Tree Mining Application to Matching of Heterogeneous Knowledge Representations

Fedja Hadzic[1], Tharam S. Dillon[1], Elizabeth Chang[1]
[1]Digital Ecosystems and Business Intelligence Institute, Curtin University of Technology, Perth, Australia
fedja.hadzic@postgrad.curtin.edu.au
{tharam.dillon, elizabeth.chang}@cbs.curtin.edu.au

Abstract

Matching of heterogeneous knowledge sources is of increasing importance in areas such as scientific knowledge management, e-commerce, enterprise application integration, and many emerging Semantic Web applications. With the desire of knowledge sharing and reuse in these fields, it is common that the knowledge coming from different organizations from the same domain is to be matched. We propose a knowledge matching method based on our previously developed tree mining algorithms for extracting frequently occurring subtrees from a tree structured database such as XML. Using the method the common structure among the different representations can be automatically extracted. Our focus is on knowledge matching at the structural level and we use a set of example XML schema documents from the same domain to evaluate the method. We discuss some important issues that arise when applying tree mining algorithms for detection of common document structures. The experiments demonstrate the usefulness of the approach.

1. Introduction

Knowledge discovery task in general can be hard and time consuming, and hence sharing the already developed knowledge representations is desirable. This particularly occurs when a number of organizations coming from the same domain would like to have a knowledge basis on which they can integrate their organization specific knowledge. Having a general knowledge model would save time and costs associated with having to acquire general knowledge about the domain at hand. Furthermore, the knowledge base should be represented in a machine readable way so that it can also be used by communicating agents. This gave rise to the development of ontologies. Ontology in AI is defined as a formal, explicit specification of a shared conceptualization [1]. Developing domain ontologies for capturing domain specific knowledge is often characterized by merging of existing knowledge representations of the same domain [2, 3]. Automatic detection of common structures among existing domain knowledge representations can contribute to the process of automating the ontology building and matching task.

While many general knowledge representations exist where the underlying structure is a graph, in this work we narrow our focus on the matching of knowledge representations where the information is represented in a tree or semi-structured form (eg. XML). Hence, our approach could only contribute to the automation of ontology building and matching process, if the underlying structure of the knowledge representation is in form of a tree. Many web services use XML as a unified exchange format, as it provides the extensibility and language neutrality that is the key for standard–based interoperability between different software applications [2, 4]. The process of discovering particular web services and composing them together in order to accomplish a specific goal is an important step toward the development of 'semantic web services' [2, 4, 5]. In this process, being able to detect common knowledge structures between the information presented by the services to be integrated will be a useful step toward automation.

Matching of knowledge structures has been of interest for a long time and many useful applications can be found in scientific knowledge management, e-commerce, enterprise application integration, scientific knowledge management, etc. The emergence of semi-structured data sources (such as XML) that are commonly used for describing domain knowledge has called for the development of methods capable of efficiently analyzing such documents. The TreeDiff software package [6] takes two XML documents as input, represents them as ordered labeled trees and finds sequence of edit operations to transform one document tree into another. In [7] it is noted that finding common structures among semi-structured documents is useful for document clustering methods, since the structure is usually ignored by traditional clustering approaches. They presented an algorithm for finding the minimum number of operations for transforming one data tree into another. The tree matching problem in context of change detection of hierarchically structured information was studied in [8, 9]. An algorithm that finds largest approximate common substructures between ordered labeled trees has been presented in [10]. In [11] an algorithm was presented for identifying changes in XML documents. It works by identifying unchanged portions of the documents and then inspecting the neighboring nodes using XML specific information to find further matches. A structural similarity metric for measuring structural similarity among XML documents has been proposed in [12], and is based upon a specialized tree edit distance among ordered labeled trees. In [13] the authors proposed a lower and upper bound on the tree distance metric between ordered labeled trees. This extension to the metric is more computationally efficient and takes the structural property of XML into account.

