Mining Unordered Distance-constrained Embedded Subtrees

Fedja Hadzic[1], Henry Tan[1], and Tharam Dillon[1]

[1] DEBII, Curtin University of Technology, Perth, Australia
{f.hadzic, h.tan, t.dillon}@curtin.edu.au

Abstract. Frequent subtree mining is an important problem in the area of association rule mining from semi-structured or tree structured documents, often found in many commercial, web and scientific domains. This paper presents the u3Razor algorithm, for mining unordered embedded subtrees where the distance of nodes relative to the root of the subtree needs to be considered. Mining distance-constrained unordered embedded subtrees will have important applications in web information systems, conceptual model analysis and more sophisticated knowledge matching. An encoding strategy is presented to efficiently enumerate candidate unordered embedded subtrees taking the distance of nodes relative to the root of the subtree into account. Both synthetic and real-world datasets were used for experimental evaluation and discussion.

1 Introduction

To express more complex and meaningful relationships between the data objects, many organizations represent their domain knowledge using semi-structured documents. Semi-structured documents such as XML possess a hierarchical document structure, where an element may contain further embedded elements, and each element can be attached with a number of attributes. It is therefore frequently modeled using a rooted ordered labeled tree. To support effective and efficient data analysis from tree structured documents algorithms have been developed to extract all subtree patterns that occur in the database of ordered labeled trees as many times as the user supplied support threshold. This is known as the frequent subtree mining (FSM) problem and is the first and most important and complex problem to consider when discovering useful associations among data objects from a tree structured document [1,2,3]. In many biological data analysis tasks, the aim is to find frequent structured patterns, such as frequent protein or chemical compound structures from the data. For example, the work presented in [4] demonstrates the potential of the tree mining algorithms for discovering substructures from Protein data that could be useful for discovering interesting similarities and differences in protein datasets taken across protein families and species. Tree mining has also been successfully applied in [5] for the analysis of phylogenetic databases. Driven by different application needs many algorithms have been developed that can mine different subtree types. HybridTreeMiner [6], uFreqt [7], and uNot[8], mine induced unordered trees.

Treeminer [9] is an efficient algorithm for discovering all frequent embedded subtrees in a forest using a data structure called the vertical scope-list. The SLEUTH [10] algorithm extracts all frequent unordered embedded subtrees by using unordered scope-list joins via the descendant and cousin tests. Our contribution to the area of frequent subtree mining is the introduction of a tree model guided (TMG) candidate subtree enumeration framework which was used for developing efficient algorithms for mining of ordered induced/embedded [2], ordered distance-constrained embedded subtrees [11] unordered induced [12] and unordered embedded [13] subtrees. TMG ensures that only those subtrees are enumerated which conform to the underlying tree structure of the document [3]. For a more extensive overview of the state-of-the-art of tree mining please refer to [14].

In an ordered subtree the left-to right order among the sibling nodes needs to be preserved while in an unordered subtree the order of the sibling nodes (and the subtrees rooted at those nodes) can be exchanged and the resulting subtree is still considered the same. This causes the enumeration and counting of unordered subtrees more difficult, since each enumerated subtree needs to be ordered into one logical and consistent form, so that all its variants that have different order among sibling nodes are considered as the same subtree. An induced subtree preserves the parent-child relationships from the original tree while in an embedded subtree the parent-child relationship are allowed to be ancestor-descendant relationships in the original tree. By mining embedded subtrees one can detect commonly occurring relationships between data objects in spite of the difference in the level where the relationship in the document occurred. Certain concepts may be represented in a more specific/general way in certain documents. This specific information is often in the form of additional child nodes of the concept, and hence, two general and related concepts may be separated by a number of levels in the document tree. If the user is only interested in the relationship between these two general concepts, such a relationship could be directly found in an embedded subtree set, while if induced subtrees were extracted, the information irrelevant to the user may be present in the patterns of interest.

While mining of embedded subtrees is a generalization over induced subtrees, one limitation is that the context information may be lost in some patterns. For example, when analyzing a biomedical database containing patient records of potential illness causing factors, one would be interested in common set of data object properties that have frequently been associated with a particular disease. By allowing ancestor-descendant relationships it may be possible to loose some information about the context in which particular disease characteristic occurred. This is mainly due to the fact that some attributes of the dataset may have a similar set of values and hence indicating which value belonged to which particular attribute is necessary. There appears to be a trade-off here and in this case allowing ancestor-descendant relationships can result in unnecessary and misleading information, but in other cases, it proves useful as it detects common patterns in spite of the difference in granularity of the information presented. A difficulty with embedded subtrees appears then to be that there is too much freedom allowed with respect to the difference in the distances between the nodes. All occurrence of one particular relationship are considered the same and valid even if the distance between the related data objects is so different that it is possible that it occurred in a different context and has a different intended

meaning. One way to avoid this characteristic is to further distinguish the embedded subtrees based upon the distance between the nodes. Making this distinction will have important applications in web information systems, conceptual model analysis, knowledge matching and for general knowledge management tasks by allowing for more specialized queries.

