Subtree Testing and Closed Tree Mining Through Natural Representations *

José L. Balcázar, Albert Bifet and Antoni Lozano
Universitat Politècnica de Catalunya
{balqui,abifet,antoni}@lsi.upc.edu

Abstract

Several classical schemes exist to represent trees as strings over a fixed alphabet; these are useful in many algorithmic and conceptual studies. Our previous work has proposed a representation of unranked trees as strings over a countable alphabet, and has shown how this representation is useful for canonizing unordered trees and for mining closed frequent trees, whether ordered or unordered. Here we propose a similar, simpler alternative and adapt some basic algorithmics to it; then we show empirical evidence of the usefulness of this representation for mining frequent closed unordered trees on real-life data.

1 Introduction

Undisputably tree-structured representations are a main key idea pervading all of Computer Science; they often represent an optimal compromise between the conceptual simplicity and processing efficiency of strings and the harder but much richer knowledge representations based on graphs.

There have been efforts in moving towards closure-based mining on structured data, particularly sequences, trees and graphs; see the survey [4] and the references there. One of the differences with closed itemset mining stems from the fact that the set theoretic intersection no longer applies, and whereas the intersection of sets is a set, the intersection of two sequences or two trees is not necessarily one sequence or one tree [2].

Mining frequent trees is nowadays an important task, with broad applications including chemical informatics, computer vision, text retrieval, bioinformatics, and Web analysis. Our main interest is related to closed trees since they give the same information as the set of all frequent trees in less space. Chi et al. proposed CMTreeMiner [5], the first algorithm to discover all closed and maximal frequent labeled induced subtrees without first discovering all frequent subtrees. Arimura and Uno [1] considered closed mining in attribute trees, which is a subclass of labeled ordered trees and can also be regarded as a fragment of description logic with functional roles only. These attribute trees are defined using a relaxed tree inclusion. Termier et al. [7] considered the frequent closed tree discovery problem for a class of trees with the same constraint as attribute trees.

In our recent work [3], we have developed algorithms for finding frequent closed trees, by combining ideas from CloseGraph [10] with a sequential implementation of unranked trees; the corresponding sequences employ natural numbers as alphabet letters, and the value of the corresponding number indicates depth, so that in this way the bidimensional information of trees can be reduced to single-dimensional with good results. Our focus there, and in this paper as well, is on unlabeled induced rooted trees, thus our relevant information is the root and the link structure. The appropriate notion of subtree, so-called top-down subtree, preserves root and links. We develop a similar but simpler sequential representation and the corresponding algorithms for subtree testing. Using these representations and operations as basis of our tree mining algorithms, we describe empirical evaluations that suggest them to be particularly efficient in mining real-life data. (Please note: due to the space restrictions, proofs are omitted from this paper; similar proofs of similar facts, however, can be found in [3].)

2 Preliminaries

Our *trees* will be rooted, unranked trees, *ordered* when the children of any node form a sequence of siblings, or *unordered*, in which they form a set of siblings. The set of all trees will be denoted with $\mathcal{T}$. We say that $t_1, \ldots, t_k$ are the *components* of tree t if t is made of a node (the root) joined to the roots of all the t_i's. A tree t' is a *top-down subtree* (or simply a *subtree*) of a tree t (written $t' \preceq t$) if t' is a connected subgraph of t which contains the root of t. In

*Partially supported by the 6th Framework Program of EU through the integrated project DELIS (#001907), by the EU PASCAL Network of Excellence, IST-2002-506778, by the MEC TIN2005-08832-C03-03 (MOISES-BAR), MCYT TIN2004-07925-C03-02 (TRANGRAM), and CICYT TIN2004-04343 (iDEAS) projects.

the ordered case, the order of the existing children of each node must be additionnally preserved.

Given a finite dataset $\mathcal{D}$ of transactions, where each transaction $s \in \mathcal{D}$ is an unlabeled rooted tree, the *support* of the tree t is the number of transactions where t is a subtree. A subtree t is called *frequent* if its support is greater than or equal to a given threshold min_sup. We define a frequent tree t to be *closed* if none of its proper supertrees has the same support as it has. Generally, there are much fewer closed trees than frequent ones. Given two trees, a common subtree is a tree that is subtree of both; it is a maximal common subtree if it is not a subtree of any other common subtree.

