Knowledge Analysis with Tree Patterns

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

Tree-structured knowledge representations are increasingly being used since the relationships between data objects can be represented in a more meaningful way. A number of tree mining algorithms were developed for mining different subtree types using different parameters. At this point in research it would be useful to discuss what kind of sub-problems can be solved within the current tree mining framework. In this paper we provide a general overview of the development in the area of tree mining and discuss motivations and useful application areas for each development. Implications of using different tree mining parameters and constraints are discussed. Such an overview will be particularly useful for those not so familiar with the area of tree mining as it can reveal useful applications within their domain of interest. It gives guidance as to which type of tree mining will be most useful for their particular application.

1. Introduction

Tree mining algorithms have found many useful applications in areas such as bioinformatics, scientific knowledge management, web mining, XML mining, etc. One of the key issues in the knowledge management field is how to compare and match the knowledge structures obtained from heterogeneous sources. The tree structured knowledge representations can be increasingly found in many biomedical, web and scientific domains, where traditional structured representations would fail to capture the desired semantics and relationships of data objects. This in turn has given rise to the development of new algorithms capable of efficiently extracting information presented in semi-structured form. These are generally known as frequent subtree mining algorithms and the problem can be stated as: given a tree database *Tdb* and minimum support threshold (σ), find all subtrees that occur at least σ times in *Tdb*. The scope of their application usually depends on the assumptions made about the data structure that the algorithm is to be applied on. These assumptions depend upon the domain of interest and what task is to be accomplished in a particular application. 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 into ordered and unordered subtrees. The available support definitions to use are transaction based, occurrence match [30, 25]. and hybrid support [12], and these will be explained in detail later in the paper.

The overview of the frequent subtree mining area presented in [6] is focused on algorithm comparisons and various implementation issues, such as the candidate enumeration approach, frequency counting and the representative structure of the tree. There exist a number of approaches that mine different subtrees using different support definitions and the implication for general knowledge analysis has not been addressed to a great detail. Hence at this point in tree mining research it is desirable to discuss the general implications behind using different types of tree mining algorithms.

In this paper we provide a general overview of the development in the area of frequent subtree mining. A discussion is provided on the motivations and implications for mining different subtree types and using different support definitions. We also discuss the motivation behind imposing some constraints on the embedded subtree type. For each support definition (Section 2), subtree type (Section 3) and constraint (Section4), example scenarios are provided where the use of these particular tree mining parameters will prove useful. We do not

1530-1605/08 $25.00 © 2008 IEEE

perform any experimental comparisons of the listed algorithms since many of the comparisons have already been provided in [6, 25, 11, 13]. This kind of overview will be particularly useful for those not so familiar with the area of tree mining as it can reveal useful applications within their domain of interest. It gives guidance as to which type of tree mining will be most useful for their particular application and it lists the tools available whose performance they can trace in the references. Section 5 lists the available algorithms for mining of the defined subtree types within the current tree mining framework.

2. Support definitions

This section describes the current support definition used within the frequent subtree mining framework. Each subsection starts by providing a definition of the support type and describes the motivation for its use.

2.1. Transaction-based support

To clarify the term transaction when used in the context of tree mining we find the following definition suitable. A transaction is a set of one or more items obtained from a finite item domain, and a dataset is a collection of transactions [4]. Hence, in the context of a tree database, a transaction would correspond to a fragment of the database tree whereby an independent instance is described. When using the transaction based support (*TS*) definition, the support (*σ*) of a subtree *t* in tree database *Tdb* is equal to the number of transactions in *Tdb* that contain at least one occurrence of subtree *t*. If we let the notation $t \prec k$, denote the support of subtree *t* by transaction *k*, then for *TS*, $t \prec k = 1$ whenever *k* contains at least one occurrence of *t*, and 0 otherwise. Suppose that there are N transactions k_1 to k_N of tree in *Tdb*, the transactional support of subtree *t* in *Tdb* is defined as:

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

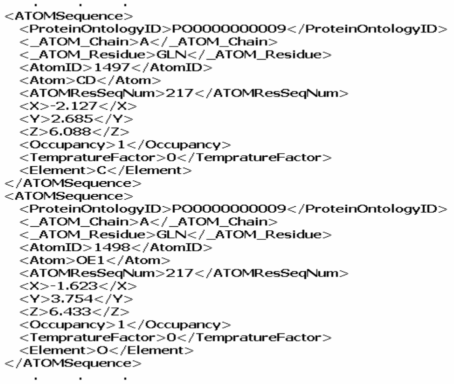
In traditional frequent itemset mining from relational data checking whether an item exists within a transaction is sufficient to determine the traditional support definition. Hence using transactional support would appear to be the obvious choice when moving from relational to XML frequent pattern mining. Furthermore, it is common that in transfer from relational to XML data an instance in relational data is described by one transaction in XML data. This has made transaction-based support the focus of many tree mining works and from the available support definitions it is the simplest one to consider.