In this study we extend our past work by developing the first algorithm that will extract all unordered embedded subtrees with node distance information. It adds more granularity to the problem as the occurrences of the embedded subtrees with different distances among the nodes are now considered as different candidates (hence the name distance-constrained). Overall, mining of unordered distance-constrained subtrees is more expensive in terms of space and time required than when mining any of other subtree types. In Section 2 we define some tree mining related concepts and provide a motivating example for mining of unordered distance-constrained embedded subtrees. The proposed u3Razor algorithm is described in Section 3 together with a suitable encoding strategy for enumerating unordered embedded subtrees that take the node distance information from the database tree into account. Section 4 presents some experiments to test the scalability of the approach and compare with the results obtained by not imposing the distance constraint. Section 5 concludes the paper.

2 Problem definition and motivation

A tree T is an acyclic connected graph with the node at the top defined as the *root[T]*. The *Parent* of node v (*parent[v]*) is defined as its predecessor. Two nodes that share the same parent are referred to as *sibling nodes*. The fan-out or degree of a node corresponds to the number of children of that node. A *leaf node* is a node without a child; otherwise, it is an internal node. A path from vertex v_i to v_j, is defined as the finite sequence of edges that connects v_i to v_j. The length of a path p is the number of edges in p. The distance between two nodes v_i and v_j can then be defined as the length of the path connecting v_i and v_j. If p is an ancestor of q and q is a descendant of p, then there exists a path from p to q. The *rightmost path* (RMP) of T is defined as the (shortest) path connecting the rightmost leaf with the root node. The *Depth/level* of a node is the length of the path from root to that node. The *size* of a tree equals to the total number of nodes in the tree. In this paper, the term 'k-subtree' refers to a subtree that consists of k number of nodes. A tree can be denoted as *T(V,L,E)*, where (1) V is the set of vertices or nodes; (2) L is the set of labels of vertices, for any vertex $v \in V$, $L(v)$ is the label of v; and (4) $E = \{(x,y) | x,y \in V\}$ is the set of edges in the tree. The problem of **frequent subtree mining** can be generally stated as: Given a tree database T_{db} and minimum support (σ) extract all candidate subtrees that occur at least σ times in T_{db}.

Within the tree mining framework, two support definitions often used are transaction-based and occurrence match support. When using the ***transaction-based support*** definition, the transactional support of a subtree t, denoted as $\sigma_{tr}(t)$ in a tree database T_{db} is equal to the number of transactions in T_{db} that contain at least one occurrence of subtree t. Let the notation $t \prec k$, denote the support of subtree t by

transaction k, then $t \prec k = 1$ whenever k contains at least one occurrence of t, and 0 otherwise. Given N transactions k_1 to k_N of tree in Tdb, the $\sigma_{tr}(t)$ in T_{db} is defined as $\sum_{i=1}^{N} t \prec k_i$.

The ***occurrence-match support*** takes the repetition of items in a transaction into account and counts the subtree occurrences in the database as a whole. Hence, the occurrence-match support of a subtree t, denoted as $\sigma_{oc}(t)$, in a tree database T_{db} is equal to the total number of occurrences of t in all transactions in T_{db}. Let function $g(t,k)$ denote the total number of occurrences of subtree t in transaction k. If there are N transactions k_1 to k_N of tree in T_{db}, $\sigma_{oc}(t)$ in T_{db} can be defined as $\sum_{i=1}^{N} g(t, k_i)$.

Next, we provide some formal definitions of commonly mined subtree types.

Given a tree $S = (V_S, L_S, E_S)$ and tree $T = (V_T, L_T, E_T)$, S is an ***induced subtree*** of T, iff (1) $V_S \subseteq V_T$;(2) $L_S \subseteq L_T$, *and* $L_S(v)=L_T(v)$; and (3) $E_S \subseteq E_T$.

