Implications of frequent subtree mining using hybrid support definition

F. Hadzic[1], H. Tan[1], T. S. Dillon[1] & E. Chang[2]
[1]Faculty of Information Technology, University of Technology Sydney, Sydney, Australia
[2]Curtin University, Australia

Abstract

Frequent subtree mining has found many useful applications in areas where the domain knowledge is presented in a tree structured form, such as bioinformatics, web mining, scientific knowledge management etc. It involves the extraction of a set of frequent subtrees from a tree structured database, with respect to the user specified minimum support. To date, the commonly used support definitions are occurrence match and transaction based support. There are some application areas where using either of these support definitions would not provide the desired information automatically, but instead further querying on the extracted patterns needs to take place. This has motivated us to develop a hybrid support definition that constrains the kind of patterns to be extracted and provides additional information not provided by previous support definitions. This would simplify some of the reasoning process which commonly takes place in certain applications. In this paper we demonstrate the need for the hybrid support definition by presenting some applications of tree mining where traditional support definitions would fall short in providing the desired information. We have extended our previous tree mining algorithms to mine frequent subtrees using the hybrid support definition. Using real-world and synthetic data sets we demonstrate the effectiveness of the method, and further implications for reasoning with the extracted patterns.
Keywords: frequent subtree mining, hybrid support, knowledge merging.

1 Introduction

In domains where the formally represented information is of tree structured form, frequent subtree mining techniques can be used for efficient querying and analysis of the domain knowledge. Tree structured information is increasingly found in biomedical, web and scientific domains and hence tree mining is a popular technique for many applications in these domains. 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}. Within this framework the two most commonly mined subtrees are induced and embedded. An induced subtree preserves the parent-child relationships of each node 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 ancestor-descendant relationships are preserved over several levels. Depending on whether the order of sibling nodes is to be considered important these subtrees can be further split in ordered and unordered subtrees.

There have been a number of algorithms developed for frequent subtree mining and the scope of their application usually depends on the assumptions made about the data structure that the algorithm can be applied to and to the types of subtrees extracted. Some of the algorithms for extracting frequent embedded subtrees from a database of rooted ordered labeled subtrees are Treeminer [1], X3-Miner [2], MB3-Miner [2], whereas AMIOT [4] mines induced ordered trees. UNI3 [5], PathJoin [6], uFreqt [7], and HybridTreeMiner [8], mine induced, unordered subtrees, and SLEUTH [9] and U3 [10] extract embedded unordered subtrees. FreeTreeMiner [11] extracts free trees in a graph database. From the application perspective, some tree mining algorithms have been successfully applied to biological domains [12–14].

Our work in the area of frequent subtree mining is characterized by adopting a Tree Model Guided (TMG) candidate generation as opposed to the join approach which is commonly used. This non-redundant systematic enumeration model ensures only valid candidates are generated which conform to the actual tree structure of the data. Furthermore, our unique Embedding List representation of the tree structure has allowed for an efficient implementation of the TMG approach which has resulted in efficient algorithms for mining embedded (MB3) [3] and induced (IMB3) [15] subtrees, from a databases of labeled rooted ordered subtrees. MB3^{-R} and IMB3^{-R} algorithms (IMB3Jref) are latest implementations that adopt a more space efficient global representation and only store the right most path information for candidate subtrees. Motivated by the fact that in many applications of frequent subtree mining the order among siblings is not considered important we have recently shifted our focus to unordered tree mining and have developed the UNI3 [5] and U3 [10] algorithms for mining frequent unordered induced subtrees. At this stage we are applying our developed algorithms to the ontology learning and matching problems. In this process we notice that there are some changes required for the algorithms to be efficiently applied for the ontology specific problems. The first change we propose in this paper is an extension to the current support definitions.