2.2. Occurrence match support

Occurrence match support (*OC*) takes the repetition of items in a transaction into account and counts the subtree occurrences in the database as a whole. Hence for *OC*, the support (*σ*) of a subtree *t* in tree database *Tdb* is equal to the total number of occurrences of *t* in all transactions in *Tdb*. Let function g(t,k) denote the total number of occurrences of subtree *t* in transaction *k*. Suppose that there are *N* transactions k_1 to k_N of tree in *Tdb*, the occurrence match support of a subtree *t* in *Tdb* can be defined as:

$$\sum_{i=1}^{N} \mathrm{g(t, ki)}$$

To illustrate the importance of occurrence match support, consider the partial XML representation of protein data displayed in Figure 2. The original dataset describes a protein ontology instance store for Human Prion Proteins in XML format [21]. Protein Ontology (PO) provides a unified vocabulary for capturing declarative knowledge about the protein domain. and classifies that knowledge to allow reasoning. Using the PO format, ATOMSequence labels can be compared easily across PO datasets for distinct protein families to determine sequence and structural similarity among them. Structured ATOMSequence labels, with repetition of Chain, Residue and Atom details can be used to compare a new unknown protein sequence and structure with existing proteins in the PO dataset, which helps users in drug discovery and design. In this case the repetition in the structure of the protein is of considerable importance.



```
   .   .   .
<ATOMSequence>
  <ProteinOntologyID>PO0000000009</ProteinOntologyID>
  <_ATOM_Chain>A</_ATOM_Chain>
  <_ATOM_Residue>GLN</_ATOM_Residue>
  <AtomID>1497</AtomID>
  <Atom>CD</Atom>
  <ATOMResSeqNum>217</ATOMResSeqNum>
  <X>-2.127</X>
  <Y>2.685</Y>
  <Z>6.088</Z>
  <Occupancy>1</Occupancy>
  <TempratureFactor>0</TempratureFactor>
  <Element>C</Element>
</ATOMSequence>
<ATOMSequence>
  <ProteinOntologyID>PO0000000009</ProteinOntologyID>
  <_ATOM_Chain>A</_ATOM_Chain>
  <_ATOM_Residue>GLN</_ATOM_Residue>
  <AtomID>1498</AtomID>
  <Atom>OE1</Atom>
  <ATOMResSeqNum>217</ATOMResSeqNum>
  <X>-1.623</X>
  <Y>3.754</Y>
  <Z>6.433</Z>
  <Occupancy>1</Occupancy>
  <TempratureFactor>0</TempratureFactor>
  <Element>O</Element>
</ATOMSequence>
   .   .   .
```

Figure1. Snapshot of the representation of Human Prion Protein dataset in XML format

Another scenario where occurrence match support may be important is when performing specialized queries on a tree structured database. As an example, consider a publication based library where author information is stored separately in each transaction (eg. Figure 2). A user may be interested in finding out information about the total number of books that were published in a certain year by a certain publisher. To satisfy this query, the repetition of book-year-publisher relations within a transaction will need to be considered. The answer is given by the total number of occurrences of relation book-year-publisher in the whole database. In these scenarios the repetition of items within a transaction is considered important and the knowledge of the number of repetitions provides useful information.

2.3. Hybrid support

As the name implies for hybrid support (*HS*) definition we are combining *TS* with *OC* support. The support of a subtree *t* is denoted by ‘$x|y$’, where ‘*x*’ denotes the number of transactions that support subtree *t*, and *y* denotes the least number of times that *t* has occurred in those *x* transactions. *HS* provides extra information about the intra-transactional occurrences of a subtree. Hence, using *HS* 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. To determine if a subtree is frequent the transaction support definition from Section 2.1 can be used with the difference that $t \prec k = 1$ iff $g(t,k) \geq y$, and $t \prec k = 0$, otherwise.

In certain applications the number of times a subtree occurs within a transaction is of interest. Example applications could be taken from many web information systems applications, where specialized queries on tree structured databases commonly take place. As an example, consider a publications database where author information may be separately stored in each transaction, as shown on left of Figure 2.