Given a tree $S = (V_S, L_S, E_S)$ and tree $T = (V_T, L_T, E_T)$, S is an ***embedded subtree*** of T, iff (1) $V_S \subseteq V_T$;(2) $L_S \subseteq L_T$, *and* $L_S(v)=L_T(v)$; (3) *if* $(v_1, v_2) \in E_S$ then $parent(v_2) = v_1$ in S and v_1 is ancestor of v_2 *in* T. Hence, the main difference between an induced and an embedded subtree is that, while an induced subtree keeps the parent-child relationships from the original tree, an embedded subtree allows a parent in the subtree to be an ancestor in the original tree. All the definitions provided above do not take into account the order among the sibling nodes. This is what makes them ***unordered*** subtrees. In an ***ordered*** subtree the left to right ordering of sibling nodes in the original tree is preserved. As mentioned in the introduction, when a distance constraint is imposed on an embedded subtree, the distance information between the nodes in the original subtree needs to be stored and used as an additional candidate grouping criterion.

Given a tree $S = (V_S, L_S, E_S)$ and tree $T = (V_T, L_T, E_T)$, S is an ***unordered distance-constrained embedded subtree*** of T iff (1) $V_S \subseteq V_T$;(2) $L_S \subseteq L_T$, and $L_S(v)=L_T(v)$; (3) if $(v_1, v_2) \in E_S$ then $parent(v_2) = v_1$ in S and v_1 is ancestor of v_2 in T; and (4) $\forall\ v \in V_S$ there is an integer stored indicating the distance between v and the root node of S in the original tree T.

For an ordered distance-constrained embedded subtree, in addition to the above pre-conditions the left-to-right ordering among the sibling nodes in the original tree would also need to be preserved [11]. To illustrate the difference in mining of different subtree types please consider the example tree in Fig. 1 where the label of each node is shown with its pre-order position on the left. In this paper, the term 'occurrence coordinate(s) *(oc)*' will be used to refer to the position(s) of a particular node or a subtree in the tree database. In the case of a node, *oc* corresponds to the pre-order position of that node in the tree database, whereas for a subtree, *oc* is a sequence of *ocs* from nodes that belong to that particular subtree. If ordered or unordered induced subtrees are mined *st* occurs only once in T with *oc*:01569, while for ordered embedded subtrees it also occurs at *oc*:01589 since the ancestor-descendant relationships between nodes c (*oc*:5) and d (*oc*:8) are allowed. With unordered embedded subtrees the order can also be exchanged and hence *st* also occurs at *oc*:01587. If unordered distance constrained subtrees are mined each of the three occurrences of *st* will be considered as a separate subtree depending on the distance of

the nodes to the root of the subtree as detected in T (i.e. st_1, st_2 and st_3 in Fig.1.). The numbers next to the link indicates the distance between the nodes connected by that link in T. Hence, st_1 is a representative of the subtree with oc:01569, st_2 of oc:01589 and st_3 of oc:01587.

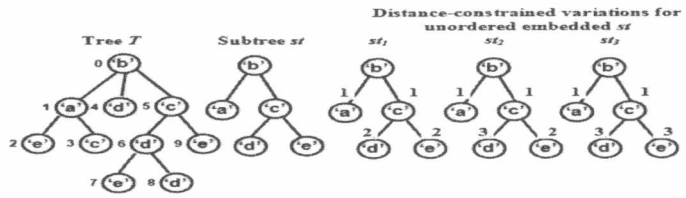

Fig. 1. Example tree T and subtree st with distance-constrained variants

For unordered subtrees the enumeration and counting phase is more difficult than for the ordered case, since each enumerated subtree needs to be ordered into one logical and consistent form, so that all its variants that have different order among sibling nodes are considered as the same subtree. The group of possible trees obtained by permuting the sibling nodes in all possible ways is referred to as the automorphism group of a tree [10]. During the pre-order traversal of a database, ordered subtrees are generated by default. It is necessary to identify which of these ordered subtrees form an automorphism group of an unordered subtree. One tree needs to be selected to uniquely represent the unordered tree. This selected tree is known as the canonical form (CF) of an unordered tree. A canonical form (CF) of an entity is in general a representative form (or a function) for which many equivalent variations of an entity can be represented (mapped) into one standard, conventional, logical form in a consistent manner [15]. The CF used by the proposed algorithm will be explained in Section 3.

To conclude this section, we provide an example that illustrates a case where adding the distance constraint is important for effective data analysis with respect to the application needs. In Fig. 2, two example trees are displayed that indicate a part of the ancestor family tree from two ill patients (the examples come from an image of a disease family tree obtained from [16]. Such information is used for linkage analysis of an illness by performing gene testing which can provide information about one having a disease-related gene mutation. When looking for a disease gene, scientists often start by studying DNA samples from family members over several generations who have a number of relatives who have developed an illness [16].

