RAZOR: mining distance-constrained embedded subtrees

Henry Tan
Faculty of IT, UTS
City Campus Broadway 1
Broadway, NSW 2000, Australia
+61-2-95144469
henryws@it.uts.edu.au

Tharam S. Dillon
Faculty of IT, UTS
City Campus Broadway 1
Broadway, NSW 2000, Australia
+61-2-95141800
tharam@it.uts.edu.au

Fedja Hadzic
Faculty of IT, UTS
City Campus Broadway 1
Broadway, NSW 2000, Australia
+61-2-95144469
fhadzic@it.uts.edu.au

Elizabeth Chang
School of IT, Curtin University
GPO Box U1987
Perth, WA 6845, Australia
+61-8-92661235
elizabeth.chang@cbs.curtin.edu.au

ABSTRACT

Due to their capability for expressing semantics and relationships among data objects, semi-structured documents have become a common way of representing domain knowledge. Comparing structures among semi-structured data objects often reveals useful information and hence tree and graph mining have become useful for applications in areas such as Bioinformatics, Ontology mining, Web mining, XML mining, schema matching etc. The type of sub-structures to be mined differs according to the needs of the applications. An important problem arises in the area of ontology matching, namely that of sub-structure matching as well as concept matching. This sub-structure matching can often help filter out 'false matches' in simple concept matching. This problem of sub-structure matching creates the need for distance constrained subtree matching. Our work is focused on the task of mining frequent subtrees from a database of rooted ordered labeled subtrees. Previously we have developed an efficient algorithm, *MB3* **[23]**, for mining frequent embedded subtrees from a database of rooted labeled and ordered subtrees. The efficiency comes from the utilization of a novel Embedding List representation for Tree Model Guided (*TMG*) candidate generation. As an extension the *IMB3* **[24]** algorithm introduces the Level of Embedding constraint. In this study we extend our past work by developing an algorithm, *Razor*, for mining embedded subtrees where the distance of nodes relative to the root of the subtree needs to be considered. This notion of distance constrained embedded tree mining will have important applications in web information systems and conceptual model analysis. Domains representing their knowledge in a tree structured form may require this additional distance information as it commonly indicates the amount of specific knowledge stored about a particular concept within the hierarchy. The structure based approaches for schema matching commonly take the distance among the concept nodes within a sub-structure into account when evaluating the concept similarity across different schemas. We present an encoding strategy to efficiently enumerate candidate subtrees taking the distance of nodes relative to the root of the subtree into account. This allows us to preserve the *TMG* approach and obtain an efficient algorithm for yet another subset of the tree mining problem. The algorithm is applied to both synthetic and real-world datasets, and the experimental results demonstrate the correctness and effectiveness of the proposed technique.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – Information filtering; H.3.4 [**Information Storage and Retrieval**]: Systems and Software – Performance evaluation (efficiency and effectiveness); I.7.0 [**Document and Text Processing**]: General

General Terms

Algorithms, Experimentation

Keywords

association mining, frequent subtree mining, mining with constraints, embedded subtree, structure matching

1. INTRODUCTION

Research in both theory and applications of data mining is expanding driven by a need to consider more complex structures, relationships and semantics expressed in the data. Association rule mining is a popular data mining technique used for discovering associations between data objects in a database. The majority of research in the area went towards the development of algorithms capable of efficiently extracting association rules from a relational database. Due to the increasing use of semi-structured information representation additional research focus is in the development of efficient tree mining algorithms. Tree mining has gained a considerable amount of interest in areas such as Bioinformatics, XML mining, Web mining, etc. In general, most of the formally represented information in these domains is a tree structured form and XML is commonly used. Feng et. al. **[7]**

have proposed an XML-enabled association rule framework. It extends the notion of associated items to XML fragments to present associations among trees. Association mining consists of frequent pattern discovery and rule construction out of which the former is considered to be a more complex task, and is the focus of our research in the area of tree mining. The two known types of subtrees are induced and embedded. An induced subtree is a subtree where the parent-child relationships must be the same to those in the original tree. In addition to this, an embedded subtree allows a parent in the subtree to be an ancestor in the original tree and hence the information about ancestor-descendant relationships is kept. Examples of induced and embedded subtrees are given in figure 2. Generally the problem of frequent subtree mining can be stated as: given a tree database T_{db} and minimum support threshold (σ), find all subtrees that occur at least σ times in T_{db}.

Most of developed algorithms for mining embedded subtrees adapt the join approach to generate candidate subtrees. While the join approach is efficient for relational data, when applied to tree structured data many candidates are generated that do not conform to the structural aspects of the tree database at hand. This hinders the performance as invalid candidates are generated and then pruned after determining that they do not exist in the tree database. The problem has motivated us to take a different approach to candidate subtree generation which ensures that only valid candidates are generated. We refer to this candidate generation strategy as Tree Model Guided (TMG) **[22, 23, 24, 25]**. This non-redundant systematic enumeration model ensures only valid candidates are generated which conform to the actual tree structure of the data. An example of a tree model would be the structural aspects of a document in XML schema, and a valid candidate would conform to this. In general, the TMG would be applicable to any area with structural models with clearly defined semantics that have tree like structures. In **[23]** we have introduced a novel and unique Embedding List (EL) representation suitable for describing embedded subtrees. The integration of the EL representation enabled the TMG candidate generation to be done in an efficient manner which was demonstrated in our experimental evaluations of the algorithm. We also developed a mathematical formula that can be used to estimate the worst case complexity of the TMG candidate generation. The formula indicates the number of candidate subtrees that will be generated at each step and it shows that mining embedding relationships can be very costly for complex trees. Using the formula one could predict infeasible cases in which the number of candidates to be generated is too large. In such situations one would be forced to constrain the mining process in some way so that at least some patterns could be discovered. This motivated us to develop a strategy to tackle the complexity of mining embedded subtrees by introducing Level of Embedding constraint **[24]**. Thus, when it is too costly to mine all frequent embedded subtrees, one can decrease the level of embedding constraint gradually up to 1, from which all the obtained frequent subtrees are induced subtrees.

In this study we extend our past work by developing an algorithm for mining embedded subtrees when the distances of the nodes relative to the root of the subtree need to be considered. We felt that the traditional embedded subtree definition may allow too much freedom with respect to the embedding of the subtrees extracted. The embedded subtrees extracted using the traditional definition are incapable of being further distinguished based upon the node distance within that subtree. For certain applications the distance between the nodes in a hierarchical structure could be considered important and two embedded subtrees with different distance relationships among the nodes need to be considered as separate entities. The distances of nodes relative to the root (node depth) of a particular subtree will need to be stored and used as an additional equality criterion for grouping the enumerated candidate subtrees. This notion of distance constrained embedded tree mining will have important applications in web information systems and conceptual model analysis. Knowledge merging is another area where the distances between the nodes within an embedded subtrees may need to be considered. Inside a concept-hierarchy the distances between the nodes indicate the amount of specific knowledge that is known about a particular concept, or is needed for the accurate classification of that concept **[21]**. The structure based approaches for schema matching commonly take the distance among the nodes within a sub-structure into account when evaluating the concept similarity across different schemas **[6, 13, 19]**. Hence our aim in this paper is to obtain an efficient algorithm that will extract all embedded subtrees with the additional node distance information.