The XML matching method presented in this paper is based on the use of our previously developed tree mining algorithms in order to automatically extract shared document structures. By using the tree mining approach many of the structural differences among the knowledge representations can be detected and the largest common structure is automatically extracted. The implications of mining different subtree types are discussed and the most suitable subtree type within the current tree mining framework is indicated. The experiments are performed on a set of XML schemas used for describing organization specific information. In general, the method is applicable for matching of any tree structured knowledge representations. To limit the scope of the current work in this paper we focus solely on knowledge structure matching and do not yet consider the problem of matching at the conceptual level. Hence the main purpose of the work is to demonstrate the potential of applying tree mining methods to the problem of knowledge matching which brings it a step closer towards automation. We have previously applied our tree mining algorithms on large and complex tree structures and experimentally demonstrated their scalability [14, 15]. Other studies [16, 17] also indicate that tree mining algorithms are generally well scalable to large tree structures. In this paper we consider smaller trees in order to illustrate the underlying concept in a more comprehensible manner.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the tree mining problem and discusses some of our developed algorithms in the area. Our approach to matching of tree structured documents is described in Section 3. Section 4 describes the results of applying our tree mining algorithms for finding common XML schema structures. The paper is concluded in Section 5.

2. Frequent Subtree Mining

This section starts by providing formal definitions of some basic tree concepts, and then proceeds into an overview of our current contributions to the area of tree mining. Only concepts necessary for understanding the current work are defined. For a more extensive overview of the area including various implementation issues and algorithm comparisons we refer the interested reader to [14, 15, 17].

A tree can be denoted as $T(v_0,V,L,E)$, where (1) $v_0 \in V$ is the root vertex; (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) $E = \{(x,y) | x,y \in V\}$ is the set of edges in the tree. A *root* is the topmost node in the tree. The *Parent* of node v is defined as the predecessor of node v. A node v can only have one parent while it can have one or more *children*. A node without any child 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 number of children of a node is commonly termed as *fan-out/degree* of the node. A path from vertex v_i to v_j, is defined as the finite sequence of edges that connects v_i to v_j, and in a tree there is a single unique path between any two vertices. The length of a path p is the number of edges in p. If p is an *ancestor* of q, then there exists a path from p to q.

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 types of 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. Formal definitions follow.

Induced subtree. A tree $T'(r', V', L', E')$ is an *induced subtree* of a tree T (r, V, L, E) *iff* (1) $V' \subseteq V$, (2) $E' \subseteq E$ and (3) $L' \subseteq L$ and $L'(v)=L(v)$.

Embedded subtree. A tree $T'(r', V', L', E')$ is an *embedded subtree* of a tree $T(r, V, L, E)$ iff (1) $V' \subseteq V$, (2) if $(v_1,v_2) \in E'$ then $parent(v_2) = v_1$ in T', only if v_1 is ancestor of v_2 in T and (3) $L' \subseteq L$ and $L'(v)=L(v)$.

Level of embedding. If $T'(r', V', L', E')$ is an embedded subtree of T, and two nodes $p \in V'$ and $q \in V'$ form an ancestor-descendant relationship, the *level of embedding (δ)* is defined as the length of the path between p and q. A *maximum level of embedding* (Φ) is the limit on the level of embedding between any p and q. In other words, given a tree database T_{db} and Φ, then any embedded subtree to be generated will have the maximum length of a path between any two ancestor-descendant nodes equal to Φ. In this regard, we could define an 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.

In addition to the previous definitions, the subtrees can be further distinguished based upon the ordering of siblings. An ordered subtree preserves the left-to-right ordering among the sibling nodes in the original tree while in an unordered subtree this ordering is not necessarily preserved. In other words, for an unordered subtree the order of the siblings (and the subtrees rooted at sibling nodes) can be exchanged and the resulting subtree would be considered the same. Examples of different subtree types are given in Figure 1 below.

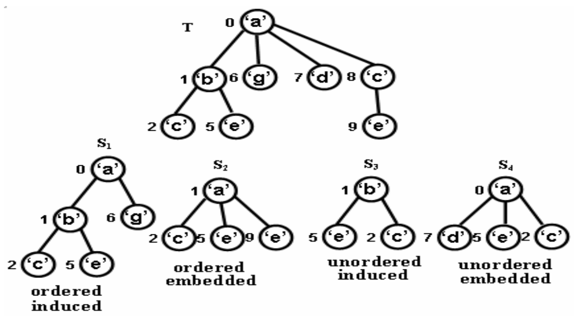

Figure 1. Example of a tree T and its different subtree types (induced subtrees are also embedded).

To determine the frequency of a subtree, most commonly used support definitions are transaction based and occurrence match support [14, 15, 16] and the choice is application dependant. In the data mining field the term transaction has been defined as a set of one or more items obtained from a finite item domain, and a dataset as a collection of transactions [18]. Hence, in context of a tree database, a transaction would correspond to a fragment of the database tree whereby an independent instance is described. Transaction based support (TS) is used when only the existence of items within a transaction is considered important, whereas occurrence match (OC) support takes the repetition of items in a transaction into account and counts the subtree occurrences in the database as a whole. Recently, we have provided the hybrid support definition [19]. 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 those x transactions. Hence, in addition to transactional support it keeps the extra information about the intra-transactional occurrences of a subtree.