Currently there are two support definitions used for determining the frequency of a subtree, and both are suitable for certain applications. Transaction based support (TS) [1] is used when only the existence of items within a transaction is considered important, whereas occurrence match (OC) [1, 3] support takes the repetition of items in a transaction into account and counts the subtree occurrences in the database as a whole. In certain applications (an example of which will be provided in the next section) the number of times a subtree occurs within a transaction is of interest. Current support definitions fall short on providing such information since they either just check for the existence of a subtree in a transaction using TS or count the total number of occurrences using OC. Therefore, we felt that an additional support definition was required which will appropriately restrict the kind of subtree patterns extracted and provide intra-transactional occurrence information for each subtree. In this paper we provide this 'hybrid' support definition and demonstrate the implications of using such support. We discuss some scenarios where this support definition is useful for providing the necessary information that could not be obtained using TS or OC support definitions. Our previously developed tree mining algorithms were extended to mine frequent subtrees using the hybrid support definition, and we describe our general TMG framework with the new support definition in place.

The rest of the paper is organized as follows. Section 2 provides a motivating example and discusses some scenarios where the hybrid support definition would be useful. The tree mining problem is briefly presented in Section 3. Section 4 provides an overview of our general TMG approach to tree mining with the additional capability of using the hybrid support definition. Some experiments on real world data are presented in Section 5, and Section 6 concludes the paper.

2 Motivating example

Automatic detection of semantic matches among ontology concepts has become the initial and most challenging stage in most of ontology learning and matching tasks [17, 18]. We approach the ontology learning problem by merging the knowledge models which have been provided by different organization for the same domain. If the knowledge models are successfully merged, i.e. a shared agreement on the conceptualization of knowledge is obtained, we have obtained an ontology for those organizations. Our intention in providing semantic mappings among the concepts in knowledge models is to mainly exploit the structure of the knowledge and avoid using any string match operators since they are not always reliable. If a tree mining algorithm is to be efficiently applied to this problem, then each knowledge model is most likely to be represented as one transaction inside the tree database. The most promising initial match would be the concept that occurs many times in all transactions. Once an initial match is made we could proceed onto matching sub-structures that occur multiple times within a transaction. As a simple illustrative example please consider Fig. 1, where we present some knowledge models obtained using different data mining tools on the publicly available 'wine' dataset obtained from [19].

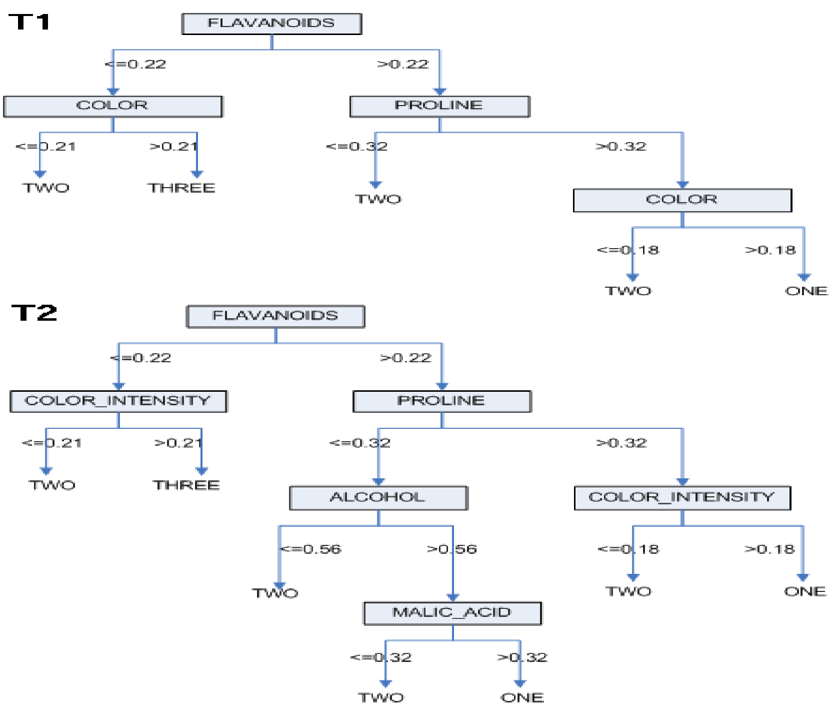


Figure 1: Tree database consisting of transaction T1 and T2 corresponding to knowledge models for 'wine' domain.