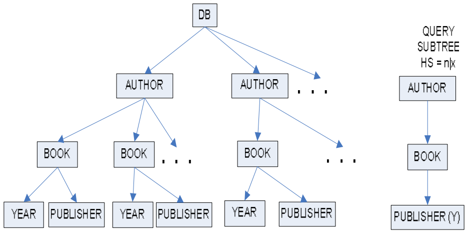


Figure 2. Example publications database DB (left) and query subtree (right)

A user may be interested in finding ‘*n*’ number of authors that have published at least ‘*x*’ books with publisher ‘*Y*’. To satisfy this query, the repetition of author-book-publisher(Y) relation within a transaction will need to be considered. This query is displayed on the right of Figure 2 and the *HS* threshold would be equal to $n|x$ since author information is stored separately in each transaction, and we want to find at least ‘*x*’ books with publisher ‘*Y*’. In these scenarios HS would provide useful information automatically without any post processing which would need to occur if either OC or TS supports were used.

3. Subtree types

This section starts by providing a quick overview of some basic tree concepts. It then discusses the common subtree types considered within the tree mining area. Each type is separately explained and the differences in implication and uses within a particular application are discussed.

A tree can be denoted as *T(V,L,E)*, where (1) *V* is the set of vertices or nodes; (2) *L* is the set of labels of vertices, for any vertex v∈V, L(v) is the label of v; and (3) $E = \{(x,y)|\ x,y \in V\}$ 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* which are defined as its successors. 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 *vi* to *vj*, is defined as the finite sequence of edges that connects *vi* to *vj*. 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.

3.1. Ordered Induced Subtrees

A tree *T’(V’, L’, E’)* is an ordered induced subtree of a *tree T (V, L, E*) iff (1) $V' \subseteq V$, (2) $E' \subseteq E$, (3) $L' \subseteq L$ and $L'(v) = L(v)$, (4) $\forall v' \in V'$, $\forall v \in V$, *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 its removal doesn't create a forest in *T*.

These subtrees are the simplest among the ones considered in this paper, and the main implication is that the parent-child relationships must remain the same as in the original tree. As such they have been extensively used and are the most common type considered by the tree mining algorithms. A reason for this would be explained as that when transforming from relational data into tree structured data the derived tree structure itself has only a few levels and hence all the meaningful information is effectively represented by an induced subtree.

As an example, consider again the tree representation of a publication database DB from Figure 2. If all the frequent ordered induced subtrees are extracted then all the possible queries could be answered with respect to the minimum support threshold used. The example query subtree on left of Figure 2 is of induced subtree type and in fact all particular queries could be answered with subtrees of an induced type. Hence, an algorithm for mining induced subtrees would be sufficient for this application.

The popularity of mining induced subtrees could be partly because they are the most easily detected subtree type by a human observer and it is naturally to think of subtrees in this sense. However when certain application indicated the need for extending the parent-child relationship between the nodes to ancestor-descendant the focus has shifted toward the mining of embedded subtrees. This allows one to detect information embedded deeply within the tree structure and the importance of this is discussed next. Induced subtree are a subset of embedded subtrees and hence mining embedded subtree adds much more complexity to the task.

3.2. Ordered Embedded Subtrees

A tree $T'(V', L', E')$ is an *embedded subtree* of a tree $T(V, L, E)$ 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 $L'(v)=L(v)$.

When structurally rich information is represented using XML it is quite common that the document is organized into several levels and each transaction can be many levels deep. In these cases many query trees posed on an XML document may be of embedded subtree type since there will be many more ancestor-descendant relationships present among the nodes. Furthermore, certain concepts may be represented in a more specific/general way in certain documents.

When trying to find the common structures among knowledge representations if embedded subtrees are mined we are allowing structures to be considered similar even if they occur at different levels in the tree. In an embedded subtree the relationship is not limited to parent-child and hence by allowing ancestor-descendant relationships enables the extraction of more sub-structures where the levels of embeddings between the nodes are not limited to one and can be different. For example if in knowledge representation 'A' the level of embedding between the nodes 'a' and 'b' representing some domain concepts is much larger than the level of embedding between the nodes representing the same concepts in knowledge representation 'B', then 'A' stores more specific knowledge about the concept represented by node 'a'. Since it is common that knowledge representations could differ in the amount of specific knowledge stored, mining of embedded subtrees is more suitable for general knowledge comparison.

The representations used in our example from Figure 3 correspond to the knowledge models used for classification purposes, and hence the extra specific knowledge corresponds to the additional number of attribute constraints used for further separation of class values. The knowledge models were obtained using data mining tools on the publicly available datasets describing the 1984 United States Congressional Voting Records Database [5]. We used different subsets of data and a feature selection approach [9] in order to mimic a real world scenario when different organizations collect their own sets of data and find different features to be relevant. Each of the knowledge models is represented as a separate subtree (transaction) within the tree database and transaction based support is used to extract the largest embedded subtree that occurs in each transaction. If we have k different models than a subtree will only be considered frequent if it occurs in all k models. Hence, the transaction based support was used with the threshold of 2.

