Razor: mining distance-constrained embedded subtrees

Henry Tan[1], Tharam S. Dillon[1], Fedja Hadzic[1] and Elizabeth Chang[2]
[1]*Faculty of Information Technology, University of Technology Sydney, Australia*
E-mail: (henryws, tharam, fhadzic)@it.uts.edu.au
[2]*School of Information Systems, Curtin University of Technology Perth, Australia*
E-mail: Elizabeth.Chang@cbs.curtin.edu.au

Abstract

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 [12], *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* [13] *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, conceptual model analysis and more sophisticated ontology matching. 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. The algorithm is applied to both synthetic and real-world datasets, and the experimental results demonstrate the correctness and effectiveness of the proposed technique.*

Keywords

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

1. Introduction

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. The problem of frequent subtree mining can be generally 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}. Many algorithms have been developed that mine different types of tree patterns. PathJoin [14], uFreqt [8], and HybridTreeMiner [3], mine induced, unordered trees. FreeTreeMiner [10] extracts free trees in a graph database. Treeminer [15] is an efficient algorithm for discovering all frequent embedded subtrees in a forest using a data structure called the vertical scope-list.

In [11], we introduced the Tree Model Guided (TMG) candidate enumeration method which is a specialization of the complete and non-redundant right most path extension approach [1, 15]. TMG generates fewer candidates as opposed to the commonly used join approach. In [12] we have introduced a novel and unique Embedding List (EL) which enables an efficient implementation of the TMG candidate generation. It was accompanied with a mathematical formula that indicates the number of candidate subtrees that will be generated at each step. Using the formula one could predict infeasible cases in which the mining process needs to be constrained 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 [13]. Thus, when it is too costly to mine all frequent embedded subtrees, one can gradually decrease the level of embedding constraint.

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. 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

Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)
0-7695-2702-7/06 $20.00 © 2006 IEEE

important and two embedded subtrees with different distance relationships among the nodes need to be considered as separate entities. This notion of distance constrained embedded tree mining will have important applications in web information systems, conceptual model analysis, knowledge merging and semantic matching. Our aim in this paper is to obtain an efficient algorithm that will extract all embedded subtrees with the additional node distance information.

The rest of the paper is organized as follows. The problem is described in section 2. Section 3 provides a motivating example. The Razor algorithm is described in section 4. The experiments on real world and synthetic data are presented in section 5, and section 6 concludes the paper.

2. 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. 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.

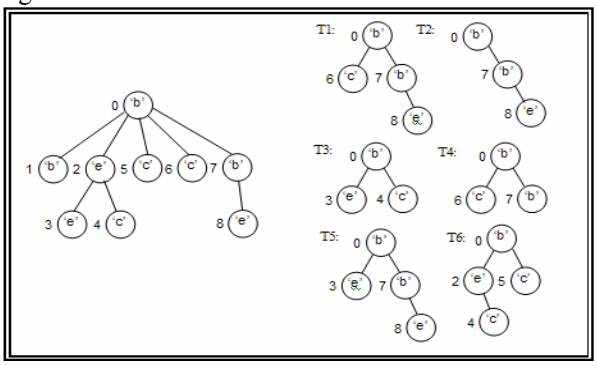


Figure 1. Example of induced subtrees (*T1, T2, T4, T6*) and embedded subtrees (*T3, T5*) of tree T

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 1 (note all induced subtrees are also embedded).

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 and Φ, 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. Hence, an induced subtree T can be defined as an embedded subtree where the maximum *level of embedding* in T is equal 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'.

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:

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

Adding distance constraint. The distance between the nodes needs to be used as an additional equality criterion to group the enumerated candidates. Consider the example tree shown in Figure 2. 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.

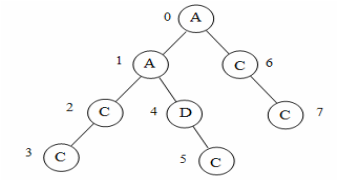


Figure 2. Example tree with labeled nodes ordered in pre-order traversal

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.

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 must also be frequent.

3. Motivating example

Automatic detection of semantic matches among ontology concepts has become the initial and most challenging stage in most of ontology merging and alignment tasks [5, 9]. The problem is analogous to schema matching in databases. Current approaches commonly utilize the schema information and the structure of the conceptual models to form mappings [4, 6, 7].