Our work in the area of frequent subtree mining is characterized by adopting a Tree Model Guided (TMG) [20] 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. TMG can be applied to any data that has a model representation with clearly defined semantics that have tree like structures. In the cases where the model representation of the tree structure is unavailable the TMG approach will still perform the candidate generation according to the tree structure of the document, by obtaining the model from the document itself. Furthermore, our unique Embedding List [20] representation of the tree structure has allowed for an efficient implementation of the TMG approach which resulted in efficient algorithm MB3-Miner [20] for mining of ordered embedded subtrees. In the same work we presented the TMG mathematical model for estimating the worst case complexity of enumerating all embedded subtrees. The large complexity of mining embedded subtrees motivated our Level of Embedding [14] constraint so that one can decrease the level of embedding constraint gradually down to 1, from which all the obtained subtrees are induced. Razor algorithm [21] was a further extension developed for mining embedded subtrees where the distance of nodes relative to the root of the subtree needs to be considered. Motivated by the fact that in many applications of frequent subtree mining the order among siblings is not considered important we have recently extended our general TMG framework for the unordered tree mining problem. We developed the UNI3 [22] and U3 [15] algorithms for mining frequent unordered induced and embedded subtrees, respectively. From the application perspective, in [23] we have indicated the potential of the tree mining algorithms in providing interesting biological information when applied to tree structured biological data. Since the order of sibling concepts is not considered important when comparing different knowledge structures, it is the unordered tree mining that will have best applications for the problem of knowledge matching.

3. Overview of the method

In this section we provide an overview of our proposed method for knowledge structure matching. The approach described is motivated by a real world scenario of different organizations using different knowledge structures for representing their domain related information. The organizations may want to discover the general knowledge of the domain that is shared in all knowledge representations in order to obtain a shared understanding of the domain. Furthermore, the shared knowledge structure can be used as the knowledge basis to be used by new organizations. It is also analogous to the problem of building a domain specific ontology from a number of existing knowledge bases. In real world, it is common that different organizations use different names for same concepts but this is a different problem of concept matching which is out of the scope of the current work. Hence, the current assumption is that all concept names are the same, or some concept matching algorithm has already been applied to find the corresponding mappings.

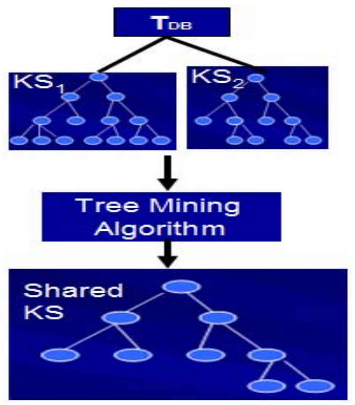

Figure 2. Proposed knowledge matching approach.

Figure 2 illustrates our general approach taken to the matching of tree structured knowledge and at the same time describes our experimental setup in this paper. While only two knowledge structures (KSs) are displayed in the figure the approach is valid when the number of available KSs is larger. Suppose that we have two document structures used by different organizations for representing the knowledge from the same domain. The aim is to merge them into one representation which captures the general knowledge for that specific domain. Each KS is most likely to differ in the way the knowledge is represented and the amount of concept granularity. However since they are describing the same domain there will be some common parts of knowledge. This is where a tree mining approach will prove useful since sub-structures from large tree databases can be automatically extracted.

The first step is to set up a tree database (T_{DB}) so that each KS is represented as an independent subtree (i.e. transaction). A tree mining algorithm can then be applied to the T_{DB} in order to extract the common knowledge structure (shared KS) among each KS. Since each KS is represented as a transaction, transactional support will be used and set to equal the number of KSs present in the T_{DB}. Now we need to discuss which specific tree mining algorithm should be used, or in other words what subtree type should be extracted.