Please note that since in this domain we are dealing with a binary classification problem and the knowledge model is represented by a binary decision tree, we filter out any of the subtree patterns where any of the nodes has a degree larger than 2. These patterns would be meaningless as there would be three class values distinguished by two attribute constraints. They do not indicate substructures that imply true shared knowledge since their classificatory purpose has been lost.

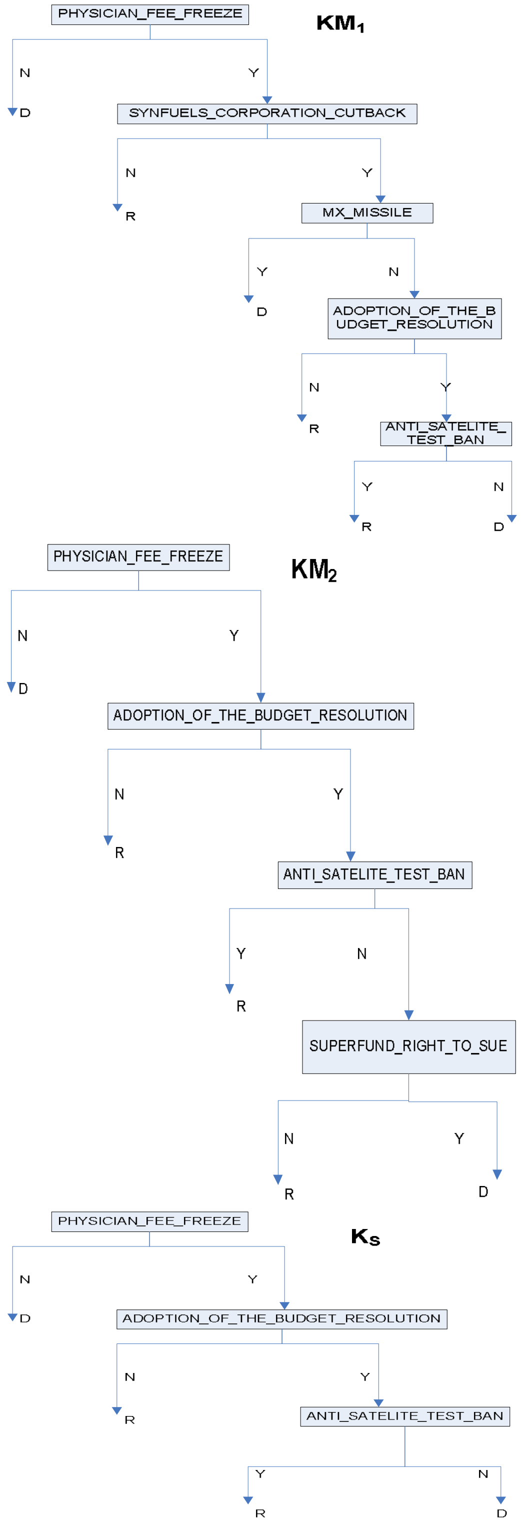


Figure 3. Different knowledge models (KM1, KM2, KM3) and the shared conceptualization (common embedded subtree KS)

The largest frequent ordered embedded subtree (K_S) would display the largest common structure between the KMs and is displayed last in Figure 3. By comparing K_S with the knowledge models we can see that it was necessary to mine embedded subtrees in order to detect the common knowledge structure. If induced subtrees were mined only the root node and its value nodes would be considered frequent. KM_2 only differs to the Ks in the sense that it has one extra attribute after the last node which splits the class values further into 'D' and 'R', rather than generalizing it to 'R'. The knowledge model KM_1 stores two additional attributes after the first attribute node. We can say that KM_2 is only more general than KM_1 in between the first and second attribute while it is more specific at the last node.

Merging of knowledge structures has been of interest for a long time and many useful applications can be found in e-commerce, enterprise application integration and the general management of scientific knowledge. Another related area where merging of tree structured knowledge models will be useful is in a classification ensemble which is a multi-classifier system where each classifier is developed for the same domain problem [18]. The shared knowledge structure would indicate the general knowledge of the domain. The new classifier is then expected to have better generalization capability and hence be more accurate in classifying future unseen data objects. As shown in Figure 3 mining of embedding subtrees can be useful for this task.

As already noticed in some of our examples mining of ordered subtrees is useful for queries performed on a single database where the sibling node order is already known and hence the order restriction can be placed on the subtree. Ordered subtrees have many applications in molecular biology [19] protein structure analysis [10], natural language processing [16] etc. A general remark for these applications is that the left-to-right order among sibling nodes is commonly fixed and known beforehand.