In real world the data collected by different organizations can differ in various aspects. Certain attributes may be found important by one organization and irrelevant by another. To mimic the real-world scenario we have used different sets of data and/or different pre- and post-pruning options for the data mining techniques. This introduced some differences in the level of detail among the obtained knowledge models. When knowledge models coming from different organizations are to be merged, the concept names are usually different. Hence, string match operators may not be appropriate even though it appears from the example here that exact matches could be easily found. In Fig.1 each knowledge model is considered as a separate transaction within the tree database. As can be seen there are a few concepts that occur multiple times in each transaction The concepts 'color' and 'color_intensity' appear two times in T1 and T2, respectively. This provides a promising initial match and the similarity could be propagated throughout the neighboring concepts. More generally, if there are 'n' knowledge models to be merged, one would mine patterns that occur in all n transaction (transaction based support = n) and that occur multiple times in all n transactions. This would not be possible using transaction or occurrence match support solely and hence the need for the hybrid support definition. Other examples could be taken from many web information systems applications, where specialized queries on tree structured databases commonly take place.

Consider a library based application where author information may be separately stored in each transaction. A user may be interested in finding a number of authors that have published at least X books with publisher Y. To satisfy this query, the repetition of author-book-publisher relation within a transaction will need to be considered. In these scenarios where the repetition of items within a transaction is considered important hybrid support would provide useful information automatically without any post processing which would need to occur if either occurrence match or transaction based supports were used.

3 Problem statement

This section provides a general definition of the problem of frequent subtree mining. Due to the space limitations and the current scope of our work, we do not provide a detailed overview of the basic tree concepts, but refer the reader to our previous works [3, 15], where such information has been provided.

Mining frequent subtrees. Let T_{db} be a tree database consisting of N transactions of trees, K_N. The task of frequent subtree mining from T_{db} with given minimum support (σ), is to find all candidate subtrees that occur at least σ times in T_{db}.

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.

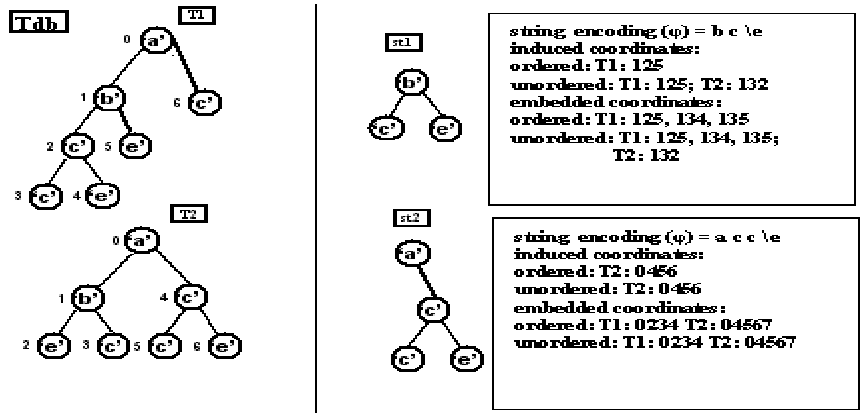


Figure 2: Example tree database (Tdb) with two transactions (T1 & T2).

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 $v' \in V'$, $v \in V$ 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.

An unordered embedded or induced subtree would be the same as above, except that condition 5 is relaxed so that the left-to-right ordering among the siblings does not need to be preserved. For examples of different subtree types please consider Fig. 2. Here we display an example database (Tdb) and two subtrees st1 and st2. On the right of each subtree we display the coordinates of valid occurrences of the subtree in each transaction when mining a particular subtree type.

Support Definitions. We use $t \prec k$ to denote an embedded subtree t that is supported by transaction $k \subseteq K$ in database of tree $T_{db.}$ This occurs when k contains at least one occurrence of t. If there are L occurrences of t in k, let function *g(t,k)* denote 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 transaction k. In other words, for transaction-based support, the support of a subtree t is equal to the numbers of transactions that support subtree t. For *occurrence-match support*, $t \prec k$ corresponds to the number of all occurrences of t in transaction 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 an embedded subtree t in T_{db} is defined as:

$$\sum_{i=1}^{N} t \prec k_i \tag{1}$$

Hybrid Support. As the name implies for this support definition we are combining transaction based with occurrences match support. The support threshold is denoted by 'x|y', where 'x' denotes the number of transactions that support subtree t, and y denotes the number of times that t must occur in those x transactions. Hence, using hybrid support threshold of x|y, a subtree is considered frequent iff it occurs in x transactions and it occurs at least y times in each of the x transactions.