As mentioned in the previous section, when comparing the similarity among knowledge structures the order of sibling nodes is irrelevant, since exchanging the order of sibling concepts does not change the general information content. Hence, unordered as opposed to ordered subtrees will be mined. One further choice to make is whether we are interested in mining induced or embedded subtrees. The difference occurs in the fact that if embedded subtrees are mined we are allowing sub-structures to be considered common even if they occur at different levels in the KS. In an embedded subtree the relationships are not limited to parent-child, and hence allowing ancestor-descendant relationships enables the extraction of more sub-structures where the levels of embeddings are different. It is worth noting here that the specific knowledge about a concept is not always stored in a way just described. It is commonly the case that extra specific information about a concept is stored in form of additional child nodes of that concept. However since both induced and embedded subtrees keep all the parent-child information relationships from the original subtree this difference does not influence the choice of the subtree type to be mined. In regards to these observations our U3 [15] algorithm will be used in order to extract the largest common unordered embedded subtree from the KSs in the T_{DB}.

The shared KS detected could be less specific than the KS from a particular organization, but it is therefore valid for all the organizations. Furthermore, each of the different organizations could have their own specific part of knowledge which is only valid from their perspective, and which can be added to the shared KS so that every aspect for that organization is covered. Hence, the shared KS can be used as the basis for structuring the knowledge for that particular domain and different communities of users can extend this model when required for their own organization specific purposes.

4. Experimental Evaluation

This section describes the knowledge models used in the experiments and shows the resulting common document structure as detected by the U3 [15] algorithm. The example XML (schema) documents were obtained from the ontology matching website [24]. The documents correspond to a representation of specific domain information used by different organizations. Please note that in our examples the node labels concepts describing the same concepts have been replaced where necessary by a common name, since we are not addressing the problem of concept matching in this work.

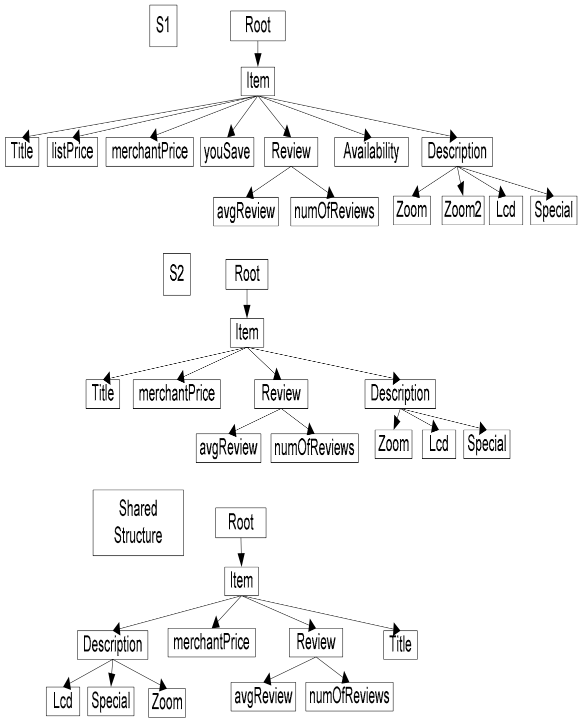

Figure 3. XML schema structures about sale item description by Amazon (S1) and Yahoo (S2), and their shared document structure

In Figure 3 we can see the underlying tree structures form the XML schema documents used by Amazon (S1) and Yahoo (S2) for describing a sales item. Each of those tree structures was represented as a separate transaction in the XML document that we used as input to our U3 algorithm [15] in order to extract the largest unordered embedded subtree. We have used the transaction support with the threshold set to 2 since we are comparing only two document structures. The largest detected subtree is presented at the bottom of Figure 3. The XML schema used by Amazon has more specific information for describing the sale information of an item. As can be seen in Figure 3, the sibling node order has changed in the shared structure. This is because an algorithm for mining unordered subtree has to use a canonical form of a subtree [25, 15, 22], according to which candidate subtrees will be converted and grouped. In the canonical form used by the U3 algorithm [15] the sibling nodes are ordered according to the alphabetical order and the node with the smallest starting letter is placed to the left of the subtree (as shown in shared structure of Figure3). This process is required so that all the subtrees with different order of sibling nodes describing the same concept, are still grouped to one candidate.

In scenario from Figure 3, it would have even been sufficient to mine ordered induced subtrees since no specific concept information was stored at different levels of the tree and all the common concepts were presented in the same order. However, one cannot assume that the compared knowledge representations will have their common concepts ordered in the same way, and that a particular set of concepts will not be grouped under a certain criterion in different representations. In this paper we have used smaller examples for a clearer illustration of the underlying concept. However, when compared document structures are much larger it is more likely that the same concept information will be stored in different order and at different levels in the tree. In the next scenario we will consider an example to illustrate that it is important to relax the order of sibling nodes and to extend the relationships between the nodes from parent-child to ancestor-descendant (i.e. extracting embedded subtrees as opposed to induced).