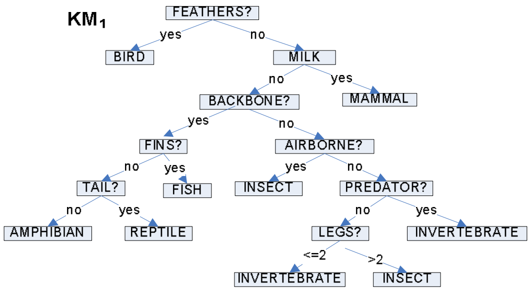
Even though in this example it happened to be that the common subtree was ordered in the same order among all the knowledge models, the order among any sibling nodes could be exchanged and the information content would still be the same. When comparing conceptual knowledge models, different sibling node order usually does not make the structure match any less. Hence, the order of sibling nodes does not need to be preserved in which case we are talking about unordered subtrees which are discussed next.

3.3. Unordered Subtrees

The definitions of unordered induced and embedded subtrees would be the same as those in Sections 3.1 and 3.2., respectively, with the only difference that the condition (5) can be relaxed so that the order among sibling nodes does not need to be preserved.

In many cases the order among the sibling-nodes is considered irrelevant to the task and is often not available. If one is interested in comparing knowledge structures among different documents it is very common that the order of sibling nodes may differ but the information contained in the structure is essentially the same. In these cases mining of unordered subtrees is much more suitable as a user can pose queries and does not have to worry about the order. All matching sub-structures will be returned with the difference being that the order of sibling nodes is not used as an additional candidate grouping criterion. This makes the task more complex since a group of ordered subtrees now maps to only one unordered candidate, and hence determining tree isomorphism is harder [27]. However, for most of knowledge management tasks the unordered mining is starting to have better applications.

To illustrate this consider the two example knowledge models represented in Figure 4, which were learned from the publicly available 'zoo' dataset [5]. Even though at first sight they appear as different models, when the classification rule underlying the structure is examined it implies the same classification information. Hence the order of sibling nodes did not play any role in the information content of the structure. Furthermore, since the user cannot be sure of the order of sibling-nodes many queries posed in such domains would be of the unordered subtree type.



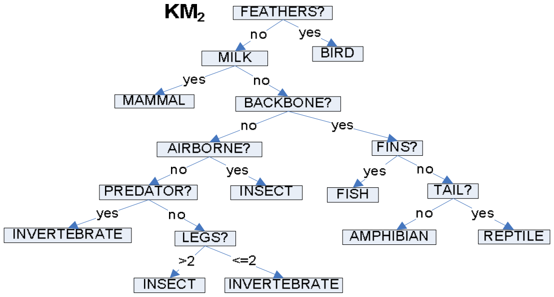


Figure 4. Two knowledge models (KM_1 and KM_2) different in sibling node order but representing same information

4. Constrained embedded subtrees

This section overviews some additional constraints that can be imposed on an embedded subtree. We narrow our focus on the actual constraints where the general problem of frequent subtree mining is not modified but the traditional definition of an embedded subtree has been changed according to the added constraint. These constraints have been implemented within the general TMG framework [23, 25]. Each constraint is discussed with some application areas where its enforcement would be useful.

4.1. Level of embedding constraint

If $T'(V', L', E')$ is an embedded subtree of T, and there is a path between two nodes p and q, the level of embedding (δ) is defined as the length of the path between p and q, where $p \in V'$ and $q \in V'$, and p and q form an ancestor-descendant relationship. 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 Tdb and Φ, then any embedded subtree to be generated will have the maximum length of a path between any two ancestor-descendant nodes in T equal to Φ. In this regard, we could define induced subtree Ti 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.

The level of embedding constraint was discussed in [25] as a way of tackling the complexity of mining embedded subtrees. By restricting the level of embedding the number of possible candidates can be reduced and less time and space is required to complete the task [25].

As was shown in the previous section there is some difference in mining induced and embedded subtrees. In certain scenarios, allowing the extra embeddings can result in unnecessary and misleading information but in other cases it proves useful as it detects common structures besides the difference in concept granularity. Moving from embedded subtrees where the allowed level of embedding is equal to the depth of the tree to induced where the level is limited to one, is a rather large jump and many useful patterns could be missed. Hence in this sense the maximum level of embedding constraint may prove useful as one could progressively decrease the level of embedding which can give clues about some general difference among the knowledge representations compared.

4.2. Distance-constrained embedded subtrees

A tree *T'(V', L', E')* is an ordered distance-constrained embedded subtree of a tree *T(V, L, E)* if it satisfies all the properties of an embedded subtrees (section 3.2), and. $v' \in V'$ there is an integer stored indicating the level of embedding (δ) in tree *T* between *v'* and the root node of *T'*.