In order to maintain the efficient use of EL for TMG candidate generation, the major extension requirement is for an appropriate candidate encoding scheme to distinguish subtrees based upon structure and the node distances within the structure. The structural aspects need to be preserved as well as extra distance information needs to be stored. We present an encoding strategy to efficiently enumerate candidate subtrees taking the distances of nodes relative to the root of the subtree into account. This allows us to preserve the TMG approach and obtain an efficient algorithm, Razor, for yet another subset of the tree mining problem. We apply the algorithm to both synthetic and real-world datasets, and the experimental results demonstrate the correctness and effectiveness of the proposed technique.

The rest of the paper is organized as follows. A motivating example is provided in section 2 together with a quick overview of the ontology matching problem. The problem decomposition is given in section 3 and the related works are discussed in section 4. The Razor algorithm is described in section 5. The experiments on real world and synthetic data are presented in section 6, and section 7 concludes the paper.

2. MOTIVATING EXAMPLE

The purpose of this section is to present an example that demonstrates the usefulness of adding the distance constraint to embedded subtree mining. As mentioned in the introduction there are a few applications where mining of distance constrained embedded subtrees would have important implications. The problem considered here is concerned with semantic matching of concepts that come from heterogeneous data sources. Semantic matching is particularly important in the area of Ontology learning and matching. We start this section by giving a brief overview of the Ontology matching problem.

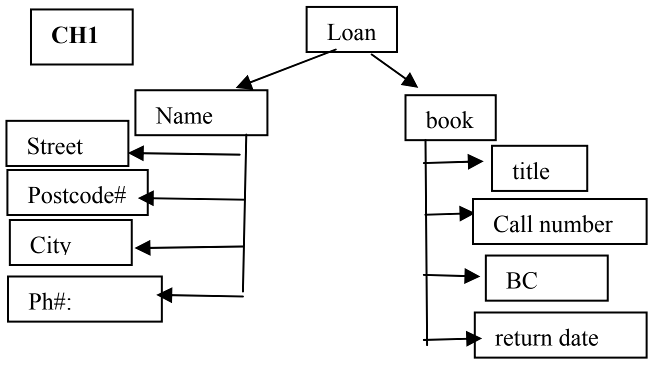
Ontology in AI is defined as a formal, explicit specification of a shared conceptualization. Formal corresponds to the fact that the ontology should be machine readable, explicit means that the concepts and their constraints should be explicitly defined, and conceptualization refers to the description of concepts and their relationships that occur in a particular domain **[11, 8].** The main differences among Ontologies occur in: vocabularies, design principles, knowledge representation, level of detail and the ontology commitment **[18, 12]**. These differences make the ontology matching a challenging task and to manually perform the task would be too time consuming and error-prone. Automatic detection of semantic matches among ontology concepts has therefore become the initial and most challenging stage in most of ontology merging and alignment tasks **[16, 10]**.

This problem is analogous to schema matching in databases. The goal is to find semantically correct matches between the schema concepts. The schema matching techniques can be distinguished as involving schema, instance, constraint, linguistic, element and structure based approaches **[13]**. When concept naming differs among ontologies the syntax based approaches such as linguistic matchers have difficulties and a semantic approach is desired.

Semantic matching takes the schema information as well as the positions of nodes in the conceptual models (graph or tree) into account. The MOMIS approach **[3]** is concerned with integration and querying of heterogeneous information sources containing semistructured and structured data. Semantic matching is performed based on the conceptual schemas of the information sources. The approach also makes use of a common thesaurus to identify semantically related information. The TreeMatch algorithm **[13]** computes the similarity of contexts in which the two concepts occur in the two schemas. It utilizes schema information and the representative tree structure. Similarity flooding algorithm **[14]** produces a similarity mapping between the concepts of two graph structures. A string match operator is used to obtain the initial matching nodes which propagate the similarity to their adjacent nodes. The Anchor-PROMPT algorithm **[17]** takes as input a set of similar terms (anchors) and determines sets of other related terms by analyzing the paths in the subgraph limited by the anchor points. It is based on the intuition that if two pairs of concepts in the source ontologies are similar and there are paths connecting those two concepts, then the concepts in those paths are often similar as well. Giunchiglia and Shvaiko **[9]** perform element and structure-level semantic matching among the elements of two graphs. Initially, the schema information is used to produce semantic relations among all the concepts and the graph structure is then traversed to construct the propositional formulas among concepts (equality, overlap, mismatch, more general/specific). For a more detailed overview and comparison of some existing approaches to automatic schema matching, please refer to **[19].**

To illustrate the usefulness of detecting distance-constrained embedded subtrees please consider figure 1. The two conceptual hierarchies (CH1, CH2) represent a borrowing record from two different library based applications. Let us assume that the concepts located at the top of the hierarchies are both known to correspond to a borrowing record. As discussed in the previous paragraph common approach would be to investigate the sub-structures containing the concepts already found to be similar in order to update the similarity of the neighboring concepts. One approach would be to detect the longest subtree whose structure matches both of the representations and then perform the similarity update among concepts within that structure. If embedded subtrees are mined the longest matching subtree would be of size 11 which corresponds to the whole structure of CH1. The CH2 is a more specific model and there are quite a few embedded subtrees in CH2 that match the whole structure of CH1. However, only one of those embedded subtrees is a true match. If similarity update was performed on all these matching subtrees there would be lots of incorrect updates and at the end it would be ambiguous to determine which were the true matches. Hence in this case extracting the largest matching embedded subtree could affect the similarity update in an undesirable way since updates would not distinguish among the subtrees where the distance among the nodes is different. This information is needed for a more exact structure matching where the level of granularity among concepts is the same in two hierarchies. At this stage, where labels are unknown, we consider a subtree an exact match of another subtree only if the structure and the distance among the nodes is the same in both subtrees. In other words, both sets of concept nodes need to be positioned in exactly the same way in both subtrees. The embedded subtree definition relaxes this constraint and we therefore felt that an additional distance constraint among nodes is required to obtain the exact match among subtrees.