In Figure 4 we represent two trees which correspond to the XML schemas used by different organizations for describing their purchase order (PO) commercial document. The largest common document structure is displayed last and by comparing it to S1 and S2 one can see that it was necessary to extract largest unordered embedded subtree. First of all the order of nodes describing the address information was different, and the nodes were stored at different levels in the tree. Hence the common relationship between the 'DeliverTo' nodes and the address details would not be detected if largest induced subtree was extracted, since the level of embedding (see Section 3) is equal to 2, while the allowed level of embedding in an induced subtree is limited to 1. The largest common induced subtree would only consist of the root node 'PO' and two nodes emanating from the node 'PO' i.e. 'DeliverTo' and 'BillTo'. The remaining common structures would be missed altogether and hence mining embedded subtrees in these scenarios is a necessity.

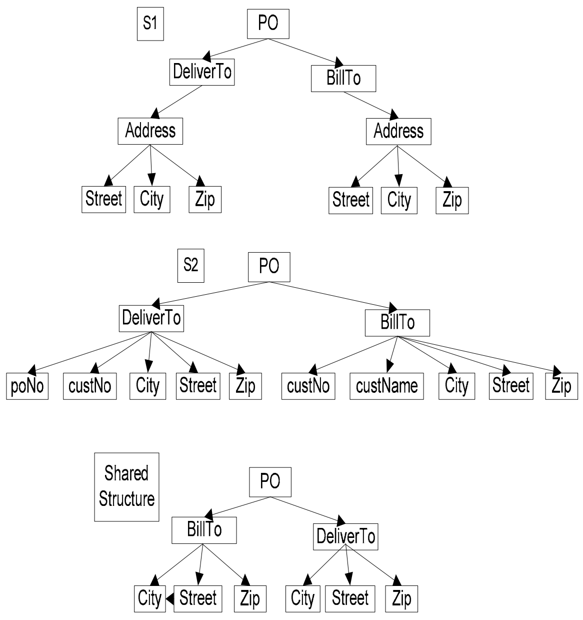

Figure 4. XML schema structures describing different post order document (S1 and S2) and their shared document structure

It is worth noting that it could be possible that some knowledge structures have a few nodes with the same label located deeper in the tree. In this case if embedded subtrees are mined misleading results could be returned since the level of embedding allowed in the subtrees is not limited. To make the large change from mining embedded to induced subtrees runs the risk of missing many other common structures where the level of embedding among the nodes is different. In these cases the maximum level of embedding (Φ) constraint [14] could be used to impose a limit on the allowed level of embedding in the extracted embedded subtrees. The Φ could also be progressively decreased until some differences are resolved. In this scenario where multiple nodes exist with the same label it probably would not provide only one matching document structure as for the induced case, but many common structures of same size could be detected. Which method to adapt is again dependent on the type of knowledge that is being matched as for some applications induced subtrees may be sufficient and the level of embedding can be ignored while for others it is important as it indicates that extra specific information is stored for a particular concept in a document structure. Even if the user is not a domain expert different options can be tried with respect to Φ and this should itself reveal some more detail about the similarities and differences among the compared document structures. Besides our focus on the application to the knowledge matching problem this capability of efficiently finding common structures is believed to be well suitable for general analysis and querying of domain knowledge.

In this section we have demonstrated how a tree mining algorithm could be applied to knowledge matching problem. When mining different knowledge structures it can efficiently find the largest common structure which indicates the shared knowledge of the domain at hand. It is worth noting that, even though the examples used here are quite simple the tree mining algorithms are in general well applicable to large datasets composed of complex tree structures. This was experimentally demonstrated in [13, 14, 17].

5. Conclusions and Future Work

In this paper we have described a way in which the tree mining algorithms can be effectively used for detecting a shared knowledge structure from XML documents describing same domains. This is our preliminary work in the area, and as such the aim was to discuss how tree mining can be appropriately applied to the problem and to demonstrate its great potential in automating the task of knowledge matching. We have used real world XML schemas, and the application of our U3 algorithm for mining of unordered embedded subtrees has indeed shown its capability of detecting the shared document structures. To limit the scope of the work we have made the assumption that the concepts have already been matched, and hence our immediate future work is to find semantically correct matches among concepts through the utilization of tree mining for efficient knowledge structure analysis.