Essentially this constraint allows the subtrees to be distinguished based upon their node distance relative to the root node. Since many embedded subtrees can form one candidate this adds more granularity since each of those subtrees are now grouped as different candidates when the distance among the nodes is different. Hence mining distance constrained subtree is more expensive in terms of space and time required [23].

For some applications it is important to note this difference so that specialized queries could be posed. As an illustrative example, consider Figure 5 where we display two example trees which indicate a part of the ancestor family tree from two ill patients (the examples were extracted from an image of a disease family tree obtained from [1]). Such information is used for linkage analysis of an illness by performing gene testing which can provide information about one having a disease related gene mutation. When looking for a disease gene scientists often start by studying DNA samples from family members over several generations that have a number of relatives who have developed an illness [1].

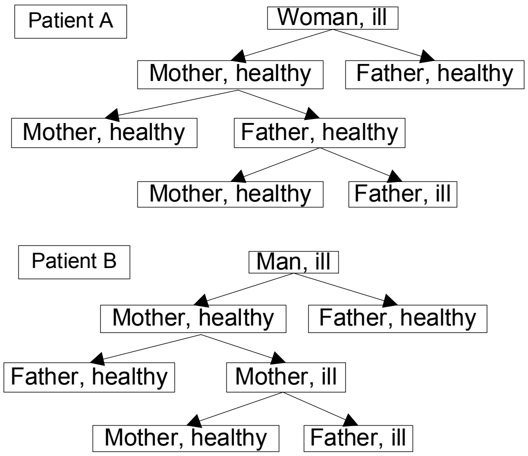


Figure 5. Example representation of two ill patients (A and B) with common ancestors

For example a scientist may want to discover how many ill relatives an ill patient has and to discover the number of generations that separates them. Using the traditional embedded subtree definition we could only extract information about the number of ill relatives but could not have the information about the number of generations that separate the patient and the relatives that have a common disease. This is because the traditional embedded subtree definition does not have this kind of expressive capability. In contrast, by utilizing the distance-constrained embedded subtrees, we can find out exactly how many generations they are separated by, through inspecting the distance information stored between the nodes. From the Figure 5 patient A has only one diseased ancestor and it is her grand-grand father, while patient B has two diseased ancestors, a grand-mother and grand-grand father.

Even though we do not have such an example in a figure, it is worth noting that it could well be the case that an ill patient will have two ancestors of the same gender that have the illness. In this case the traditional embedded subtree definition would group these subtree occurrences as one candidate and indicate wrongly that there is only one ancestor with a disease. On the other hand, by mining distance-constrained embedded subtrees, both occurrences will be considered as separate entities due to the difference in the distance to the root node which is used as an additional candidate grouping criterion.

This notion of distance constrained embedded tree mining will have important applications in biological sequences, web information systems and conceptual model analysis. However it is important to note here that by no means, we are claiming that distance constrained embedded subtrees should replace the mining of embedded subtrees. Embedded subtrees without any constraint has still many important applications an one of them was ahown in Section 3.2.

5. Frequent subtree mining algorithms

This section gives an overview some of the algorithms developed for the problem of frequent subtree mining. Initially the enumeration problem for tree-structured data is discussed which is one of the main performance bottlenecks. The enumeration method employed is commonly the determining factor for the efficiency of an algorithm since if unnecessary candidates are generated which then have to be checked for frequency will result in a decrease in space and time efficiency.

With more complex relationships inherent in tree-structured data, enumeration of subtrees becomes more challenging than enumeration of itemsets from structured data. The two commonly used enumeration strategies used for tree-structured data are enumeration by extension and join [6]. Other reported technique to mine frequent subtrees is to utilize the Pattern-Growth method [28], which is an extension of the FP-growth [14] method for structured data. Pattern-Growth does not perform level-by-level candidate enumeration as is normally done by Apriori-based approaches and it is the only known method to obtain frequent subtrees without candidate generation process. 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. This idea was extended to the Tree Model Guided (TMG) candidate generation [22, 24] which utilizes the underlying tree-structure of the document for efficient candidate generation. This non-redundant systematic enumeration technique ensures that all the candidate subtrees generated are valid, in the sense that they conform to the structural aspects of the document.

In regards to mining of induced ordered subtrees some of the available algorithms are: FREQT [2], AMIOT [15], IMB3-Miner[25], and PrefixTreeISpan [32].

FREQT [2] algorithm uses the rightmost expansion technique and an optimized pruning technique. AMIOT [15], uses the 'right-and-left tree join' method to enumerate only those candidates that have a high probability of being frequent. IMB3-Miner [25] is one of the algorithms developed within the TMG candidate enumeration framework and it utilizes the level of embedding constraint to mine induced subtrees. More recently the algorithm PrefixTreeISpan [32] extends the notations of prefix and post-fix from sequential mining and uses the idea of divide and conquer to find the complete set of frequent patterns. The main idea is to examine the prefix-tree subtrees and project their corresponding postfix-forests [32] into the projected database.