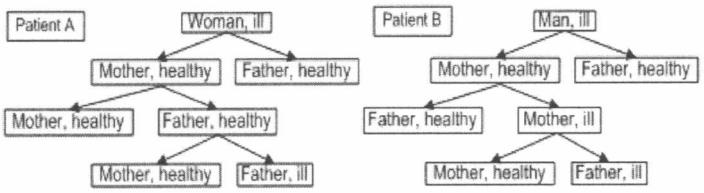

Fig. 2. Example ancestor representation of two ill patients (A and B)

For example, a scientist may want to discover how many ill relatives an ill patient has had and to discover the number of generations that separates them. Using the traditional embedded subtree definition, we can extract information only about the number of ill relatives, but cannot have the information about the number of generations that separate the patient and the relatives that have a common disease. This is because the traditional embedded subtree definition does not have this kind of expressive capability. In contrast, by utilizing the distance-constrained embedded subtrees, we can find out exactly how many generations they are separated by, by inspecting the distance information stored between the nodes. From Fig. 2, patient A has only one diseased ancestor and it is her great-grandfather, while patient B has two diseased ancestors, a grandmother and great-grandfather. Even though we do not have such an example in the figure, it is worth noting that it could well be the case that an ill patient will have two ancestors of the same gender that have the illness. In this case, the traditional embedded subtree definition would group these subtree occurrences as one candidate and indicate wrongly that there is only one ancestor with a disease. On the other hand, by mining distance-constrained embedded subtrees, both occurrences will be considered as separate entities due to the difference in the distance to the root node which is used as an additional candidate grouping criterion. Generally speaking, an algorithm for mining of unordered distance-constrained embedded subtrees will have some important applications in analysis of biological sequences, web information systems and conceptual model analysis.

3 u3Razor Algorithm

The steps taken by the u3Razor algorithm are presented in Fig. 3. The tree database is first transformed into a database of rooted integer-labelled trees as hashing integer-labelled trees is much faster than hashing of string-labelled trees. It is then ordered into its canonical form (CF) to reduce the average number of candidate trees that need to be ordered. Recursive List (*RL*) is constructed which is a global sequence of encountered nodes in the pre-order traversal together with the necessary node information. During this process the node labels are hashed to obtain the set of frequent 1-subtrees (*F1*). TMG candidate generation using the *RL* structure takes place and the string representatives of candidate subtrees with the distance information between the nodes are hashed to the Ck hash table and their occurrences are stored. Prior to hashing the string representation of each candidate subtree, it is first ordered into its CF, if necessary. The process repeats until all frequent k-subtrees are enumerated. To enable the mining of unordered distance constrained embedded subtrees the major change to our general TMG framework [2, 3, 12, 13] took place in the way that candidate subtrees are represented at the implementation level to take into account the distance information and the CF used, which is explained next. We then explain the *RL* structure and the TMG process for enumerating a complete set of unordered distance-constrained embedded subtrees.

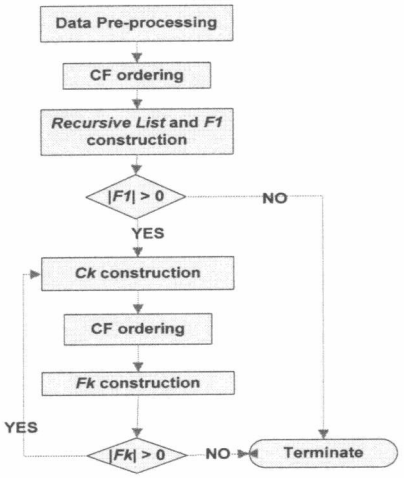

Fig. 3. General description of the steps taken in the proposed approach

Tree Representation and CF Ordering. Our work utilizes the pre-order string encoding (φ) as described in [2,9], which is a sequential representation of the nodes of a tree as encountered during the pre-order traversal of that tree. The backtrack symbol ('/') is used whenever moving up a node in the tree during the pre-order traversal. To take the distance between the nodes into account, the encoding of a subtree is obtained by reading the nodes in the pre-order traversal and for each node storing the distance to the root of the subtree (node depth). The distance to the root is worked out from the node levels stored in the *RL* structure, where the root of the subtree is assigned the depth of 0 and all other nodes are assigned the difference between their level and the original level of the new subtree root. Further modification of the encoding consists in storing a number next to each backtrack '/' symbol indicating the number of backtracks in the subtree, as opposed to storing each of those backtracks as a separate symbol. This representation allows easier string manipulation due to uniform block size. We denote encoding of a subtree *T* as *φ(T)* and eg. from Fig. 1, *φ(T):* 'b0 a1 e2 /1 c2 /2 d1 /1 c1 d2 e3 /1 d3 /2 e2 /2', *φ(st1):* 'b0 a1 /1 c1 d2 /1 e2 /2', and *φ(st2)*: ' 'b0 a1 /1 c1 d3 /1 e2 /2'. The backtrack symbols can be omitted after the last node eg. *φ(st3)*: ' 'b0 a1 /1 c1 d3 /1 e3'.