Considering the concept hierarchies from figure 1 again, if we mine distance-constrained embedded subtrees, the largest matching subtree has 7 nodes. This subtree corresponds to the right hand side of the CH1 plus the node in level 1. This subtree is the largest exact match between CH1 and CH2, and the similarity update among the neighboring nodes in this subtree can be performed with high confidence. The unmatched subtrees of the structures are known to differ in the amount of specificity and the node distance information could prove to be useful for additional reasoning over concept similarity. Another option at this stage is to start mining the embedded subtrees from the remaining unmatched structure. This would relax the distance constraint and similar structures which differ in concept granularity could be detected.



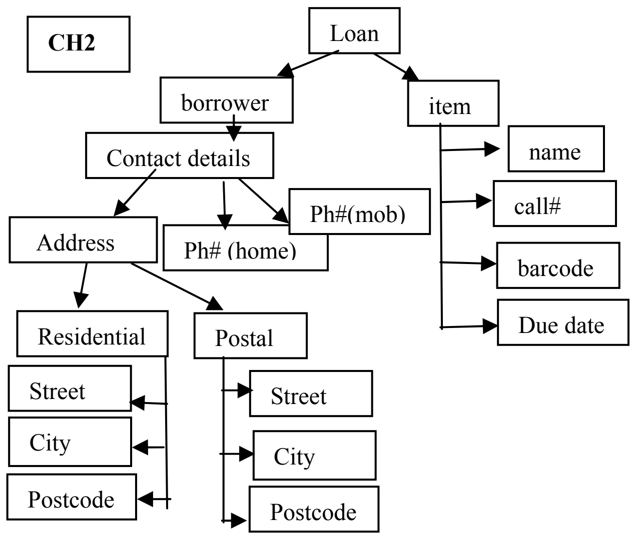


Figure 1. Libraries borrowing record schemas

In the context of determining the semantic similarity among concepts mining distance-constrained embedded subtrees will provide a more strict structural matching approach as concept nodes must occur at the same positions among extracted subtree patterns. It is more strict in the sense that the extracted embedded subtrees where one concept is known to be the same, ie candidates for comparison, are much higher in number. Each concept related embedded subtree is now refined in multiple distance constrained embedded subtrees which keep the information about the node positions in the original tree. This refines the comparison since extra reasoning can take place, taking into account the concept node positions among extracted subtrees. It is important to note here that we are not claiming that the mining of distance constrained embedded subtrees should replace the mining of embedded subtrees for the purpose of semantic matching. Embedded subtrees without the distance constraint are still important as the amount of concept granularity can differ among knowledge representations and we need to relax the node distance constraint in order to investigate such relationships. Consequently, one could start the structural matching process by first mining distance constrained embedded subtrees. This would detect initial exact matches where the concept granularity is the same among knowledge representations. It also indicates the point in the structure matching process where differences in the concept granularity occur and extra care has to be taken with similarity update. The distance constraint could then be relaxed (i.e extract embedded subtrees) in order to detect other pairs of concept related subtrees which are similar in structurebut the concept granularity differs among knowledge representations. This would refine the concept matching process and help avoid initial bad matches which could affect the rest of the knowledge (ontology) matching process. Hence the whole process could be performed in a more controlled manner where the node positions in the representative structure are taken into account.

3. PROBLEM DEFINITIONS

A tree can be denoted as T(r,V,L,E), where (1) r $\in$ V is the root node; (2) V is the set of vertices or nodes; (3) L is the set of labels of vertices, for any vertex v$\in$ V, L(v) is the label of v; and (4) E is the set of edges in the tree. Each node v in the tree has only one parent, parent(v), which is defined as the predecessor of node v. A node v can have one or more children, children(v), which are defined as its successors. If p is an ancestor of q and q is a descendant of p, then there exists a path from p to q.. A path from vertex v_i to v_j, is defined as a finite sequence of edges that connects v_i to v_j. The length of a path p is the number of edges in p. When referring to the distance between the two nodes we simply refer to the length of the path connecting those two nodes. Height of a node is the distance to its furthest leaf, whereas the depth of a node is its distance to the root. The number of children of a node is commonly termed as fan-out/degree of the node, degree(v). A node without any children is a leaf node; otherwise, it is an internal node. If for each internal node, all the children are ordered, then the tree is an ordered tree. The rightmost path of T is defined as the path connecting the rightmost leaf with the root node. The size of a tree is determined by the number of nodes in the tree. All trees considered in this paper are rooted ordered labeled.

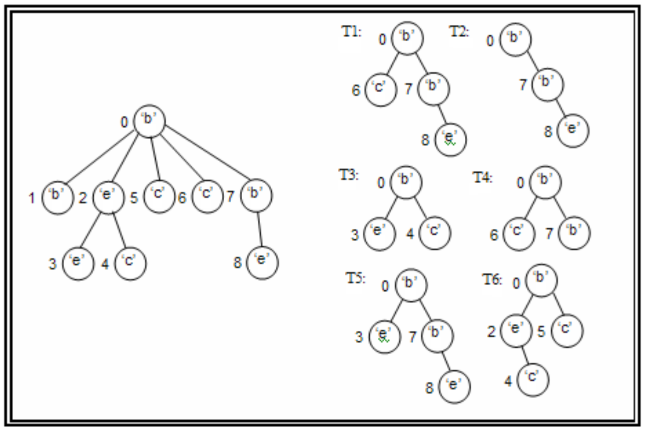


Figure 2. Example of induced subtrees (*T1, T2, T4, T6*) and embedded subtrees (*T3, T5*) of tree T (note that induced subtrees are also embedded)

Induced Subtree. A tree T'(r', V', L', E') is an ordered induced subtree of a tree T (r, V, L, E) iff (1) V'$\subseteq$V, (2) E'$\subseteq$E, (3) L'$\subseteq$L and L'(v)=L(v), (4) $\forall$v'$\in$V', $\forall$v$\in$V and v' is not the root node, and v' has a parent in T, then parent(v')=parent(v), (5) the left-to-right ordering among the siblings in T' is preserved. An induced subtree T' of T can be obtained by repeatedly removing leaf nodes or the root node if their removal doesn't create a forest in T.

Embedded Subtree. A tree T'(r', V', L', E') is an ordered embedded subtree of a tree T(r, V, L, E) if and only if it satisfies properties 1, 2, 3 and 5 of an induced subtree and it generalizes property (4) such that $\forall$v'$\in$V', $\forall$v$\in$V and v' is not the root node, the sets ancestor(v') and ancestor (v) form a non-empty intersection. Examples of induced and embedded subtrees are given in Figure 2.

Level of Embedding (Φ). If T'(r', V', L', E') is an embedded subtree of T, and there is a path between two nodes p and q, the *level of embedding (Φ)* is defined as the length of the shortest path between p and q, where p$\in$V' and q$\in$V', and p and q form an ancestor-descendant relationship. In other words, given T andΦ, then any embedded subtree to be generated will have the length of the shortest path in T between any two ancestor-descendant nodes from T' equal or less than Φ. In this regard, we could define induced subtree T as an embedded subtree where the maximum *level of embedding* that can occur in T is equal to 1, since the *Level of Embedding* of two nodes that form a parent-child relationship equals to 1.