Some of the existing algorithms capable of extracting frequent ordered embedded subtrees are TreeMiner [30], XSpanner [28], X3-Miner [22], MB3-Miner [24], and IMB3Miner [25] The TreeMiner [30] algorithm uses a data structure called the vertical scope-list and utilizes the join approach for candidate generation. TreeMiner consists of two versions, one which adopts a depth first search approach and the other which uses the breadth-first approach for candidate generation and counting. XSpanner [28] utilizes the Pattern-Growth method for candidate generation. X3-Miner was where the idea of TMG was first introduced and the algorithm was mines XML documents and returns XML sub-patterns with label and value information. It was not optimized very well for the problem and it had some time efficiency issues. It was then extended to MB3-Miner [24] which introduced an efficient representation of a tree structure called Embedding List [24] so that the TMG approach can be efficiently implemented, and this resulted in great performance increase. In [24] authors provided a TMG mathematical model for estimating the worst case complexity of enumerating all embedded subtrees. This motivated the strategy of tackling the complexity of mining embedded subtrees by introducing the Level of Embedding [25] constraint. Thus, when it is too costly to mine all embedded subtrees, one can decrease the level of embedding constraint gradually down to 1, from which all the obtained subtrees are induced. From the application perspective, the work presented in [10] demonstrates the potential of the algorithms for mining ordered subtrees in providing interesting information when applied to tree structured biological data.

For the problem of extracting all frequent unordered induced subtrees some of the existing algorithms are: Unot [2], RootedTreeMiner [8], HybridTreeMiner [7], the method presented by Nijssen and Kok in [17] and UNI3 [11].

The uNot algorithm [2] uses a reverse search technique for incremental computation of unordered subtree occurrences. Breadth-first canonical form (BFCF) and depth-first canonical form (DFCF) for labeled rooted unordered trees have been presented in [8]. In the same work the authors proposed two algorithms: RootedTreeMiner which is the authors' re-implementation of uNot, a vertical mining algorithm based upon BFCF and FreeTreeMiner, based on extension of DFCF for discovering labeled free trees. As an extension to the work, HybridTreeMiner [7] is an efficient algorithm that systematically enumerates subtrees by traversing an enumeration tree which is defined based upon the BFCF for unordered subtrees. Nijssen & Kok [17] present a bottom-up strategy for determining the frequency of subtrees, and argue that the complexity of enumerating unordered trees as opposed to ordered is not much higher. All these approaches only consider the transactional support definition while the UNI3 algorithm [11] which is an extension of the TMG approach for the problem of unordered subtree mining, can use any of the three discussed support definitions.

There are not many algorithms available that mine unordered embedded subtrees. TreeFinder [26] was the first attempt which uses inductive-logic programming for enumerating all candidate subtrees, but which in the process can miss many frequent subtrees. SLEUTH [31] was the first complete approach proposed and it enumerates unordered subtrees by using unordered scope-list joins via the descendant and cousin tests. While it can handle occurrence match support its main focus is on transaction based support and there are some limitations for occurrence match support as demonstrated in [13]. As an extension to the TMG framework for mining of unordered embedded subtrees, the U3 algorithm [13] was proposed. The previously used Embedding List [11] was replaced by a compressed version called Recursive List [13] that reduces the memory space consumption and has additional functionalities. U3 has the capability of restricting the level of embedding allowed in the extracted subtrees and can use any of the three support definitions. Unordered tree mining has been successfully applied in [20] for the analysis of phylogenetic databases.

Besides the Razor algorithm [23], we are not aware of any other tree mining algorithms that take the distance among the nodes into account when extracting all frequent embedded subtrees.

A final note from the overviewed approaches is that the general TMG approach for candidate enumeration has been extended for all the sub-problems of tree mining discussed in this paper. Any of the subtree types can be mined and all the support definitions can be used. Furthermore this adaptability for a variety of problems was never at a cost of a noticeable reduction in efficiency since the extended algorithms in most of cases performed better to the state-of-the-art algorithms at that time. This was experimentally demonstrated in [24, 25, 11, 13]. Furthermore the integration of capability for mining constrained subtrees can prove useful for more specific queries on knowledge representations, as was shown in the previous section.

