight of a side is the probability of picking the corresponding frequent rooted subtree. We also associate each potential frequent rooted subtree a corruption level, which is obtained from a normal distribution with mean $0.5$ and variance $0.1$. Then, when constructing a synthetic tree based on the potential frequent rooted subtree, we keep deleting nodes randomly as long as a uniformly distributed random number between $0$ and $1$ is less than the corruption level of the potential frequent rooted subtree.

The third column of Table II shows the default values. Basically, there are $7$ datasets used in the following experiments. The key parameter values used in generating the datasets are shown in Table II $(b)$.

[4]Note that the generated synthetic trees do not really convey meaningful semantics. But it does not affect the evaluation of our algorithm by assuming the generated labeled trees are semantically meaningful.

| | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| *Avg_Intra* | 0.312 | 0.339 | 0.386 | 0.380 | 0.386 | 0.388 |
| *Avg_Inter* | 0.142 | 0.181 | 0.205 | 0.199 | 0.205 | 0.204 |

Fig. 10. Accuracy of clustering method.

**Accuracy of clustering method.** We first conduct experiments to study the accuracy of the clustering method. We evaluate the accuracy of the clustering based on two metrics, *average intra-cluster dissimilarity* and *average inter-cluster similarity*, which are defined as follows.

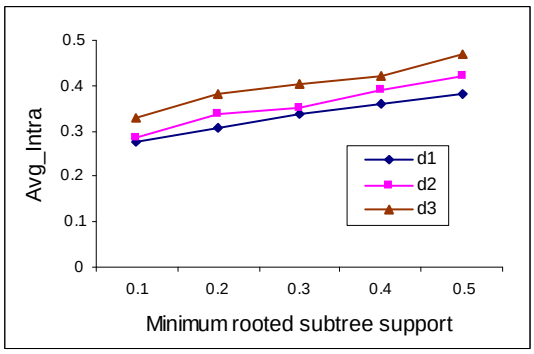
$$Avg\_Intra = \frac{1}{k}\sum_{i=1}^{k} Intra(C_i) \tag{1}$$

$$Avg\_Inter = \frac{2 \times Inter(C_i, C_j)}{k \times (k-1)} \tag{2}$$

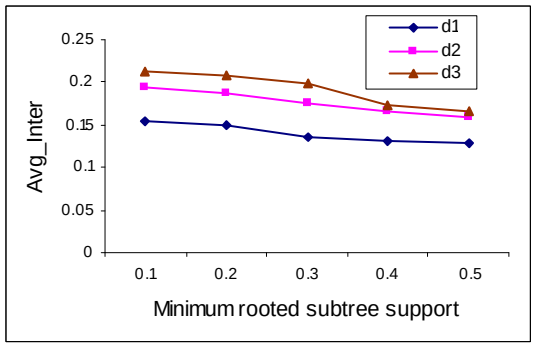
where $Intra(C_i)$ is the intra-cluster dissimilarity as defined in Definition 8 and $Inter(C_i, C_j)$ is the inter-cluster similarity as defined in Definition 10, $k$ is total number of clusters. For a good clustering, both values should be low. We conduct experiments on the set of data sets $d_1$ through $d_6$. The data sets $d_1$, $d_2$ and $d_3$ are generated with different overlap values. While, the data sets $d_4$, $d_5$ and $d_6$ are generated with different outlier ratios. Both the *minimum rooted subtree support* and the *minimum edge support* are set as $25\%$. Figure 10 shows the accuracy of the produced clusters. We have the following observations:

- Generally, our clustering method can achieve both small average intra-cluster dissimilarity and small average inter-cluster similarity.
- When the overlap value between potential frequent rooted subtrees decreases (e.g., from $d_3$ to $d_1$), our method can achieve better accuracy.
- The variation of the outlier ratio, which is used to generate data sets $(d_4, d_5, d_6)$, does not affect the performance of our method obviously.