Distance-Constrained Embedded Subtree. A tree T'(r', V', L', E') is an ordered distance-constrained embedded subtree of a tree T(r, V, L, E) if it satisfies all the properties of an embedded subtrees (above), and $\forall v' \in V'$ there is an integer stored indicating the level of embedding (Φ) in tree T between v' and the root node of T'.

Adding distance constraint. In this paper we are concerned with mining embedded subtrees where the distance between the nodes in the original tree database needs to be considered. The main difference is that the distance between the nodes is used as an additional equality criterion to group the enumerated candidates. To illustrate the difference that this additional distance constraint will impose on the task of mining embedded subtrees, consider the example tree shown in Figure 3. If the traditional mining technique for embedded subtrees is used, a subtree 'A C' by occurrence match support definition would have support equal to 8. On the other hand if the distance equality constraint is added we would need to distinguish this candidate into three candidates depending on the varying distance between the nodes. Hence the three 'A C' subtree candidates would have varying distances of 1, 2 and 3 and the support of 2, 4 and 2 respectively.

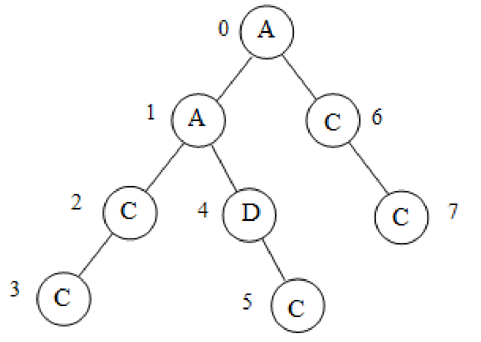


Figure 3. Example tree with labeled nodes ordered in pre-order traversal

For subtrees with more nodes the stored distance for each node will correspond to its distance to the root of that particular subtree. It can be seen that this additional constraint will add extra complexity to the traditional frequent subtree mining problem. More candidate subtrees will need to be enumerated and counted during the task.

Transaction based vs occurrence match support. We say that an embedded subtree t is supported by transaction $k \subseteq K$ in database of tree T_{db} as $t \prec k$. If there are L occurrences of t in k, a function g(t,k) denotes the number of occurrences of t in transaction k. For transaction based support, $t \prec k=1$ when there exists at least one occurrence of t in k, i.e. $g(t,k) \geq 1$. For occurrence match support, $t \prec k$ corresponds to the number of all occurrences of t in k, $t \prec k=g(t,k)$. Suppose that there are N transactions k_1 to k_N of tree in T_{db}, the support of embedded subtree t in T_{db} is defined as:

$$\sum_{i=1}^{N} t \prec k_i \qquad (1)$$

Transaction based support has been used in a number of works **[4, 27, 30]**, whereas occurrence match support has been less utilized and discussed. Occurrence match support takes repetition of items in a transaction into account whilst transaction based support only checks for existence of items in a transaction. There has not been any general consensus which support definition is used for which application. However, it is intuitive to say that whenever repetition of items in each transaction is to be accounted and order is important, occurrence match support would be more applicable. Generally, transaction based support is very applicable for relational data. Our focus is on occurrence match support in this paper.

Mining frequent embedded subtrees. Let T_{db} be a tree database consisting of N transactions of trees, K_N. The task of frequent embedded subtree mining from T_{db} with given minimum support (σ), is to find all the candidate embedded subtrees that occur at least σ times in T_{db}. Based on the downward-closure lemma **[2]**, every sub-pattern of a frequent pattern is also frequent. In relational data, given a frequent itemset all its subsets are also frequent. However, when mining induced and embedded subtrees from a rooted ordered labeled database of trees, there can be frequent subtrees with one or more of its subsets infrequent. We refer to these types of frequent subtrees as pseudo-frequent subtrees **[23, 24, 25]**. Hence, in the case where there exists a frequent subtree 's' with one or more of its subtrees infrequent, then 's' also needs to be considered infrequent for the antimonotone property to hold. Tree structured data has a hierarchical structure where 1-to-many relationships can occur, as opposed to relational data where only 1-to-1 relationships exist between the items in each transaction. This multiplication between one node to its many children/ descendants makes the antimonotone property not hold for tree structured data. It should be noted that if transaction based support is used, pseudo-frequent subtrees will not be generated. When the repetition of items is reported only once per transaction the 1-to-many relationship between a node to its children is treated as set of items like in relational database. When using occurrence match support, a full (k-1) pruning should be performed at each iteration when generating k-subtree from a (k-1)-subtree **[23, 25, 30]** so that no pseudo-frequent subtrees would be generated.

4. RELATED WORKS

There are different types of trees and one can distinguish between unrooted unordered trees (free trees) **[5, 20]**, rooted unordered trees **[15]**, and rooted ordered trees **[1, 22, 23, 24]** These three types have increasing topological structure **[4]** as one progresses from the first to the third. Many algorithms have been developed that mine different types of tree patterns. FreeTreeMiner **[20]** extracts free trees in a graph database. PathJoin **[28]**, uFreqt **[15]**, and HybridTreeMiner **[5]** mine induced, unordered trees. In data mining community, a string-like representation of a tree structure is becoming very popular **[1, 4, 30, 27]**. Each item in the string can be accessed in O(1) time and the representation itself has been reported to be space efficient and easy for manipulation **[4, 22, 30]**. When using breadth first string-like representation, the scope of a node denotes the position of its right-most descendant. This string-like representation preserves Thus, the hierarchical relationships from the original tree database are semantically preserved and the original tree structure can be reconstructed from the string-like representation. The two known enumeration strategies are enumeration by extension and join **[5]**. Recently, Zaki **[30]** adapted the join enumeration strategy for mining frequent embedded rooted ordered subtrees, and developed the efficient TreeMiner algorithm for discovering frequent embedded subtrees in a forest using a data structure called the vertical scope-list. An idea of utilizing a tree model for efficient enumeration appeared in **[29]**. The approach uses the XML schema to guide the candidate generation so that all candidates generated are valid because they conform to the schema. We further generalized this concept of schema guided into tree model guided candidate generation for mining embedded rooted ordered labeled subtrees **[22, 23]**. We refer to such an enumeration method as Tree Model Guided (TMG). TMG can be applied to any data with clearly defined semantics that have tree like structures. It ensures that only valid candidates which conform to the actual tree structure of the data are generated. The enumeration strategy used by TMG is a specialization of the right most path extension approach **[1, 30, 22, 23].** However it is different from the one that is proposed in FREQT **[1]** because TMG enumerates embedded subtrees and FREQT enumerates only induced subtrees. The right most path extension method is reported to be complete and all valid candidates are enumerated at most once (non-redundant) **[1, 22, 23]**. This is in contrast to the incomplete method TreeFinder **[26]** that uses an Inductive Logic Programming approach to mine unordered, embedded subtrees. TreeFinder can miss many frequent subtrees. The extension approach utilized in the TMG generates fewer candidates as opposed to the join approach **[30]**. Independently, XSpanner **[27]** extends the Pattern-Growth concept into tree structured data and its enumeration model also generates only valid candidates. XSpanner only reports distinct embedded subtrees similar to the recently published TreeMinerD **[30]**. TreeMinerD is different to TreeMiner in the sense that TreeMiner reports all embedding subtrees.