The canonical form ordering occurs at the start where the whole tree database is ordered into its canonical form and later where candidate subtrees are ordered so that unordered subtrees are correctly enumerated. The canonical form according to which we order the trees uses the idea of the DFCF [1] where the nodes are sorted at each level of the subtree in a bottom up fashion (i.e. starting from the leaf nodes), and the nodes with labels that sort lexicographically smaller are placed to the left of the subtree. The ordering process is determined by the means used for comparing nodes or subtrees so that they are placed at the right position in the tree. At the implementation level the process can be formally explained as: given two trees T1 and T2, with root[T1] = r1 and root[T2] = r2, let C(r1) and C(r2) denote the children sets of r1 and r2, respectively. Further, let $\varphi(Tx)_k$ denote the k^{th} element of the pre-

order string encoding of tree Tx (x = 1 or 2)(this can be either a node label or the special backtrack ('/') symbol which is considered smaller than any other label). In case the node labels are the same the distance information associated with each node will be considered so that the nodes with smaller distances to the root of the tree are placed to the left. T1 is considered smaller than T2 iff either:

a.) $L(r1) < L(r2)$, or

b.) $L(r1) = L(r2)$ and either $size(C(r1)) < size(C(r2))$ and $\varphi(T1)_k = \varphi(T2)_k$ for all $1 \leq k \leq length(\varphi(T1))$, or $\varphi(T1)_k < \varphi(T2)_k$ for some $1 \leq k < length(\varphi(T1))$

This ordering scheme will ensure that all the instances of unordered distance-constrained embedded subtrees are correctly represented and counted.

Recursive List (RL) and F1 Construction. The tree database, T_{db}, is scanned once to create the global sequence RL in memory, through which nodes' related information can be directly accessed. Each node is stored following the pre-order traversal of the T_{db}. *Position*, *label*, *scope*, and *level* information are stored for each node. The scope of a node refers to the position of its rightmost leaf node or its own position if it is a leaf node itself [2,9] whereas the level refers here to the level in the T_{db} tree, where this node occurs. An item in RL at position i is referred to as *RL[i]*. Every time a node is inserted into the *RL*, we generate a candidate 1-subtree. Based on its label, we increment its support count in the C_1 hash table. If its support count is $\geq \sigma$ (user-specified minimum support count), we insert the candidate 1-subtree to the frequent 1-subtree set, F_1. An example RL structure representing the tree T from Fig. 1 is displayed in Fig. 4. The pre-order position of a node in the tree database is equal to the index of the RL at which that nodes is stored, and the label, scope and level are shown in that order underneath the entry. All this information is necessary to enumerate only valid subtree candidates and is accessed in the TMG candidate enumeration process explained next.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
b, 9,0	a,3,1	e,2,2	c,3,2	d,4,1	c,9,1	d,8,2	e,7,3	d,8,3	e,9,2

Fig. 4. Recursive List representation of tree T (Fig. 1)

A particular subtree, as defined by its encoding can be found at many places in the database and these different occurrences need to be stored so that subsequent set of candidates can be generated. We only store the occurrence coordinates of the nodes in the right-most path of the subtree (referred to as *RMP-oc*). Within our framework, this information is sufficient for enumerating candidate *(k+1)*-subtrees from a frequent *k*-subtree. Given a k-subtree T with *oc* $[e_0, e_1, \ldots e_{k-1}]$, the *RMP-oc* of T, denoted by $\Psi(T)$, is defined by $[e_0, e_1, \ldots, e_j]$ such that $\Psi(T) \subseteq oc(T)$; $e_j = e_{k-1}$; and $j \leq k$-1 and the path from e_j to e_0 is the *RMP* of tree T. Vertical Occurrence List (*VOL*) is used to store all $\Psi(T)$ of a subtree T represented by its pre-order string encoding $\varphi(T)$, and to determine the occurrence-match and transaction-based support. A transaction identifier (*tid*) is stored for each $\Psi(T)$ so that the occurrence match of T equals to $|VOL|$ while the transaction-based support equals to the number of unique *tid*s in *VOL*.