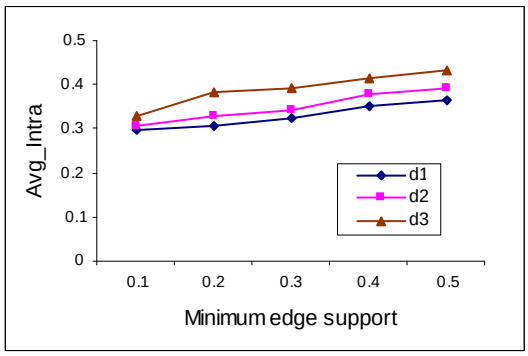
**Sensitivity to parameters.** We conduct the second experiment to study how the variation of parameters affect the accuracy of the clustering. There are two parameters used in the clustering method: *minimum rooted subtree support* $(\delta)$ and *minimum edge support* $(\chi)$. We conduct experiments on data sets $d_1$, $d_2$ and $d_3$ to study the effect of the two parameters respectively. Figures $11(a)$ and $(b)$ shows the $Avg\_Intra$ and the $Avg\_Inter$ by varying $\delta$ respectively. We noticed that when $\delta$ is larger, the $Avg\_Intra$ value increases while the $Avg\_Inter$ decreases. This



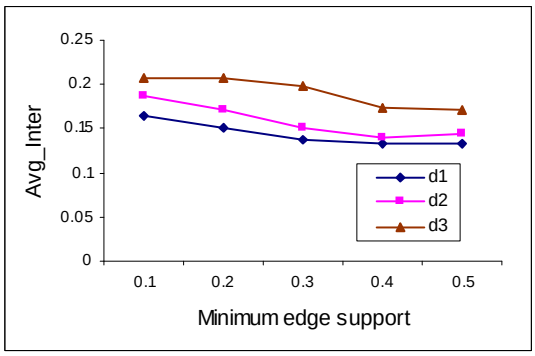

(a) sensitivity of *Avg_Intra* to minimum rooted subtree support




(b) sensitivity of *Avg_Inter* to minimum rooted subtree support




(c) sensitivity of *Avg_Intra* to minimum edge support

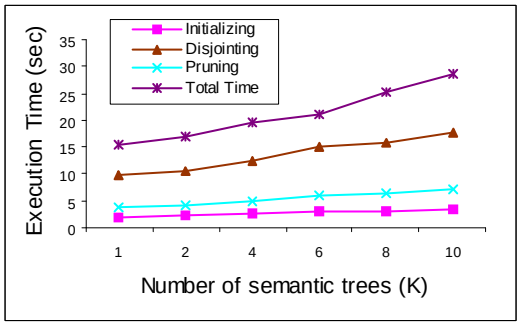



(d) sensitivity of *Avg_Inter* to minimum edge support

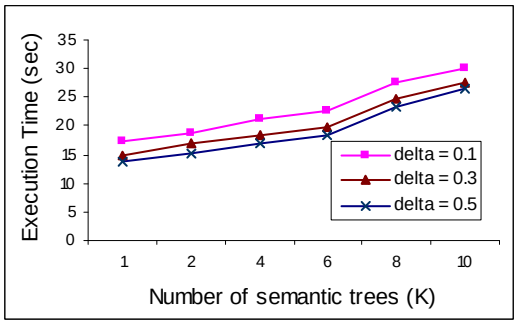
Fig. 11. Sensitivity to parameters.

is because when $\delta$ is larger, fewer frequent rooted subtrees are generated and fewer clusters are found accordingly. Figures $11(c)$ and $(d)$ show the $Avg\_Intra$ and the $Avg\_Inter$ by varying $\chi$ respectively. We observed that when $\chi$ increases, although the $Avg\_Intra$ and $Avg\_Inter$ values vary in the same trend as when $\delta$ increases, they vary within smaller range. That is, the parameter $\delta$ has more effect on the accuracy of the clustering.

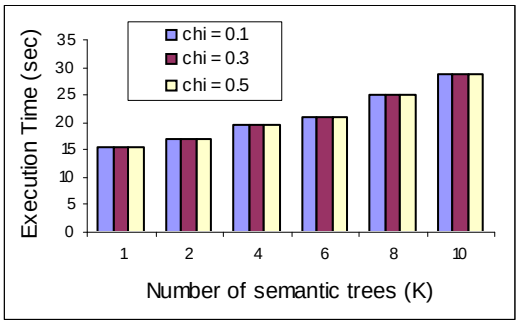
**Efficiency of Clustering Method.** We evaluated the efficiency of our clustering method by conducting experiments on data set $d_7$ and varying the number of trees from $1K$ to $10K$. We first examine the time cost of each clustering step. Both the minimum rooted subtree support and the minimum edge support are set as $25\%$. The experimental results are shown in Figure $12(a)$. The main cost of our clustering method is the disjointing step as it needs to construct the merged trees for clusters. We further evaluate the efficiency of the clustering method with respect to the variation of the two parameters: minimum rooted subtree support $(\delta)$ and minimum edge support $(\chi)$. The experimental results are shown in Figures $12(b)$ and $(c)$ respectively. We notice