5. RAZOR ALGORITHM

The Razor algorithm mines frequent embedded subtrees taking the distance between the ancestor-descendant nodes in the given tree database into account. The necessary amendments for incorporating the additional distance constraint occur in the way candidate subtrees are enumerated during candidate enumeration and (k-1)-subtree generation phase. We have utilized the TMG **[22, 23, 25]** candidate generation approach for an optimal, non-redundant candidate subtree enumeration. The previously introduced embedding list representation **[23]**, is used for an efficient implementation of the TMG candidate generation approach. Following, is a detailed description of the algorithm with the required adjustments.

Database scanning. As the first step in the process, a tree database, T_{db}, is scanned in order to generate a global sequence D in memory, which is referred to as a *dictionary*. The dictionary stores each node from the T_{db} following the pre-order traversal indexing. The node information stored consists of position, label, right-most descendant position (scope), depth and parent position of that particular node in T_{db}. Thus each dictionary item is defined as a tuple of position (pos), label (l), scope (s), depth (d), parent (p), {pos, l, s, p}. An item at index position i in the dictionary is referred to as *dictionary[i]*. During the construction of the dictionary the complete set of frequent 1-subtrees, F_1, is enumerated. Once the dictionary is constructed, no further database scanning is required.

String encoding (φ). We utilize a slight modification of the pre-ordering string encoding (φ) **[22, 23, 30]**, in order to store the additional distance information for each node of the encoded subtree. 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 depths stored in the dictionary, where the root of the subtree is assigned the depth of 0 and all other nodes are assigned the difference between their depth and the original depth of the new subtree root. Hence the additional information corresponds to the depths of nodes within the newly encoded subtree. 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 for easier string manipulation due to uniform block size. Thus, from figure 5, the main difference between IMB3 **[24]** and Razor lies in the *GetEncoding* computation. We denote encoding of a subtree T as φ(T). For each node in T (figure 1), its label is shown as a single-quoted symbol inside the circle whereas its pre-order position is shown as indexes at the left/right side of the circle. From figure 2, φ(T1):'b0 c1 /1 b1 e2 /2'; φ(T3):'b0 e1 /1 c1 /1'; φ(T6): 'b0 e1 c1 /2 c1 /1', etc. The backtrack symbol could be omitted after the last node, i.e. φ(T1):'b0 c1 /1 b1 e2'. The number next to each node label corresponds to the depth of that node. We refer to a group of subtrees with the same encoding L as candidate subtree C_L. A subtree with k number of nodes is denoted as k-subtree. Throughout the paper, the '+' operator is used to conceptualize an operation of appending two or more tree encodings. However, this operator should be contrasted with the conventional string append operator, as in the encoding used the backtrack symbols need to be computed accordingly.

Embedding List (EL) construction. In this section we describe the process of constructing the EL which allows for an efficient implementation of the TMG candidate enumeration. For each frequent internal node in F_1, a list is generated which stores its descendant nodes' hyperlinks **[27]** in pre-order traversal ordering such that the embedding relationships between nodes are preserved. The notion of hyperlinks of nodes refers here to the positions of nodes in the dictionary. For a given internal node at position i, such ordering reflects the enumeration sequence of generating 2-subtree candidates rooted at i (figure 4). Hereafter, we call this list as embedded list (EL). We use notation i-EL to refer to an embedded list of node at position i.; The position of an item in EL is referred to as slot. Thus, i-EL[n] refers to the (n-1)th item of in the list at slot n with zero-based indexing.. Whereas |i-EL| refers to the size of the embedded list rooted of node at position i. Figure 4 illustrates an example of the EL representation of tree T (figure 2). In fig 4, 0-EL for example refers to the list: 0:[1,2,3,4,5,6,7,8] and , 0-EL[0] = 1 and; 0-EL[4] = 5; 0-EL[6] = 7.

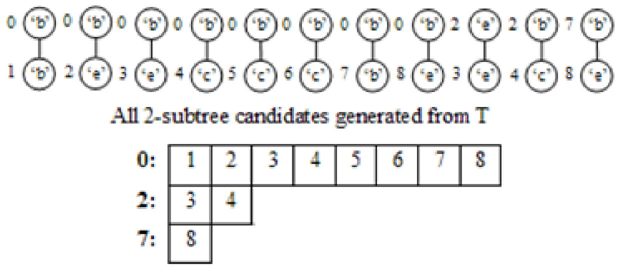


Figure 4. The EL representation of T in figure 1

Figure 4 illustrates an example of the EL representation of subtree T (figure 1). For each node in T, its label is shown as a single-quoted symbol inside the circle whereas its position is shown as indexes at the left side of the circle. Also, please note that each list stores node positions rather than labels.

Occurrence Coordinate (OC). A candidate subtree can occur at different positions in the database and OC is used to denote the node positions of that particular subtree so that it can be distinguished from other subtrees having the same encoding. When generating k-subtree candidates from (k-1)-subtree, we consider only frequent (k-1)-subtrees for extension. Each occurrence of k-subtree in T_{db} is encoded as occurrence coordinate r:[e_1,…e_{k-1}]; where r refers to the k-subtree root position in the dictionary D and e_1,…,e_{k-1} are refer to the indexes of slots in r-EL. Each e_i corresponds to node (i+1) in the k-subtree and in r; $e_1 < e_{k-1}$. We refer to e_{k-1} as tail slot. From figure 2 & 4, the OC of a 3-subtree (T2) with encoding 'b0 b1 e2' is encoded is encoded as 0:[6,7]; 4-subtrees T1 with encoding 'b0 c1 /1 b1 e2' are encoded as 0:[5,6,7], and so on. Each OC of a subtree describes an instance of that subtree in T_{db}, and hence each candidate subtree has at least one OC associated with it.