4 Integrating hybrid support

This section first provides an overview of our general TMG framework that has been applied to a variety of tree mining problems, and then proceeds onto discussing the candidate counting phase which is where hybrid support integration takes place. Since TMG framework has been presented in many of the previous works we only provide a quick overview and refer the reader to [3, 5, 15] for more details. Please note that there could be some slight differences since a few optimizations took place between the development of new algorithms. To speed up the processing, the database of XML documents is first transformed into a database of rooted integer-labeled ordered tree. The tree database is traversed once to create a global sequence which stores each node in the pre-order traversal together with the necessary node information (position, label, scope). At the same time the set of frequent 1-subtrees is obtained by hashing the encountered node labels. The tree database representation which enables efficient candidate generation is constructed. TMG candidate generation takes place and for each $k \geq 1$ the right most path coordinates of each frequent (k-1)-subtree (subtree consisting of k-1 nodes) are stored in 'F_{k-1}' hashtable.

Each frequent (k-1)-subtree is extended one node at a time, starting from the last node of its RMP (right most node), up to its root, whereby all k-subtrees are enumerated. The whole process is repeated until all k-subtrees are enumerated and counted.

The main difference with the integration of the hybrid support definition lies in the way the subtree occurrences are counted and hence the way the frequency is determined. A detailed discussion of this aspect of our algorithm follows.

Candidate subtree counting. To determine if a subtree is frequent, we check whether the number of times a subtree occurs is greater or equal to the specified minimum support σ. In a database of labeled trees many instances of subtrees can occur with the same encoding. Hence, the notion of encoding is utilized in the candidate counting process. We say that a subtree with encoding *L* has a frequency n if there are n instances of subtrees with the same encoding *L*, i.e. we group subtree occurrences by its encoding.

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 *Tdb* is encoded as *occurrence coordinate r:[e1,…ek-1]; r* refers to k-subtree root position and *e1,…,ek-1* refer to the positions of the rest of the nodes ordered in pre-order traversal. Since we utilize our tree representation for efficient TMG candidate generation the positions correspond to the slots in the structure. However, to keep the explanation simple node positions will refer to the node positions in the tree database (Tdb). From fig. 2, the OCs of the ordered embedded 3-subtree 'st1' with encoding '*b c / e*' in T1 are encoded as 1:[2,5], and OCs of the ordered embedded 4-subtree 'st2' in T2 with encoding '*a c c / e*' are encoded as 0:[4,5,6]. Each OC of a subtree describes an instance of each occurrence of the subtree in *Tdb*. Hence, each *candidate instance* has an OC associated with it. The storage requirement for longer subtrees can grow significantly if we have to store each coordinate of each node in long subtrees.

RMP Occurrence Coordinate (RMP-OC). By its definition, RMP is the shortest path from the right most node to the root node. Thus storing RMP coordinates is always guaranteed to be maximal. The worst case of storing the RMP coordinates would be equal to storing every coordinate of a node in a subtree, i.e. when the subtree becomes a sequence (each node has degree 1). The best case of storing RMP coordinates for k-subtrees where $k>1$ is that it stores only 2 coordinates, i.e. whenever the length of the RMP is equal to 1. 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). Each occurrence of a subtree is stored as RMP-OC in VOL as previously described. The VOL of a subtree groups the RMP-OCs of the subtree by its encoding and it is used to count the occurrence-match support and transaction-based support. For occurrence-match support we suppress the notion of the transaction id (*tid*) that is associated with each RMP-

OC. For transaction-based support the notion of *tid* of each occurrence coordinate is accounted for when determining the support. This can be seen in Fig. 3 where the tid is stored to the left of the table. When the occurrence-match support is used, the frequency of ordered embedded subtree ‘st1’ of tree database *Tdb* (fig. 2) with encoding ‘*b c / e*’, is equal to the size of the VOL, i.e. 3 (fig. 3). When transaction-based support is used the support of ‘st1’ is equal to 1 since it occurs in only one transaction. In example from fig. 3 there is only 1 transaction (*tid*:1) that supports subtree ‘st1’. If we are considering hybrid support definition then the support of ‘st1’ is 1|3, since it occurs three times in one transaction. On the other hand if we were mining unordered embedded subtrees, then the hybrid support of ‘st1’ would equal 2|1 since there is an additional occurrence of ‘st2’ in transaction T2 and ‘st2’ occurs at least once in each transaction (as opposed to three times when it only occurred in T1).