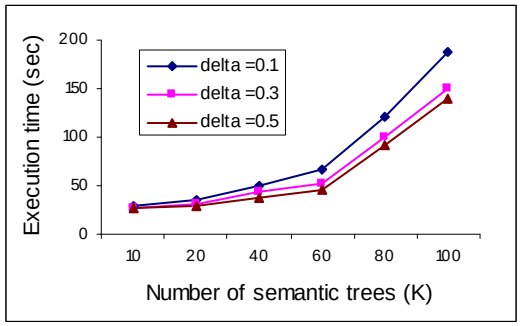

(a) Efficiency of clustering steps




(b) Efficiency with different minimum rooted subtree support (delta)




(c) Efficiency with different minimum edge support (chi)

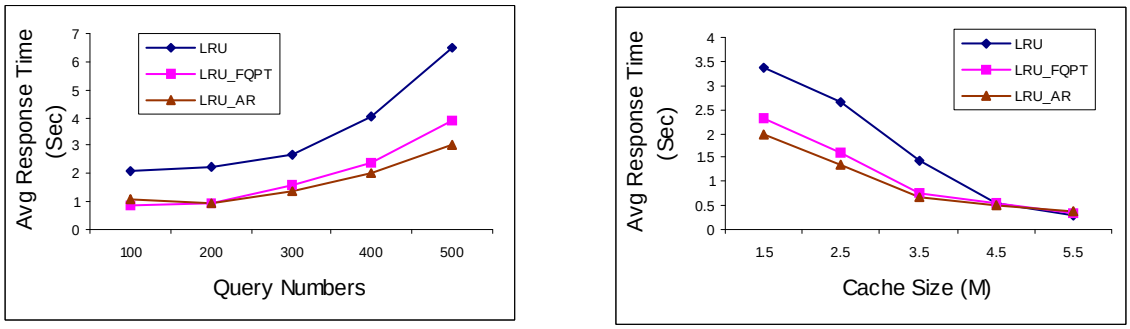



(d) Scalability with different minimum rooted subtree support (delta)

Fig. 12. Performance of clustering.

that when the threshold of rooted subtree support is increased, the clustering method is more efficient. The reason is that when the minimum rooted subtree support is high, fewer clusters will be generated. Thus, both the disjointing step and the merging step need to handle fewer initial clusters. However, the minimum edge support does not affect the efficiency of our clustering method obviously. This is because the computation efficiency of each clustering step does not depend on this parameter.

**Scalability Study.** We evaluate the scalability of the clustering method by duplicating the trees in data set $d_7$ until we get $100K$ semantic trees. Since the parameter of minimum edge support $\chi$ does not affect the execution time of the clustering method, we fix it at $25\%$. Three different minimum rooted subtree support values are used: $\delta = 10\%$, $\delta = 30\%$ and $\delta = 50\%$. The experimental results are shown in Figure $12(d)$. It verifies that our clustering method scales well with respect to the number of trees.




(a) Average response time with respect to query numbers

(b) Average response time with respect to cache size

Fig. 13. Performance of XML query caching strategy.

### B. *Performance of Replacement Strategy*

We then show the effectiveness of our replacement strategy with FRECLE rules mined from XML query sequence database. We used a simple XQuery processor [31] that proceeds queries directly from the source XML file. Hence, no underlying storage strategy or indexing techniques will be involved in affecting the query response time. However, such a processor is not very scalable with respect to the size of the source XML document. Consequently, in our experiment, we generated a fragment of DBLP data as the source XML document. The file size is $10.5M$ and there are totally $248,215$ nodes.

Synthetic XML queries are generated with the similar method described in the previous subsection. Firstly, we generate a set of potential frequent XML queries based on the $DTD$ of $DBLP$. Secondly, we generate a set of potential frequent sequences such that each sequence contain two potential frequent XML queries. Thirdly, a sequence of $1000$ XML queries are generated based on the frequent queries and frequent sequences generated in the first two steps. We then use the first $M$ queries in the sequence as the training data to discover FRECLE rules and the remaining $N$ queries as the test to evaluate the performance of caching.