The two conceptual hierarchies (CH1, CH2) in figure 3 represent a borrowing record from two different library based applications. If the concepts at the top of the hierarchies are known to correspond to a borrowing record, the substructures containing those concepts would be compared. 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. 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 match of sub-structures where the concept granularity is the same. 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. 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.

Consulting figure 3 again, if we mine distance-constrained embedded subtrees, the largest matching subtree has 7 nodes. It corresponds to the right hand side of the CH1 plus the node in level 1, and is the largest exact match between CH1 and CH2. 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.

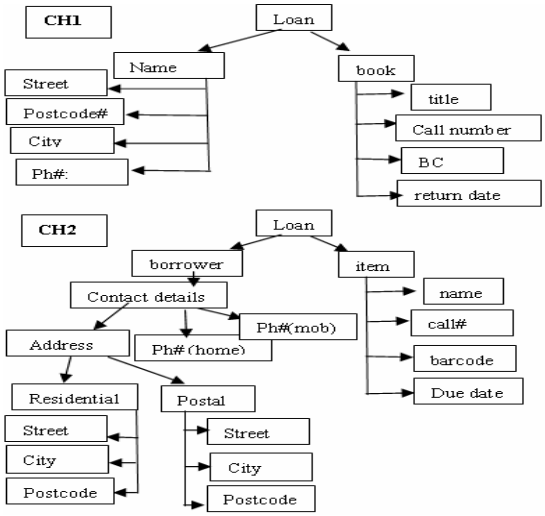


Figure 3. Libraries borrowing record schemas

4. Razor algorithm

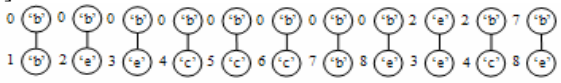
The necessary amendments for incorporating the additional distance constraint occur in the way candidate subtrees are enumerated during candidate enumeration and (k-1)-subtree generation phase. Following, is a detailed description of the algorithm with the required adjustments.

Database scanning. A tree database, T_{db}, is scanned once in order to generate a global sequence D in memory, referred to as *dictionary*. The dictionary stores each node from the T_{db} following the pre-order traversal indexing. Each dictionary item is defined as a tuple of position (pos), label (l), right-most descendant position or 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 dictionary construction the complete set of frequent 1-subtrees, F_1, is enumerated.

String encoding (φ). 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. 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. 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 1, 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, while '+' denotes an operation that appends tree encodings and computes backtrack positions.

Embedding List (EL) construction. For each frequent internal node in F_1, a list is generated which stores its descendant nodes' positions (from dictionary) in pre-order traversal ordering such that the embedding relationships between nodes are preserved. For a given internal node at position i, such ordering reflects the enumeration sequence of generating 2-subtree candidates rooted at i (figure 4). 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 in the list at slot n with zero-based indexing, and |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 1). 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.



All 2-subtree candidates generated from T

0:	1	2	3	4	5	6	7	8
2:	3	4						
7:	8							

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.

Occurrence Coordinate (OC). Each occurrence of k-subtree in T_{db} is encoded as OC 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 1 & 4, the OC of a 3-subtree (T2) with encoding 'b0 b1 e2' 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

TMG enumeration formulation. 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 φ(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∈ 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. Razor algorithm was implemented with the capability to restrict the level of embedding. Extension occurs only when the level of embedding of a node at position j to its extension point is less than Φ.

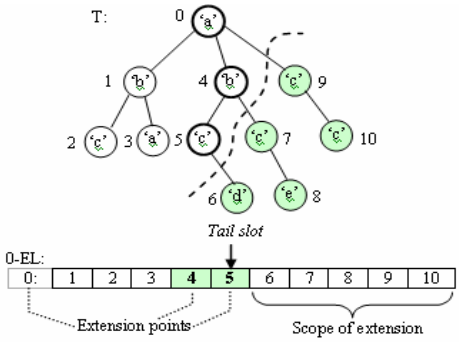


Figure 5. TMG enumeration: extending (k-1)-subtree t_{k-1} where (t_{k-1}):'a b / b c'

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 it should be pruned.

k-1 full pruning implies that at most (k-1) numbers of (k-1)-subtrees need to be generated from the currently expanding k-subtrees. The rationale of this has been discussed in [12, 15]. 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.