From here on, the *intersection* of a set of trees is the set of all maximal common subtrees of the trees in the set. It is proved in [2] that the size of the intersection of two trees can be exponential in their joint sizes; we must note, though, that all the preliminary experiments suggest that intersection sets of cardinality beyond 1 hardly ever arise.

3 Natural Representations

We propose here a specific data structure to implement trees that will provide us with a particularly streamlined implementation of the closure-based mining algorithms. It is a rather simplified version of the one used in [3]. The following operations will be handy for the definition:

Definition 1. *Given two sequences of natural numbers x, y, we represent by $x \cdot y$ the sequence obtained as concatenation of x and y, by $x + i$ the sequence obtained adding i to each component of x, and by x^+ the sequence $x + 1$.*

Definition 2. *A* natural sequence *is a sequence $(x_1, \ldots, x_n)$ of natural numbers such that $x_1 = 0$ and each subsequent number x_{i+1} belongs to the range $1 \leq x_{i+1} \leq x_i + 1$.*

For example, $x = (0, 1, 2, 3, 1, 2)$ is a natural sequence that satisfies $|x| = 6$ or $x = (0) \cdot (0, 1, 2)^+ \cdot (0, 1)^+$. Now, we are ready to represent trees by means of natural sequences.

Definition 3. *We define a function $\langle \cdot \rangle$ from the set of ordered trees to the set of natural sequences as follows. Let t be an ordered tree. If t is a single node, then $\langle t \rangle = (0)$. Otherwise, if t is composed of the trees $t_1, \ldots, t_k$ joined to a common root r (where the ordering $t_1, \ldots, t_k$ is the same of the children of r), then*

$$\langle t \rangle = (0) \cdot \langle t_1 \rangle^+ \cdot \langle t_2 \rangle^+ \cdot \cdots \cdot \langle t_k \rangle^+$$

Here we will say that $\langle t \rangle$ is the natural representation of t.

Note the role of the previous definition:

Proposition 4. *Natural sequences are exactly the sequences of the form $\langle t \rangle$ for ordered, unranked trees t.*

That is, our encoding is a bijection between the ordered trees and the natural sequences. This encoding $\langle t \rangle$ basically corresponds to a preorder traversal of t, where each number of the sequence represents the depth of the current node in the traversal.

The essence of this representation is that, by recursively extending a sequence that starts with a single zero, adding at the end one more number in all valid manners, we traverse once each tree in the whole space of all unranked ordered trees. The argument is very similar to the one presented in [3].

3.1 Subtree Testing

Subtree testing of two ordered trees can be obtained by performing a simultaneous preorder traversal of the two trees [8]. This algorithm can be implemented on natural representations as shown in Figure 1. There, pos_t traverses sequentially the natural representation of tree t and pos_{st} similarly traverses the purported subtree st. The natural number found in the natural representation at position pos_t is exactly depth(t, pos_t). We assume that the sequences of natural numbers include some sort of end marker END_OF_TREE.

Suppose, we are given the trees st and t, and we would like to know if st is a subtree of t. Our method begins visiting the first node in tree t and the first node in tree st. While we are not visiting any tree end,

- If the depth of a tree t node is greater than the depth of tree st node then we visit the next node in tree t
- If the depth of a tree st node is greater than the depth of a tree t node then we backtrack to the last node in tree st that has the same depth as tree t node
- If the depth of the two nodes are equal then we visit the next node in tree t and the next node in tree st

If we reach the end node of tree st, then tree st is a subtree of tree t.

An incremental version of this algorithm follows easily, as it is explained in next section.

We can test if an unordered tree r is subtree of an unordered tree t by reducing the problem to bipartite matching. Figure 2 shows this algorithm.