6. References

[1] T.R., Gruber, "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition*, 5 (2), 1993, pp. 199-220.

[2] Fensel, D., Lausen, H., Polleres, A., Bruijn, J.D., Stollberg, M., Roman, D., and Domingue, J., *Enabling Semantic Web Services: The Web Service Modeling Ontology*, Springer-Verlag, Berlin, 2007.

[3] 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, 2004.

[4] Alesso, H.P., and Smith, C.F. *Thinking on the Web : Berners-Lee, Gödel, and Turing*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.

[5] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities", In *Proceedings of the International Semantic Web Conference (ISWC)*, 2002, pp. 333–347.

[6] J. T. L. Wang, D. Shasha, G. Chang, L. Relihan, K. Zhang, and G. Patel, "Structural matching and discovery in document databases, *ACM SIGMOD Int'l Conf. on Management of Data*, 1997, pp. 560-563.

[7] K. Wang, and H. Liu, "Discovering Structural Association of Semistructured Data", *IEEE Transactions on Knowledge and Data Engineering*, 1999.

[8] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change detection in hierarchically structured information", *ACM SIGMOD Int'l Conf. on Management of Data*, 1996, pp. 493–504.

[9] S. Chawathe, "Comparing hierarchical data in external memory"*, Proceedings of the Twentyfifth International Conference on Very Large Data Bases* (1999), p. 90-101

[10] J. T. L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey, "An algorithm for finding the largest approximately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998, 20(8), pp. 889–895.

[11] G. Cobena, S. Abiteboul, and A. Marian, "Detecting changes in XML documents" In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[12] A. Nierman and H. V. Jagadish. "Evaluating Structural Similarity in XML Documents". In *Int'l Workshop on the Web and Databases (WebDB)*, Madison,WI, Jun. 2002.

[13] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. "Approximate XML Joins". In ACM SIGMOD, Madison, WI, Jun. 2002. 175

[14] H. Tan, T.S. Dillon, F. Hadzic, L. Feng, E. Chang, "IMB3-Miner: Mining Induced/Embedded subtrees by constraining the level of embedding" *PAKDD'06,* Singapore, 2006.

[15] F. Hadzic, H. Tan, T.S Dillon, and E. Chang, "U3 – Mining unordered embedded subtrees using model guided candidate generation", Submitted to the *1st ACM Int'l Conf. on Web Search and Data Mining*, San Francisco Bay Area, California, USA, 2008.

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

[17] Y. Chi, S. Nijssen, R.R. Muntz, and J.N. Kok, "Frequent Subtree Mining--An Overview", *Fundamenta Informaticae, Special Issue on Graph and Tree Mining*, vol. 66, No. 1-2, 2005, pp. 161-198.

[18] R. J. Bayardo, R. Agrawal, and D. Gunopulos, "Constraint-based rule mining on large, dense data sets" *Int. Conf. Data Engineering (ICDE'99)*, Sydney, 1999.

[19] F. Hadzic, H. Tan, T.S. Dillon, and E. Chang, "Implications of frequent subtree mining using hybrid support definition", *Data Mining & Information Engineering 2007*, 18-20 June, The New Forest, UK, 2007.

[20] H. Tan, T.S. Dillon, F. Hadzic, E. Chang, and L. Feng, "MB3-Miner: mining eMBedded sub-TREEs using Tree Model Guided candidate generation", *1st Int'l Workshop on Mining Complex Data (MCD'05)*, held in conjunction with ICDM'05, Houston, Texas, USA, 2005.

[21] H. Tan, T.S. Dillon, F. Hadzic, and E. Chang, "Razor: mining distance constrained embedded subtrees" IEEE ICDM 2006 *Workshop on Ontology Mining and Knowledge Discovery from Semistructured documents (MSD 2006)*, 28-22 December, Hong Kong, 2006.

[22] F. Hadzic, H. Tan, and T.S. Dillon, "UNI3 – Efficient Algorithm for Mining Unordered Induced Subtrees Using TMG Candidate Generation" *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007),* Honolulu, Hawaii, 2007.

[23] F. Hadzic, T.S. Dillon, A. Sidhu, E. Chang, and H. Tan, "Mining Substructures in Protein Data", IEEE ICDM 2006 Workshop on Data Mining in Bioinformatics (DMB 2006), 18-22 December, Hong Kong, 2006.

[24] Ontology Matching [www.ontologymatching.org].

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