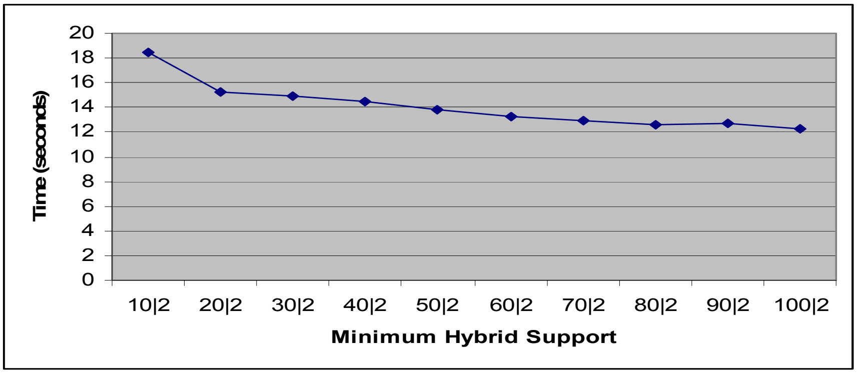
1	**1**	**5**
1	**1**	**4**
1	**1**	**5**
‘b c / e’		

Figure 3: VOL(‘b c / e’) of ‘st1’ in fig. 2 when embedded subtrees are mined.

5 Experimental results and discussions

This section provides some experiments performed on the Prions dataset that describes Protein Ontology database for Human Prion proteins in XML format [20]. The dataset consists of 17511 transactions. The 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 –g and –O3 parameters. The total run-time and the number of frequent subtrees detected is displayed in Fig. 4, for the IMB3^{-R} [16] algorithm when varying hybrid support thresholds were used. To our knowledge there are no other algorithms that use the same hybrid support definition and hence providing comparisons with other tree mining approaches was not possible at this stage. However, from Fig.4 we can see that the approach is well scalable and the efficiency of our general TMG approach to tree mining is preserved when hybrid support definition is integrated.

Please note that we have intentionally used small hybrid support thresholds to demonstrate the scalability of approach for cases when many frequent subtrees exist. However, in practice one may use larger support thresholds to limit the frequent subtrees to those that occur in a large percentage of transactions. This is of course dependent on the user interests and the overall aim of the application. In regards to the application of tree mining algorithms to Protein Ontology datasets, the use of the hybrid support definition allows for easier comparisons of protein datasets taken across protein families and species. The extracted patterns would help in discovering of interesting similarities and differences among protein families.



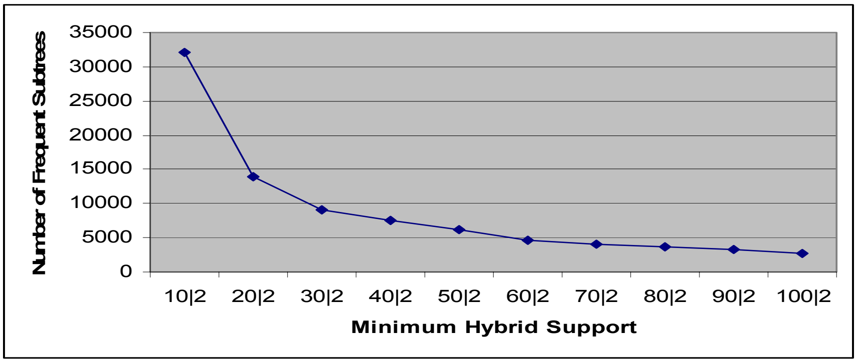


Figure 4: Total time taken (top) and number of frequent subtrees extracted (bottom) from Prions dataset with varying hybrid support thresholds.