Three sets of experiments were carried out to investigate the effect of varying the number of queries, varying the size of cache, and varying the size of training data set respectively. We compare our FRECLE association rule-based LRU replacement strategy (denoted as $LRU\_AR$) with another two strategies, LRU and LRU integrated with frequent query patterns [29] (denoted $LRU\_FQPT$). Note that our approach is not competing with $LRU\_FQPT$ but rather complements it. This is because we can integrate $LRU\_FQPT$ with $LRU\_AR$ by using frequent query

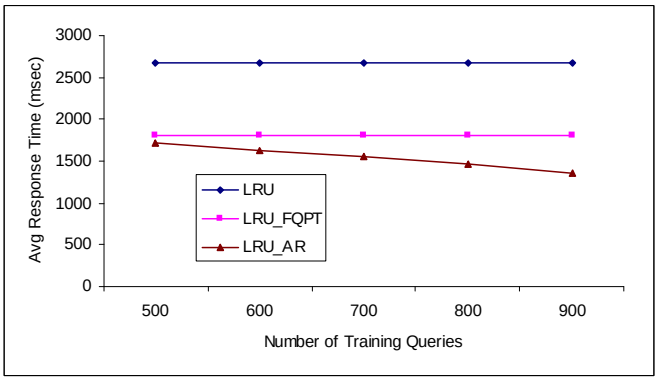



Fig. 14. Average response time with respect to the training data size.

patterns in each semantic cluster to enhance the cache replacement strategy. We use the *average response time*, which is the ratio of total execution time for answering a set of queries to the total number of queries in this set, as the metric.

**Variation of number of queries.** Because of the limited power of the query processor, we vary the number of test queries $N$ from $100$ to $500$ (Note that, we fix the training queries at $900$ and duplicate the test queries). The cache size is fixed at $2.5$MB, which is nearly one quarter of the source XML document. The set of thresholds $\xi$, $\delta$ and $\chi$ are $0.1$, $0.25$, and $0.25$, respectively. The experimental results are shown in Figure $13(a)$. We observed that $LRU\_AR$ have consistently superior performance than $LRU$. Also, when the number of queries is larger than $200$, the strategy $LRU\_AR$ works best. Particularly, when the number of queries is $500$, $LRU\_AR$ works nearly $1$s faster than $LRU\_FQPT$. Note that this improvement is not small as the query processor averagely takes $0.005$s to process an XML query. Furthermore, according to the trend shown in Figure 13, when more test queries are used, more efficiency gains will be achieved by $LRU\_AR$.

**Variation of cache size.** In this experiment, we vary the size of cache from $1.5$M to $5.5$M, the number of test queries $N$ is fixed at $300$. The set of thresholds are same above the above experiment. As shown in Figure $13(b)$, the more limited the cache size, the greater gap in average response time between $LRU\_AR$ and the rest.

**Variation of training data size.** In this experiment, we evaluate the performance of our caching strategy with respect to the variation of the training data size. We vary the number of training

queries $M$ from $500$ to $900$ and use the subsequent $100$ queries as the test data (In order to show the difference clearly, the test data is duplicated until we get $300$ testing queries). We fix the cache size at $2.5$M. The experimental results are shown in Figure 14. It can be observed that our XML caching strategy works better with a larger training data size. The reason is that when the number of training queries increases, the discovered FRECLE rules are more accurate and the corresponding caching strategy is more effective.

## VII. Related Work

### A. Frequent Pattern Mining for Tree Structured Data

Most existing work focus on discovering the frequent substructures from a collection of semi-structured data such as XML documents. Wang and Liu [25] developed an Apriori-like algorithm to mine frequent substructures based on the "downward closure" property. They first found the frequent *1-tree-expressions* that are frequent individual *label path*s. Discovered frequent *1-tree-expression*s are joined to generate candidate *2-tree-expression*s. The process is executed iteratively till no candidate *k-tree-expression*s is generated. Asai et al. [2] developed another algorithm, FreqT, to discover all frequent tree patterns from large semi-structured data. They modeled the semi-structured data as *labeled ordered tree* and discover frequent trees level by level. At each level, only the rightmost branch is extended to discover frequent trees of the next level. Thus, efficiency can be achieved without generating duplicate candidate frequent trees.