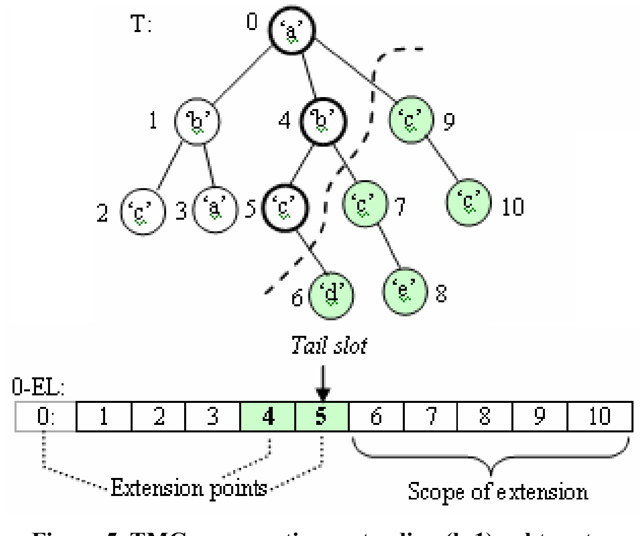


Figure 5. TMG enumeration: extending (k-1)-subtree t_{k-1} where $\varphi(t_{k-1})$:'a b / b c' occurs at position (0,1,4,5) with node at position 6, 7, 8, 9, and 10

TMG enumeration formulation. TMG is a specialization of right most path extension method which has been reported to be complete and non-redundant **[1, 22, 23]**. To enumerate all embedded k-subtrees from a (k-1)-subtree, TMG enumeration approach extends one node at the time to the right most path of (k-1)-subtree. We refer to each node in the right most path as extension point (figure 5). One important property of EL is that the positions of nodes are stored in pre-order manner. The scope of extension of a node denotes the range of nodes that can be appended to that node for the formation of new candidate subtrees. Hence, given a (k-1)-subtree with known tail slot, the subsequent slots in EL will form the scope of extension from i to j. All embedded k-subtree are generated by attaching a node at position i to j to the (k-1)-subtree. Suppose l(i) denotes a labeling function of node with at position dictionary coordinate i. Given frequent (k-1)-subtree t_{k-1} with $\varphi(t_{k-1})$:L, the root position r, tail position t, encoding L and occurrence coordinate r:[m,…,n], k-subtrees are generated by extending t_{k-1} with $j \in$ r-EL such that $t < j \leq |r\text{-EL}|-1$. Thus its occurrence coordinate becomes r:[m,…,n,j] and its encoding becomes L':L+l(i) where i=r-EL[j] and m<n<j. Similarly to the IMB3 algorithm **[24]**, Razor algorithm was implemented with the capability to restrict the level of embedding. This is achieved by performing a check at each extension point, of whether the level of embedding is less or equal to the specified Φ. Only when the level of embedding of a node at position j to its extension point is less than Φ, the extension is performed. From fig 5, suppose that Φ is set to 1, when we extend a subtree with OC 0:[0,3,4] with node at position 6, 7, and 9 (0:[5], 0:[6], 0:[8]), the level of embedding between nodes at position 6, 7, and 9 to their extension point equals to 1 (≤ Φ), and thus should not be pruned. However when it is extended with node at position 8 and 10 (0:[7], 0:[9]) the level of embedding between node at position 8 and 10 to their extension points is>2 (≥ Φ), and thus should be pruned.

k-1 full pruning. To ensure the absence of pseudo-frequent subtrees, full (k-1) pruning must be performed. The rationale of this has been discussed in **[23, 30]**. From this point onward we refer to full (k-1) pruning as full pruning. This implies that at most (k-1) numbers of (k-1)-subtrees need to be generated from the currently expanding k-subtrees. Exception is made whenever the Φ constraint is set to 1, i.e. mining induced subtree, we only need to generate l numbers of (k-1)-subtrees where l < (k-1) and l equal to the number of leaf nodes in k-subtrees. When the removal of root node of k-subtree doesn't generate a forest **[22, 23, 30]** then an additional (k-1)-subtree is generated by taking the root node off from the expanding k-subtree. The expanding k-subtree is pruned when at least one (k-1)-subtree is infrequent, otherwise it is added to the frequent k-subtree set. This ensures that the method generates no pseudo-frequent subtrees. Doing full pruning is quite time consuming and expensive. To accelerate full pruning, a caching technique is used by checking whether a candidate is already in the frequent k-subtree. If a (k-1)-subtree candidate is already in the frequent k-subtree set, it is known that all its (k-1)-subtrees are frequent, and hence only one comparison is made.

```
Inputs: $T_{db}$(Tree database),σ(min. support), Φ(max. level of embedding)
Outputs: $F_k$(Frequent subtrees), D(dictionary)
{D, $F_1$}: DatabaseScanning ($T_{db}$)
{EL, $F_2$}: ConstructEmbeddedList ($F_1$,D,Φ)
k=3

while( |$F_k$| ≥ 0 ){
  $F_k$ = GenerateCandidateSubtrees($F_{k-1}$,Φ)
  k = k+1
}
GenerateCandidateSubtrees($F_{k-1}$,Φ){
for each frequent k-subtree $t_{k-1}$ ∈ $F_{k-1}$
  $L_{k-1}$ = GetEncoding ($t_{k-1}$) (*)
  VOL-$t_{k-1}$ = GetVOL($t_{k-1}$)
  foreach occurrence coord. $oc_{k-1}$(r:[m,…n])∈VOL-$t_{k-1}$
     for (j = n+1 to |r-EL|-1 )
       if( EmbeddingLevel(j) ≤  Φ ) then{
          $oc_k$, $L_k$} = TMG-extend( $oc_{k-1}$,$L_{k-1}$,j )
            if( Contains($L_k$, $F_k$) )
              Insert( hashkey($L_k$),$oc_k$,$F_k$ )
            else
              If( k-1Pruning ($L_k$) == false)
                 Insert( hashkey($L_k$),$oc_k$,$F_k$ )
  return Fk
}
(*): The main difference between IMB3-Miner and Razor
```

Figure 6. Razor algorithm pseudo code

Vertical Occurrence List (VOL). To determine if a subtree is frequent, we count the occurrences of that subtree and check if it is greater or equal to the specified minimum support σ. We say that a subtree has a frequency n if there are n instances of subtrees with same encoding. Each occurrence of a subtree is stored as an occurrence coordinate, as previously described. Computing the frequency of a subtree can be easily determined from the size of the VOL. We use the notation VOL(L) to refer to the vertical occurrence list of a subtree with encoding L. Consequently, the frequency of a subtree with encoding L is denoted as |VOL(L)|. When transaction based support is used the occurrence of each subtree is grouped by its transaction id and the support count corresponds to the number of unique transactions in the VOL. The Razor algorithm as a whole can be described by the pseudo-code displayed in Figure 6.