Vertical Occurrence List (VOL). VOL stores the OCs of a candidate subtree whose frequency is then determined from the VOL size. VOL(L) notation refers to the vertical occurrence list of a subtree with encoding L, and the frequency is denoted as |VOL(L)|. For transaction based support 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 pseudo-code for the Razor algorithm is given in Figure 6.

```
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

5. 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 [12] and IMB3 [13] 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 other methods as to our knowledge there is currently no algorithm that mines distance constrained embedded subtrees. 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.

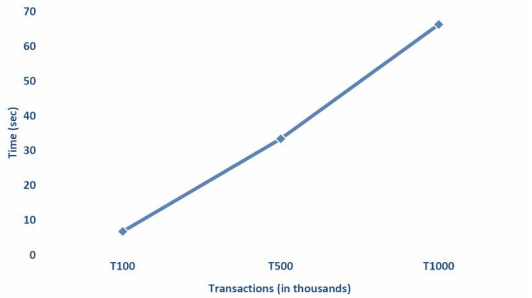


Figure 7. Scalability test - time performance / number of transactions

5.1. Scalability test

This experiment tests 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 (# of nodes) for each test is varied from 100,000, 500,000 to 1 million with minimum support 50, 250, and 500 respectively. Figure 7 shows that the time to complete the operation scales linearly with the increase in transaction size.

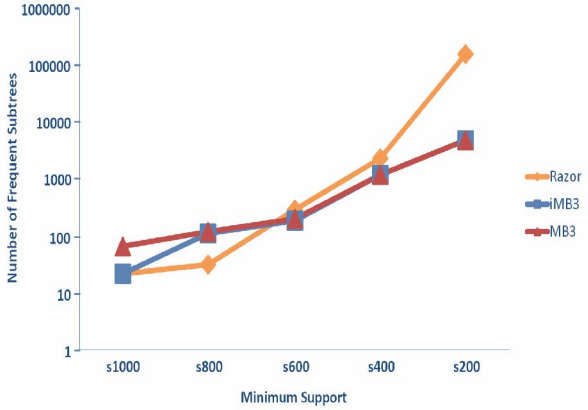


Figure 8. Number of frequent subtrees detected for varying support thresholds

5.2. Frequent subtrees over different support

For this experiment we have used a reduction of the CSLogs data set [12] previously used by Zaki [15]. IMB3 and Razor were set with the Φ constraint set to 5. The comparison of the number of frequent subtrees generated among the MB3, IMB3 and Razor algorithms, for varying support thresholds is presented in figure 8. We can see that the number of frequent subtrees detected by the Razor algorithm increases significantly when the support is lowered. With lowers support more subtrees become frequent and more variations of those subtrees with respect to the distance among the nodes also become frequent.

6.2. Varying the level of embedding

A data set characterizing a deep tree with the maximum depth equal to 17, was artificially made up of 10,000 transactions with a total of 273,090 nodes. The support threshold was set to 100. In figure 9 we compare the number of frequent subtrees detected for varying Φ thresholds. When the Φ threshold is increased Razor algorithm detects more frequent subtrees than IMB3. 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.

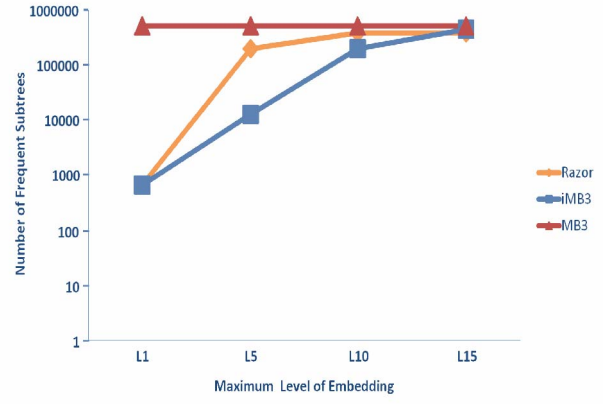


Figure 9. Varying the level of embedding

6. Conclusion

In this work we have extended the traditional definition of embedded subtrees in order to take the distance amongst the nodes into account. 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 the Razor 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.

7. References

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

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

[3] Y. Chi, Y. Yang, and R.R. Muntz, "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.

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

[5] 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.

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

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

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

[9] N.F. Noy, and M. Musen, "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.

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

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

[12] H. Tan, T.S. Dillon, F. Hadzic, E. Chang, and L. Feng, "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.

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

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

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