TreeMinerH and TreeMinerV [30] are two algorithms for mining frequent trees in a forest. TreeMinerH is an Apriori-like algorithm based on a horizontal database format. In order to efficiently generate candidate trees and count their frequency, a smart *string encoding* is proposed to represent the trees. In contrast, TreeMinerV uses vertical *scope-list* to represent a tree. Frequent trees are searched in depth-first way and the frequency of generated candidate trees are counted by joining *scope-lists*. TreeFinder [22] is an algorithm to find frequent trees that are *approximately* rather than *exactly* embedded in a collection of tree-structured data modeling XML documents. Each labeled tree is described in *relaxed relational description* which maintains ancestor-descendant relationship of nodes. Input trees are clustered if their atoms of *relaxed relational description* occur together frequently enough. Then maximal common trees are found in each cluster by using algorithm of *least general generalization*. Recently, there is another line of work that employs the pattern-growth strategy to discover frequent subtrees [23], [27].

The critical difference between our proposed frequent semantic tree cluster sequences mining and existing works on XML data mining is as follows. Firstly, all the above works focus on mining a set of structural tree pattern whereas we focus on mining a set of semantic tree pattern sequences where each sequence contains a set of semantic trees that are temporally associated. Secondly, rather than finding frequent pattern from a collection of trees, we focus on clustering the semantic trees into semantically-related clusters and then discover frequent cluster sequences. These clusters represent how a group of semantically-related trees is associated with another group of semantically-related trees.

### *B. Caching XML Data*

Semantic caching was proposed in [11] as a more flexible model than the previous tuple [12] and page [4] caching systems. Semantic caching manages the cache at the data granularity of group of tuples. Hence, it incurs less space overhead than tuple caching for cache management (such as buffer control blocks, hash table entries etc). Due to its flexibility, semantic caching is popular in Web query caching [9] and recently XML query caching [14] [6]. For example, Chidlovskii and Borghoff [9] discussed using semantic caching to group together semantically related Web documents covered by a boolean Web user query (i.e., the documents dynamically generated by cgi-scripts after the user filling out a search form). Hristidis and Petropoulos [14] proposed a compact structure, *modified incomplete tree (MIT)*, to represent the semantic regions of XML queries. ACE-XQ [6] is a holistic XQuery-based semantic caching system. They discussed how to judge whether a new query is contained by any cached query and how to rewrite the new query with respect to the cached queries. They also proposed a fine-grained replacement strategy rather than replacing a complete query region at a time. However, this work did not consider using the knowledge mined from historical user queries to design the replacement function.

Recently, intelligence has been incorporated into Web/XML query caching by constructing predictive models of user requests with the knowledge mined from historical queries [17] [3] [29]. Lan et al. [17] mined association rules from Web user access patterns. Then they prefetched Web documents based on discovered associations and current requested documents. They focused on the placement strategy (fetching and prefetching) while we focused on the replacement strategy. Bonchi et al. [3] mined association rules from Web log data to extend the traditional LRU

replacement strategy. For example, given a discovered association rule *URL A* $\Rightarrow$ *URL B*, when a user queries the resource at *URL A*, if the resource at *URL B* is in the cache, its eviction should be delayed. However, their work cannot be applied in XML query caching directly because answers to XML query do not have explicit identifiers such as URL. Hence, our work is different from this one in that we mine association rules between query groups in which queries are semantically close. Furthermore, we also use negative association rules to demote the replacement values of corresponding query regions.

## VIII. CONCLUSIONS

In this paper, we have introduced frequent semantic tree cluster sequences (FRECLE) that represent semantic associations between tree-structured data. Given a tree sequence database, where each tree represents some semantic meaning, we have proposed a technique to discover FRECLE patterns from the underlying database. Our approach consists of two phases. In the first phase, each semantic tree is categorized to a semantic cluster. To this end, we proposed a clustering algorithm on the set of tree structures in the database so that trees in the same cluster represent similar semantics. As a result, the semantic tree sequence database are transformed into a semantic cluster database. Next, FRECLE patterns are discovered from the transformed database by adopting an existing frequent sequential pattern mining algorithm.

FRECLEs discovered from semantic tree sequence database can be useful in several applications such as XML query cache replacement strategy, prefetching XML data, and web user clustering. Particularly, we showed that FRECLEs can be used in the context of XML query sequence database to infer that when a user issues a query of some particular information, he/she will query about some other information of the XML document subsequently. Then, FRECLEs can be used in designing optimal XML query cache replacement strategies. We first derive both positive and negative association rules from discovered FRECLEs. Then, we promote the replacement value for queries in the consequences of positive rules and demote the replacement value for those of negative rules. As verified by the experimental results, FRECLE-based replacement strategy improved the cache performance.

## REFERENCES

[1] R. Agrawal and R. Srikant. Mining Sequential Patterns. *In ICDE*, 1995.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient Substructure Discovery from Large Semi-structured Data. *In SIAM DM*, 2002.