6. EXPERIMENTAL RESULTS

In this section we present some of the tests performed on the Razor algorithm. Firstly we show the scalability of the approach followed by the comparisons with the MB3 **[23]** and IMB3 **[24]** algorithms on the grounds of the number of frequent subtrees generated for varying support and level of embedding thresholds. For the problem of mining frequent subtrees most of the time and space complexity comes from the candidate enumeration and counting phase. The distance-constrained subtrees are much larger in number then induced or embedded subtrees and in general an algorithm for this task would require more space and run-time. This makes our approach incompatible to the current tree mining algorithms and to our knowledge there is currently no algorithm that mines distance constrained embedded subtrees to allow for a compatible comparison. Note that the occurrence match support definition is used in all the experiments. The minimum support σ is denoted as (sxx), where xx is the minimum frequency. Experiments were run on 3Ghz (Intel-CPU), 2Gb RAM, Mandrake 10.2 Linux machine and compilation was performed using GNU g++ (3.4.3) with the –g and –O3 parameters.

6.1 Scalability test

The purpose of this experiment is to test whether the algorithm is well scalable 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 is varied from 100,000, 500,000 to 1 million with minimum support 50, 250, and 500 respectively. The result shows that the time to complete the operation scales linearly with the increase in transaction size.

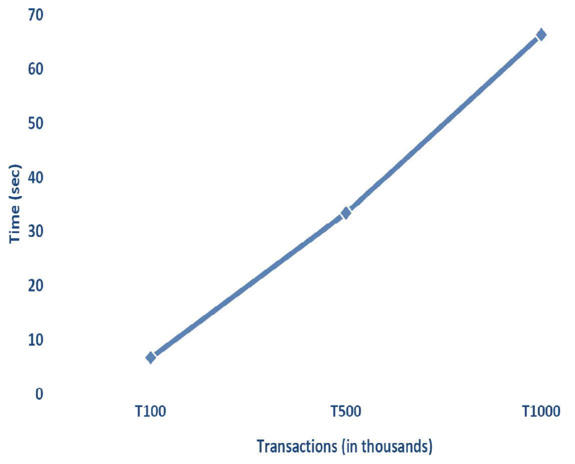


Figure 7. Scalability test - time performance / number of transactions

6.2 Frequent subtrees over different support

For this experiment we have used a reduction of the CSLogs data set previously used by Zaki **[30]** for testing the TreeMiner algorithm using the transactional support definition. If occurrence match support definition is used the transaction number would need to be reduced **[23]** so that the tested algorithms could return the set of all frequent subtrees. We have randomly reduced the number of transactions from 52,291 to 32.421. The comparison of the number of frequent subtrees generated among the MB3, IMB3 and Razor algorithms, for varying support thresholds is presented in figure 7.

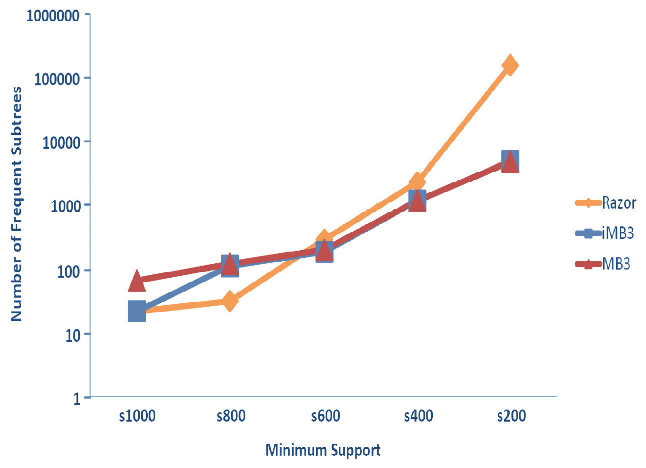


Figure 8. Number of frequent subtrees detected for varying support thresholds

IMB3 and Razor were set with the level of embedding (Φ) constraint equal to 5. Consequently, the number of detected frequent subtrees differs slightly between the IMB3 and MB3 algorithms. In figure 8 we can see that the number of frequent subtrees detected by the Razor algorithm increases significantly when the support is lowered. This is due to the additional distance relationship used to uniquely identify candidate subtrees. As shown in section 3 an embedded subtree may be split into multiple candidate subtrees when the distance is taken into account. As the support is lowered many more subtrees will be frequent and hence many more variations of those subtrees with respect to the distance among the nodes will also be frequent. This is the explanation for such a large jump in the number of frequent subtrees detected by the Razor algorithm, when support threshold is lowered sufficiently.

6.3 Varying the level of embedding

The purpose of this experiment is to compare the number of frequent subtrees detected by the algorithms when the level of embedding (Φ) is varied. We have artificially created a data set that characterizes a deep tree with the maximum depth equal to 17. It consists of 10,000 transactions with a total of 273,090 nodes. The support threshold was set to 100. The MB3 algorithm does not restrict the level of embedding and hence the number of frequent subtrees detected remains the same for all cases. As the largest level of embedding of the dataset tree is 17 the MB3 miner detects the largest number of frequent subtrees.

Variation in the number of frequent subtrees detected by the Razor and the IMB3 algorithm can be observed in figure 9. At Φ:1, same number of subtrees are detected since no extra candidates can be derived when mining induced subtrees (i.e. the distance between the nodes is always equal to 1).

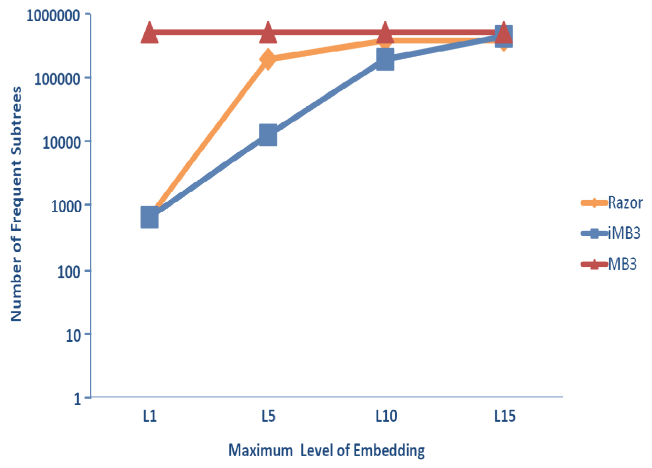


Figure 9. Varying the level of embedding

When the Φ threshold is increased Razor algorithm detects more frequent subtrees for the reasons explained in the previous section. However, when the Φ threshold was increased to 15, IMB3 algorithm detected more subtrees as frequent. When such high level of embedding is allowed many embedded subtrees previously infrequent will become frequent as there is more chance for their re-occurrence. On the other hand, the Razor algorithm may further distinguish each of those subtrees based upon the distance of nodes relative to the root, and the frequency of the new candidate subtrees may not reach the support threshold. This explains why IMB3 has detects a larger number of frequent subtrees at Φ:15, in comparison to the Razor algorithm. As expected, it can also be seen that as the Φ threshold is increased the number of frequent subtrees detected by IMB3 algorithm gets closer to the number detected by the MB3 algorithm where no Φ restriction applies.