TMG Candidate Subtree Generation. TMG is a specialization of the right most path extension method which has been reported to be complete and non-redundant [2,9]. To enumerate all embedded k-subtrees from a (*k-1*)-subtree, the TMG enumeration approach extends all the nodes in the *RMP* of a (*k-1*)-subtree, by one node at a time. Hence, it is a breadth-first (BF) enumeration strategy. Suppose that nodes in the *RMP* of a subtree are defined as *extension points*. The TMG can be formulated as follows. Let $\Psi(T_{k-1}):[e_0,e_1,...e_j]$ denote the *RMP-oc* of a frequent (k-1)-subtree T_{k-1}, and Φ the *scope* of the root node e_0. TMG generates k-subtrees by extending each extension point $n \in \Psi(T_{k-1})$ with a node with *oc t* iff $n < t \leq \Phi$. Suppose that the encoding of T_{k-1} is denoted by $\varphi(T_{k-1})$ and $l(e_j,t)$ is a labeling function for extending extension point *n (i.e.* e_j) with a node at position *t*. $\varphi(T_k)$ would be defined as $\varphi(T_{k-1})+l(e_j,t)$, where $l(e_j,t)$ determines the number of backtrack symbols '/' to be appended before the label of the new node is added to $\varphi(T_k)$. The number of backtrack symbols is calculated as the shortest path length between the extension point *n* and the right-most-node *r*, (notation *pl(n,r)*). To generate RMP at each step of candidate generation, we utilize the computed number of backtrack symbols *b* that need to be appended before the new node with *oc t* is added to the encoding. Given that the $\Psi(T_{k-1})$ is $[e_0,e_1,...,e_j]$, the RMP of the k-subtree ($\Psi(T_k)$) is generated by appending *t* at position $(j+1) - b$ of the ($\Psi(T_{k-1})$) and removing any *RMP-oc* that occur after *t*, thereby making *t* the right most node of T_k. This will make sure that at each extension of (k-1)-subtree, *RMP-oc* of k-subtree are appropriately stored.

To provide an illustrative example let us say that we are extending the T_{k-1} subtree from Fig. 5, with $\Psi(T_{k-1}):[0,4,5]$ (*oc*:0145) and $\varphi(T_{k-1})$:'a0 b1 /1 b1 c2'. The label, level and scope information is obtained from the *RL* entries corresponding to the *oc* of a node as is shown on the right of figure. For example at *RL*[10], we would have the label 'c', scope 10 and level 2. If extending T_{k-1} from extension point node 'b' (*oc*:4) with node 'e' (*oc*:8) then *l(5,8)* will append one backtrack symbol (*pl(4,5)* = 1) and the label 'e' to $\varphi(T_{k-1})$ together with the distance to the root node obtained from *RL* (i.e. level of node 'e' - level of root node 'a', i.e. 3 – 0 = 3) . The new encoding $\varphi(T_k)$ becomes 'a0 b1 /1 b1 c2 /1 e3', and $\Psi(T_k):[0,4,8]$ (i.e. inserting 8 in entry (j+1) – b = (3+1) – 1 = 3 of $\Psi(T_{k-1})$).

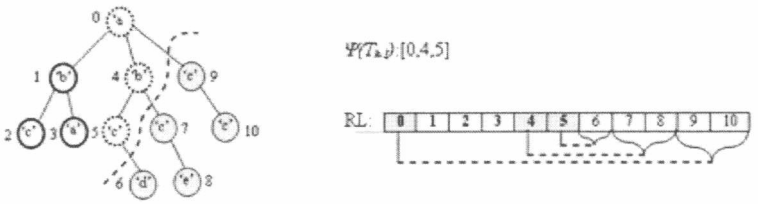

Fig. 5. TMG enumeration: extending (k-1)-subtree T_{k-1} (where $\varphi(T_{k-1})$:'a b / b c' occurs at position (0,1,4,5)) with nodes at positions 6, 7, 8, 9, and 10

In the case of unordered subtrees, the right-most-node may not always correspond to the last node (tail position) in the encoding as it does for the ordered subtree case. We refer to this case as *non-tail expansion*. A notion of pivot position ς is used to