[3] F. Bonchi, F. Giannotti, C. Gozzi et al. Web Log Data Warehousing and Mining for Intelligent Web Caching. *Data ans Knowl. Eng.*, 39(2):165–189, 2001.

[4] M. J. Carey, M. J. Franklin, and M. Zaharioudakis. Fine-Grained Sharing in a Page Server OODBMS. *In SIGMOD*, 1994.

[5] L. Chen, S. S. Bhowmick, and L.-T. Chia. Mining Positive and Negative Association Rules from XML Query Patterns for Caching. *In DASFAA*, 2005.

[6] L. Chen, E. A. Rundensteiner, and S. Wang. Xcache: a Semantic Caching System for XML Queries. *In SIGMOD*, 2002.

[7] L. Chen, S. Wang, and E. Rundensteiner. Replacement Strategies for XQuery Caching Systems. *Data Knowl. Eng.*, 49(2):145–175, 2004.

[8] Y. Chi, Y. Yang, Y. Xia, and R. Muntz. CMTreeMiner: Mining Both Closed and Maximal Frequent Subtrees. *In PAKDD*, 2004.

[9] B. Chidlovskii and U. M. Borghoff. Semantic Caching of Web Queries. *VLDB Journal*, 9(1):2–17, 2000.

[10] T. Dalamagas, T. Cheng, K. Winkel, and T. K. Sellis. Clustering XML Documents by Structure. *In SETN* , 2004.

[11] S. Dar, M. J. Franklin, B. Thór et al. Semantic Data Caching and Replacement. *In VLDB*, 1996.

[12] D. DeWitt, P. Futtersack, D. Maier, and F. Velez. A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems. *In VLDB*, 1990.

[13] M. El-Sayed, C. Ruiz, and E. A. Rundensteiner. FS-Miner: Efficient and Incremental Mining of Frequent Sequence Patterns in Web Logs. *In WIDM*, 2004.

[14] V. Hristidis and M. Petropoulos. Semantic Caching of XML Databases. *In WebDB*, 2002.

[15] B. C. M. Fung, K. Wang, and M. Ester. Hierarchical Document Clustering using Frequent Itemsets. *In SDM*, 2003.

[16] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns Without Candidate Generation. *In SIGMOD*, 2000.

[17] B. Lan, S. Bressan, B. C. Ooi, and K.-L. Tan. Rule-Assisted Prefetching in Web-Server Caching. *In CIKM*, 2000.

[18] B. Mandhani and D. Suciu. Query Caching and View Selection for XML Databases. *In VLDB*, 2005.

[19] V. Ng, C. C. Kong, and S. H. Wang. Prefetching XML Data with Abstract Query Mining. *In ITCC*, 2004.

[20] J. Pei, J. Han, B. Mortazavi-Asl, et al. PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. *In ICDE*, 2001.

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

[22] A. Termier, M-C. Rousset, and M. Sebag. TreeFinder: a First Step towards XML Data Mining. *In ICDM*, 2002.

[23] C. Wang, M. Hong, J. Pei, H. Zhou, W. Wang, and B. Shi. Efficient Pattern-Growth Methods for Frequent Tree Pattern Mining. *In PAKDD*, 2004.

[24] L. Wang, D. W.-L. Cheung, N. Mamoulis, and S.-M. Yiu. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure.In *IEEE TKDE, 16(1): 82-96*, 2004.

[25] K. Wang and H. Liu. Discovering Structural Association of Semistructured Data. *IEEE TKDE,* vol.12, 2000.

[26] K. Wang, C. Xu, and B. Liu. Clustering Transactions Using Large Items. *In CIKM*, 1999.

[27] Y. Xiao, J. F. Yao, Z. Li, and M. H. Dunham. Efficient Data Mining for Maximal Frequent Subtree. *In ICDM*, 2003.

[28] L. Yang, M. Lee, W. Hsu, and S. Acharya. Mining Frequent Query Patterns from XML Queries. *In DASFAA*, 2003.

[29] L. Yang, M. Lee, and W. Hsu. Efficient Mining of XML Query Patterns for Caching. *In VLDB*, 2003.

[30] M. J. Zaki. Efficiently Mining Frequent Trees in a Forest. *In SIGKDD*, 2002.

[31] XP: A Simple XQuery Processor Implemented in Java. http://www.cs.wisc.edu/∼mcilwain/classwork/cs764/.