7. CONCLUSIONS & FUTURE WORK

In this work we have extended the traditional definition of embedded subtrees in order to take the distance amongst the nodes into account. As the traditional embedding definition allows too much freedom with respect to the frequent subtrees extracted, we felt that an extra grouping criterion was required. In reality the distance between the nodes in a hierarchical structure could indicate the amount of specific information stored about that concept, which would be considered important especially for applications in web information systems and conceptual model analysis. We have presented Razor, an algorithm which groups candidate subtrees based upon the node labels, node structure, and the depth of the nodes within that structure. The correctness and implications of the approach were demonstrated with experiments using real world and synthetic data. Enforcing the distance constraint adds extra complexity to the task as more candidate subtrees will need to be enumerated and counted. We have presented an encoding strategy to efficiently enumerate candidate subtrees with the additional grouping criterion and the efficient TMG approach to candidate enumeration was preserved in yet another subset of the tree mining problem.

8. REFERENCES

[1] Abe, K., Kawasoe, S., Asai, T., Arimura, H., and Arikawa, S. Optimized substructure discovery for semistructured data. In Proc. *of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, Helsinki, Finland, 2002, 1–14.

[2] Agrawal, R. and Srikant, R. Fast Algorithm for Mining Association Rules. In *Proc. of the 20th VLDB'94*, 1994, 487–499.

[3] Bergamaschi, S., Castano, S. and Vincini, M. Semantic Integration of Semistructured and Structured Data Sources. *SIGMOD Record* 28(1), 1999, 54-59.

[4] Chi, Y., Nijssen, S., Muntz, R.R., Kok. J.N. Frequent Subtree Mining: An Overview. *Fundamenta Informaticae, Special Issue on Graph and Tree Mining*, 2005.

[5] Chi, Y., Yang, Y., Muntz, R.R. HybridTreeMiner: An efficient algorihtm for mining frequent rooted trees and free trees using canonical forms. In Proc. of *the 16th International Conference on Scientific and Statistical Database Management*, Santorini Island, Greece, 2004.

[6] Do, H.-H. and Rahm, E. COMA – A System for Flexible Combination of Schema Matching Approaches. In Proc. of *the VLDB'02*, 2002, 610-621.

[7] Feng, L., Dillon, T.S., Weigand, H., and Chang, E. An XML-Enabled Association Rule Framework. In Proc. of *DEXA'03*, 2003, 88-97.

[8] Fensel, D. *Ontologies: a silver bullet for knowledge management and electronic commerce*. Springer, Berlin, 2004.

[9] Giunchiglia, F. and Shvaiko, P. Semantic matching. *Ontologies and Distributed Systems workshop, IJCAI (2003).*

[10] Gómez-Pérez, A., Fernández-López, M. and Corcho, O. *Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic Web*. Springer-Verlag, London, 2003.

[11] Gruber, T.R., Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human and Computer Studies*, 43(5/6), 1995, 907-928.

[12] Guarino, N. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In M. T. Pazienza (ed.) *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*. Springer Verlag, 1997, 139-170.

[13] Madhavan, J., Bernstein, P.A. and Rahm. E. Generic Schema Matching with Cupid. In Proceedings of *the International Conference on very Large Data Bases (VDLB)*, 2001, 49 – 58, Rome, Italy.

[14] S. Melnik, H. Molina-Garcia, and E. Rahm. Similarity flooding: a versatile graph matching algorithm. In Proc. of *ICDE-02*, 2002.

[15] Nijssen, S., Kok, J.N. Efficient discovery of frequent unordered trees. In Proc. of *the 1st International Workshop Mining Graphs, Trees, and Sequences (MGTS-2003),* Dubrovnik, Croatia, 2003.

[16] Noy, N.F. and Musen, M. An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support. In Proceedings of *the Sixteenth National Conference on Artificial Intelligence (AAAI99), Workshop on Ontology Management*, Orlando, FL, 1999.

[17] Noy, N. F. and Musen, M. A. Anchor-PROMPT: Using non-local context for semantic matching. In *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*. Seattle, WA.

[18] Pinto, S.H. Some issues on Ontology Integration. In Proceedings of the IJCA-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Eds. Benjamins, V.R., Chandrasekaran, B., Gomez-Perez, A., Guarino, N. and Uschold, M., Stockholm, 1999.

[19] Rahm, E. and Bernstein, P. A survey of approaches to automatic schema matching. *VLDB Journal,*10, 4, 2001, 334-350.

[20] Ruckert, U. and Kramer, S. Frequent free tree discovery in graph data. In Proc. of *the 2004 ACM symposium on Applied computing*, Nicosia, Cyprus, 2004, 564 – 570.

[21] Sestito, S. and Dillon, T.S. Automated Knowledge Acquisition. Prentice Hall of Australia, Sydney, 1994.

[22] Tan, H., Dillon, T.S., Feng, L., Chang, E. and Hadzic, F. X3 Miner: mining patterns from XML Database. In Proc. of *Data Mining '05*, Skiathos, Greece, 2005.

[23] Tan, H., Dillon, T.S., Hadzic, F., Chang, E., and Feng, L. MB3 Miner: mining eMBedded sub-TREEs using Tree Model Guided candidate generation. In Proc. of *the 1st International Workshop on Mining Complex Data*, held in conjunction with ICDM'05, Houston, Texas, USA, 2005.

[24] Tan, H., Dillon, T.S., Hadzic, F., Feng, L., and Chang, E. IMB3 Miner: Mining Induced/Embedded Subtrees by Constraining the Level of Embedding, *In Proc. of PAKDD'06*, Singapore, 2006.

[25] Tan, H., Dillon, T.S., Hadzic, F., Feng, L., and Chang, E. Tree Model Guided Candidate Generation for Mining Frequent Subtrees from XML. Submitted to Transactions on Knowledge Discovery from Data (TKDD), January, 2006.

[26] Termier, A., Rousset, M-C., and Sebag, M. Treefinder: A First Step Towards XML Data Mining. In Proc. of *IEEE ICDM'02*, 2002.

[27] Wang, C., Hong, M., Pei, J., Zhou, H., Wang, W., and Shi, B. Efficient Pattern-Growth Methods for Frequent Tree Pattern Mining. *In Proc. of PAKDD'04*, 2004.

[28] Xiao, Y., Yao, J.-F., Li, Z., Dunham, M.H. Efficient data mining for maximal frequent subtrees. In Proc. of *the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, Melbourne, Florida, USA, 2003, 379-386.

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

[30] Zaki, M.J. Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications. In IEEE Transaction on Knowledge and Data Engineering, 17, 8, 2005, 1021-1035.