denote the position in the subtree encoding that corresponds to the right-most-node. Each RMP-OC of a subtree will store an integer indicating the pivot position ς in the encoding for that particular occurrence of the subtree. Hence, for a *non-tail expansion* of a subtree T_{k-1}, if we are appending a new node with label l and OC t, rather than appending the backtrack symbols (if any) and l to the last node in $\varphi(T_{k-1})$, it will be appended to the pivot position ς by the function $l(\varsigma, t)$, in order to obtain $\varphi(T_k)$. Please note that if there are b backtrack symbols to be appended with l and there were already some backtrack symbols after the pivot position ς in $\varphi(T_{k-1})$, then l will be appended after the b^{th} backtrack symbol. Furthermore, an additional backtrack symbol will be appended after the position in the encoding where l has been appended. To illustrate this please consider the subtree *st3* from tree T in Fig. 1, with *oc*:01587, $\Psi(st3)$:*[0,5,8] and* $\varphi(st3)$:'b0 a1 /1 c1 d3 /1 e3' As can be seen the right-most node does not correspond to the last node 'e' in the encoding with oc:7, but rather to node 'd' with *oc*:8. Therefore, if we are extending *st3* from extension point node 'c' (*oc*:5) with node 'e' (*oc*:9) then *l(5,9)* will append the label 'e' to $\varphi(st3)$ at pivot position ς and add '/1' after 'd' (from *pl(5,8)*. The new encoding becomes 'b0 a1 /1 c1 d3 /1 e2 /1 e3'.]. Whenever a new candidate k-subtree is generated, full (k-1) pruning [2,3,9] is performed where a k-subtree is pruned if at least one of its (k-1)-subtree is infrequent. The whole process of TMG candidate enumeration is repeated until all frequent k-subtrees are enumerated.

4 Experimental Results

The experiments were run on Intel Xeon E5345 at 2.33 GHz with 8 cores, 8 GB RAM and 4MB Cashe Open SUSE 10.2. The purpose of the first experiment is to test the scalability of the proposed u3Razor algorithm with respect to the increasing number of transactions present in a database. An artificial database was created, where the size of the transactions for each test was varied from 100,000, 500,000 to 1 million with minimum support 50, 250, and 500, respectively. Occurrence match support definition was used and the result displayed in Fig. 6, shows that the time to complete the task approximately scales linearly with the increase in transaction size.

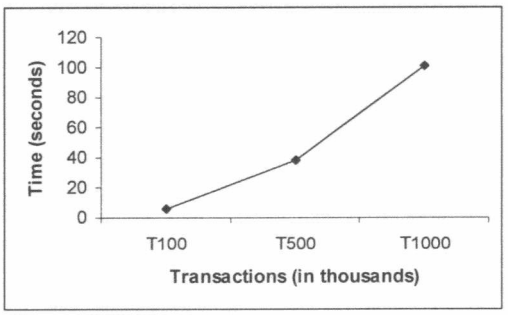

Fig. 6. scalability – time performance / number of transactions

The second experiment was performed to examine compare the number of frequent subtrees detected between the proposed algorithm and the UNI3 [12] and U3 [13] algorithms for mining of unordered induced and embedded subtrees, respectively. Real world CSLogs data set [9] consisting of 32421 transactions was used, and the transactional support definition was used. The number of frequent subtrees detected by the u3Razor and UNI3 algorithms is equal for support thresholds of 1000, 800 and 600 (Figure 7(a)). At these supports, the U3 algorithm detects additional subtrees as frequent, which implies that those additional embedded subtrees occur with a different distance among the nodes in the original tree. Otherwise, a number of them would have been detected by the u3Razor algorithm, where the distances have to be the same. It should be noted here that there may be some embedded subtrees that occur with the same distance among the nodes but the number of occurrences of such subtrees is not sufficient to be considered as frequent by the u3Razor algorithm for the given support. Since there are no embedded levels among the nodes in induced subtrees (i.e. the distance between all the nodes is equal to 1), both u3Razor and UNI3 detect the same frequent subtrees in this scenario. For lower support thresholds, more subtrees will be considered as frequent. The difference between the number detected by u3Razor and UNI3 in Fig. 7(a) indicates that there are some sufficient occurrences of embedded subtrees where the distance among the nodes is different in the original tree. There are 4 such embedded subtrees for s400 and 39 for s200. For some applications it may be of interest to the user to analyze such patterns to reveal some specific dataset characteristics. If the difference in the level is caused by same information being stored differently then they can be considered as valid patterns, while if the difference is due to the items with same labels being used in different contexts then they should be considered invalid. They are valid with respect to the support threshold. Fig. 7(b) shows the time taken by the algorithms for completing this task. For most support thresholds, the u3Razor algorithm takes slightly longer and this is due to the fact that the level information needs to be stored for all the nodes of the enumerated subtrees. At s200, the U3 algorithm takes the longest which is explained by the additional 54 subtrees that it considers as frequent in comparison with u3Razor (see Fig. 7(a)).