6 Concluding remarks

In this paper we have provided a new hybrid support definition within the tree mining framework. For certain applications an additional support definition was required which will appropriately restrict the kind of subtree patterns extracted and provide intra-transactional occurrence information for each subtree. We have discussed some scenarios where this support definition is useful for providing the necessary information that could not be obtained using TS or OC support definitions. Our previously developed tree mining algorithms were extended to mine frequent subtrees using the hybrid support definition and experiments on real world data demonstrated the effectiveness of the method.

References

[1] Zaki, M. J., Efficient Mining of Trees in the Forest. *SIGKDD '02*, Edmonton, Alberta, Canada, ACM, 2002.

[2] Tan, H., Dillon, T.S., Feng, L., Chang, E., Hadzic, F., X3-Miner: mining patterns from XML database. *Data Mining 2005*, Skiathos, Greece, 287-297, 2005.

[3] 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*, Houston, Texas, USA, 2005.

[4] Hido, S. & Kawano, H., AMIOT: Induced Ordered Tree Mining in Tree-structured Databases, In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)*, Houston, Texas, USA, 2005.

[5] Hadzic, F., Tan, H., Dillon, T.S., UNI3: efficient algorithm for mining unordered induced subtrees using TMG candidate generation, To appear in *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007)*, Honolulu, Hawaii, April 1-5, 2007.

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

[7] Nijssen, S. & Kok, J.N., Efficient discovery of frequent unordered trees. In *Proc. of the 1st International Workshop Mining Graphs, Trees, and Sequences*, Dubrovnik, Croatia.

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

[9] Zaki, M.J., Efficiently Mining Frequent Embedded Unordered Trees, *Fundamenta Informaticae 65*, IOS Press, pp. 1-20, 2005.

[10] Hadzic, F., Tan, H., Dillon, T.S., Chang, E., U3 – Unordered subtree mining using TMG candidate generation and the level of embedding constraint, Submitted to the 13th International Conference on Knowledge Discovery and Data Mining, San Jose, California, Aug 12-15, 2007.

[11] Chi, Y., Yirong, Y., Muntz, R. R., Canonical Forms for Labeled Trees and Their Applications in Frequent Subtree Mining, *Knowledge and Information Systems*, 2004.

[12] Hadzic, F., Dillon, T. S., Sidhu, A., Chang, E., Tan, H., Mining Substructures in Protein Data, *IEEE ICDM 2006 Workshop on Data Mining in Bioinformatics (DMB 2006)*, in conjunction with the 2006 International Conference on Data Mining, 18-22 December, Hong Kong,

[13] Shasha, D., Wang, J.T.L., Zhang, S., "Unordered Tree Mining with Applications to Phylogeny", *20th International Conference on Data Engineering*, 2004.

[14] Wang, J. T. L., Shan, H., Shasha, D., and W. H. Piel, Treerank: A similarity measure for nearest neighbor searching in phylogenetic databases, In Proc. of the 15th Intl. Conf. on Scientific and Statistical Database Management (SSDBM'03), 2003.

[15] Tan, H., Dillon, T.S., Hadzic, F., Feng, L., Chang, E. IMB3-Miner: Mining Induced/Embedded subtrees by constraining the level of embedding. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2006)*, Singapore, 2006.
[16] Tan, H., Dillon, T.S., Hadzic, F., Chang, E., Feng, L., Mining induced/embedded subtrees using the level of embedding constraint, Submitted to *Fundamenta Informaticae*, IOS Press, 2007.
[17] Giunchiglia, F. & Shvaiko, P., Semantic matching, *Ontologies and Distributed Systems workshop*, IJCAI, 2003.
[18] Gómez-Pérez, A., Fernández-López, M., Corcho, O. *Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic Web.* Springer-Verlag, London, 2003
[19] Blake, C., Keogh, E. & Merz, C.J., 1998. "UCI Repository of Machine Learning Databases", Irvine, CA: University of California, Department of Information and Computer Science., 1998. [http://www.ics.uci.edu/~mlearn/MLRepository.html].
[20] Sidhu, A.S., Dillon, T.S., Sidhu, B.S., Setiawan, H., A Unified Representation of Protein Structure Databases, in *Biotechnological Approaches for Sustainable Development*, M. S. Reddy and S. Khanna, Eds. India: Allied Publishers, pp. 396-408, 2004.