6. Conclusions and future work

In this paper we have provided a general overview of our developments in the area of tree mining. The motivation behind each development was illustrated with example scenarios. An overview of possible tree mining parameters that can be used within the current tree mining framework was provided. These parameters include the support definitions, types of subtrees that can be mined, and types of constraints that can be imposed on the subtrees. Possible application areas and the implications of using different tree mining parameters were discussed, and the available algorithms using those specific parameters were listed. Our future work involves the integration of distance constraint for unordered embedded subtrees and the application of tree mining to the problems of automatic ontology learning and matching.

7. References

[1] Access Excellence @ the national health museum "Understanding gene testing: What does a predictive gene tell you" Retrieved June 15, 2007, from http://www.accessexcellence.org/AE/AEPC/NIH/gene14.html.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, "Efficient substructure discovery from large semi-structured data", *In Proc. of the 2nd SIAM Int'l Conf. on Data Mining (SDM'02)*, 2002, pp. 158–174.

[3] T. Asai, H. Arimura, T. Uno, and S. Nakano, "Discovering Frequent Substructures in Large Unordered Trees", *In Proc. of the 6th Int'l Conf. on Discovery Science*, 2003.

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

[5] C. Blake, E. Keogh, and C.J. Merz, "UCI Repository of Machine Learning Databases". [http://www.ics.uci.edu/~mlearn/MLRepository.html].

[6] 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.

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

[8] 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.

[9] F. Hadzic, and T.S. Dillon, "Using the Symmetrical Tau (τ) Criterion for Feature Selection in Decision Tree and Neural Network Learning", *Proc. of the 2nd Workshop on Feature Selection for Data Mining*, in conj. with 2006 SIAM International Conference on Data Mining, Bethesda, 2006.

[10] 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.

[11] F. Hadzic, H. Tan, T.S. Dillon, and E. Chang, "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.

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

[13] F. Hadzic, H. Tan, T.S Dillon, and E. Chang, "U3 – Mining unordered embedded subtrees using model guided candidate generation", Submitted to the *6th IEEE Int'l Conf. on Data Mining*, Omaha, NE, 2007.

[14] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", *Data Mining and Knowledge Discovery*, Vol. 8, 2004, pp. 52-87.

[15] S. Hido, and H. Kawano, "AMIOT: Induced Ordered Tree Mining in Tree-structured Databases", In Proc. of the 5th IEEE Int'l Conf. on Data Mining (ICDM'05), Houston, Texas, USA, 2005, 170-177.

[16] M. S. Neff, J.B. Roy, and A.R. Omneya, "Creating and querying hierarchical lexical data bases", In Proc. of the 2nd Applied Association for Computational Linguistics Conference, Austin, Texas, 1998, pp. 84-99.

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

[18] A. Prodromidis, P. Chan, and S. Stolfo, "Metalearning in distributed data mining systems: Issues and approaches", *Proc. of the 3rd In'l Conf. on Knowledge Discovery and Data Mining*, 1997

[19] B. Shapiro, and K. Zhang, "Comparing multiple RNA secondary structures using tree comparisons", *Computer Applications in Biosciences*, 1990, pp. 309-318.

[20] D. Shasha, J.T.L. Wang, and S. Zhang, "Unordered Tree Mining with Applications to Phylogeny" *20th Int'l Conf. Data Engineering*, 2004.

[21] A.S. Sidhu, T.S. Dillon, E. Chang, and B.S. Sidhu, "Protein ontology: vocabulary for protein data" 3rd *Int'l IEEE on Conf. on Information Technology and Applications*, (IEEE ICITA 2005), Sydney, 2005.

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

[23] H. Tan, T.S. Dillon, F. Hadzic, and E. Chang, "Razor: mining distance constrained embedded subtrees", *Workshop on Ontology Mining and Knowledge Discovery from Semistructured documents (MSD 2006)*, in conj. with the 2006 Int'l Conf. on Data Mining, 28-22 December, Hong Kong, 2006.

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

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

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

[27] Valentine, G. *Algorithms on Trees and Graphs*, Springer-Verlag, Berlin, 2002.

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

[29] L.H. Yang, M.L. Lee, and W. Hsu, "Efficient mining of XML query patterns for caching", In *Proc. of the 29th Int'l Very Large Data Bases (VLDB) Conf.*, Berlin, Germany, 69-80.

[30] 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.

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

[9] Y. Zhang, W.N. Street, and S. Burer, "Sharing Classifiers among Ensembles from Related Problem Domains", *Proc. of the 5th IEEE Int'l Conf. on Data Mining*, 2005.

[32] L. Zhou, Y. Lu, H. Zang, R. Hu, C. Zhou, "Mining Frequent Induced Subtrees by Prefix-Tree-Projected Pattern Growth", *Proceedings of the 7th International Conference on Web-Age Information Management Workshops (WAIMW'06)*, 2006.