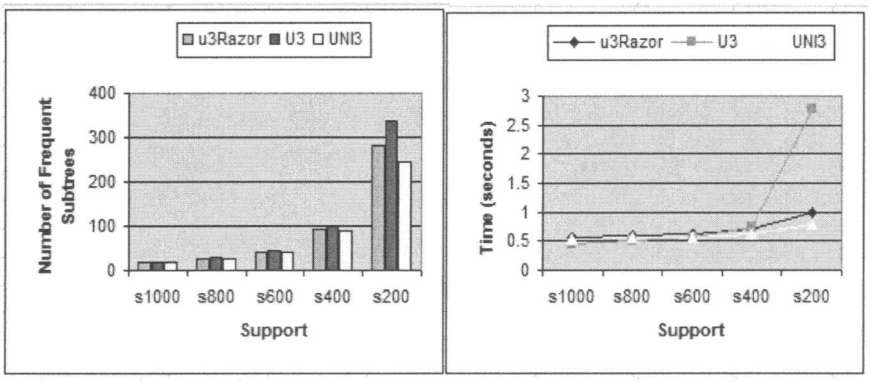

Fig. 7. (a) Number of frequent subtrees **(b)** time taken

5 Conclusions

In this paper we have discussed the motivation and some important applications for mining of unordered distance-constrained embedded subtrees. The first algorithm that solves this problem was presented as the extension to our general TMG candidate subtree enumeration framework. A number of experiments were performed using both synthetic and real-world datasets. The comparison of the results with the algorithms mining traditional subtree types, indicate the potential for more specific data analysis from tree-structured documents by considering the distance between the nodes.

References

1. Chi, Y., Yirong, Y. and Muntz, R. R. Canonical Forms for Labeled Trees and Their Applications in Frequent Subtree Mining, *Knowledge and Information Systems*, 2004.
2. Tan, H., Dillon, T.S., Hadzic, F., Feng, L., Chang, E. IMB3-Miner: Mining Induced/Embedded Subtrees by Constraining the Level of Embedding, *PAKDD'06,* Singapore, 2006.
3. Tan, H., Hadzic, F., Dillon, T.S., Feng, L., Chang, E., 2007, 'Tree Model Guided Candidate Generation for Mining Frequent Subtrees from XML', to appear in *ACM Transactions on Knowledge Discovery from Data (TKDD).*
4. Hadzic, F., Dillon, T. S., Sidhu, A., Chang, E, and Tan, H. Mining Substructures in Protein Data", invited paper in *IEEE ICDM DMB Workshop*, 18-22 December, Hong Kong, 2006.
5. Shasha, D., Wang, J.T.L. and Zhang, S. Unordered Tree Mining with Applications to Phylogeny, *20th International Conference on Data Engineering*, 2004.
6. Chi, Y., Yang, Y., and Muntz, R. R. HybridTreeMiner: An efficient algorihtm for mining frequent rooted trees and free trees using canonical forms. In *Proc. of the 16th Int'l Conf. on Scientific and Statistical Database Management*, Santorini Island, Greece, 2004.
7. Nijssen, S. and Kok, J. N. Efficient discovery of frequent unordered trees. *Int'l Workshop on Mining Graphs, Trees, and Sequences (MGTS-2003)*, Dubrovnik, Croatia, 2003.
8. Asai, T., Arimura, H., Uno, T. and Nakano, S. Discovering Frequent Substructures in Large Unordered Trees. *6th Int'l Conf. on Discovery Science*, 2003.
9. Zaki, M. J. Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications. In *IEEE Transactions on Knowledge and Data Engineering*, 17, 8, pp. 1021-1035, 2005.
10. Zaki, M. J. Efficiently Mining Frequent Embedded Unordered Trees. *Fundamenta Informaticae 65*, IOS Press, pp. 1-20, 2005.
11. Tan, H., Dillon, T.S., Hadzic, F , and Chang, E. Razor: mining distance-constrained embedded subtrees, *Workshop on Ontology Mining and Knowledge Discovery from Semistructured documents (MSD 2006)*, in conjunction with *ICDM'06*, Hong Kong, 28-22 December, 2006.
12. Hadzic, F., Tan, H., and Dillon, T.S. UNI3: efficient algorithm for mining unordered induced subtrees using TMG candidate generation. *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007)*, Honolulu, Hawaii, April 1-5, 2007.
13. Hadzic, F., Tan, H., Dillon, T.S. U3 – Mining Unordered Embedded Subtrees Using TMG Candidate Generation. Submitted to *ECML PKDD*, 15-19 Sept., Antwerp, Belgium, 2008.
14. Tan, H., Hadzic, F., Dillon, T.S., Chang, E. State of the art of data mining of tree structured information, *International CSSE Journal*, vol. 23, no 2, March, 2008.
15. Valentine, G. *Algorithms on Trees and Graphs*, Springer-Verlag, Berlin, 2002.
16. Access Excellence @ the national health museum, Understanding gene testing: What does a predictive gene tell you. http://www.accessexcellence.org/AE/AEPC/NIH/gene14.html.