Suppose we are given the trees r and t, whose components are $r_1, \ldots, r_n$ and $t_1, \ldots, t_k$, respectively. If $n > k$ or r has more nodes than t, then r cannot be a subtree of t. We recursively build a bipartite graph where the vertices represent the child trees of the trees and the edges the relationship "is subtree" between vertices. The function BIPARTITEMATCHING returns true if it exists a solution for this maximum bipartite matching problem. It takes time $O(n_r n_t^{1.5})$([8]), where n_r and n_t are the number of nodes

ORDERED_SUBTREE(st, t)

Input: A tree st, a tree t.
Output: **true** if st is a subtree of t.

```
1  pos_st = 0
2  pos_t = 0
3  while pos_st ≠ END_OF_TREE
4        do if depth(st, pos_st) > depth(t, pos_t)
5              do while depth(st, pos_st) ≠ depth(t, pos_t)
6                    do pos_st = pos_st − 1
7           if depth(st, pos_st) = depth(t, pos_t)
8              do pos_st = pos_st + 1
9           pos_t = pos_t + 1
10 return pos_st = END_OF_TREE
```

Figure 1. The Ordered Subtree test algorithm

of r and t, respectively. If BIPARTITEMATCHING returns true then we conclude that r is a subtree of t.

To speed up this algorithm, we store the computation results of the algorithm in a dictionary D, and we try to re-use these computations at the beginning of the algorithm.

4 Mining Frequent Closed Trees

The essentials of the mining algorithms, which in turn are based on gSpan [9] and CloseGraph [10], were described in [3]. The natural representations introduced here provide substantially more streamlined algorithms through better use of memory and the rather fast subtree testing described in the previous sections.

Figure 3 shows the gSpan-based algorithm for the ordered case, which is as follows: beginning with a tree of single node, it calls recursively the FOS_MINING algorithm doing one-step extensions and checking that they are still frequent.

Since we represent trees by natural representations, we can speed up these algorithms, using an incremental version of the subtree ordered test algorithm explained before, reusing the node positions we reach at the end of the algorithm. If $st1$ is a tree extended from st in one step adding a node, we can start ORDERED_SUBTREE$(st1, t)$ proceeding from where ORDERED_SUBTREE(st, t) ended. So, we only need to store and reuse the positions pos_t and pos_{st} at the end of the algorithm. This incremental method is shown in Figure 4. Note that ORDERED_SUBTREE can be seen as a call to INCREMENTAL_ORDERED_SUBTREE with pos_{st} and pos_t initialized to zero.

In unordered trees, the children of a given node form sets of siblings instead of sequences of siblings. Therefore, or-

UNORDERED_SUBTREE(r, t)

Input: A tree r, a tree t.
Output: **true** if r is a subtree of t.

```
1  if D(r, t) exists
2     then Return D(r, t)
3  if (SIZE(r) > SIZE(t)
   or #COMPONENTS(r) > #COMPONENTS(t))
4     then Return false
5  if (r = •)
6     then Return true
7  graph ← {}
8  for each s_r in SUBCOMPONENTS(r)
9        do for each s_t in SUBCOMPONENTS(t)
10             do if (UNORDERED_SUBTREE(s_r, s_t))
11                   then insert(graph, edge(s_r, s_t))
12 if BIPARTITEMATCHING(graph)
13    then D(r, t) ← true
14    else D(r, t) ← false
15 return D(r, t)
```

Figure 2. The Unordered Subtree test algorithm

dered trees that only differ in permutations of the ordering of siblings are to be considered the same unordered tree. We select one of them to act as canonical representative of all the ordered trees corresponding to the same unordered tree:

Definition 5. *Let t be an unordered tree, and let $t_1, \ldots, t_n$ be all the ordered trees obtained from t by ordering in all possible ways all the sets of siblings of t. The* canonical representative *of t is the ordered tree t_0 whose natural representation is maximal (according to lexicographic ordering) among the natural representations of the trees t_i, that is, such that*

$$\langle t_0 \rangle = \max\{\langle t_i \rangle \mid 1 \leq i \leq n\}.$$

In [3], we showed how to restrict the exploration to just the natural representations of canonical representatives, for the form of natural representations proposed there. Our simpler version in the present paper also allows for a similar development:

Theorem 6. *A natural sequence x corresponds to a canonical representative if and only if for any natural sequences y, z and any $d \geq 0$ such that $(y+d)\cdot(z+d)$ is a subsequence of x, it holds that $y \geq z$ in lexicographical order.*

Corollary 7. *Let a natural sequence x correspond to a canonical representative. Then its extension $x \cdot (i)$ corresponds to a canonical representative if and only if, for*

FOS_MINING(t, D, min_sup, T)

Input: A tree t, a tree dataset D, and min_sup.
Output: The frequent tree set T.

```
1  insert t into T
2  for every t' that can be extended from t in one step
3       do if support(t') ≥ min_sup
4            then FOS_MINING(t', D, min_sup, T)
5  return
```

Figure 3. The FOS (Frequent Ordered Subtree) Mining algorithm

INCREMENTAL_ORDERED_SUBTREE(st, t, pos_{st}, pos_t)

Input: A tree st, a tree t, and positions pos_{st}, pos_t such that the st prefix of length $pos_{st} - 1$ is a subtree of the t prefix of length pos_t.
Output: **true** if st is a subtree of t.

```
1  while pos_st ≠ END_OF_TREE
2       do if depth(st, pos_st) > depth(t, pos_t)
3            do while depth(st, pos_st) ≠ depth(t, pos_t)
4                 do pos_st = pos_st − 1
5          if depth(st, pos_st) = depth(t, pos_t)
6            do pos_st = pos_st + 1
7          pos_t = pos_t + 1
8  return pos_st = END_OF_TREE
```

Figure 4. The Incremental Ordered Subtree test algorithm

any natural sequences y, z and any $d \geq 0$ such that $(y + d) \cdot (z + d)$ is a suffix of $x \cdot (i)$, it holds that $y \geq z$ in lexicographical order.

We build an incremental canonical checking algorithm, using the result of Corollary 7.

The algorithm is as follows: each time we add a node of depth d to a tree t we check for all depths less than d, that the two last child subtrees are correctly ordered. As it is an incremental algorithm, and the tree that we are extending is canonical, we can assume that child subtrees are ordered, so we only have to check the two last ones.

4.1 Closure-based mining

Closure-based mining algorithms can be obtained from the previous algorithms by checking that the support is indeed lower for all potential extensions of a given tree, before counting it into the closed trees.

Figure 5 shows the pseudocode of CUS_MINING. It is similar to FOS_MINING, adding a closure check in lines 9–10, and a canonical representative test at the beginning.

CUS_MINING(t, D, min_sup, T)

Input: A tree t, a tree dataset D, and min_sup.
Output: The closed frequent tree set T.

```
1  if t ≠ CANONICAL_REPRESENTATIVE(t)
2    then return
3  C ← ∅
4  for every t' that can be extended from t in one step
5       do if support(t') ≥ min_sup
6            then insert t' into C
7       do if support(t') = support(t)
8            then t is not closed
9  if t is closed
10   then insert t into T
11 for each t' in C
12      do CUS_MINING(t', D, min_sup, T)
13 return
```

Figure 5. The CUS (Closed Unordered Subtree) Mining algorithm

5 Empirical Validations

We tested our algorithms on synthetic and real data, comparing the results with CMTreeMiner [5], using its original implementation available on the web. All experiments were performed on a 3.6GHz Pentium 4 processor with 1GB main memory, running Linux. As far as we know, CMTreeMiner is the only algorithm for mining induced closed frequent trees in databases of rooted trees. The main difference with our approach is that CMTreeMiner works with labeled nodes and unrooted trees, and we deal with unlabeled rooted trees.

On synthetic data, we use the same dataset as in [5] and [11] for rooted ordered trees restricting the number of distinct node labels to one. The running times showed that our algorithm compares well to CMTreeMiner since for supports above 2% our algorithm is slightly faster, and for the unordered case, where we take care of avoiding repetitions of structures that are isomorphic under the criterion of unordered trees (which CMTreeMiner would not prune) our method, although a bit worse, is not too far off.

As a test on data coming from real life, we tested our algorithm and CMTreeMiner using KDD Cup 2000 data [6].

This dataset is a web log file of a real internet shopping mall (gazelle.com). This dataset of size 1.2GB contains 216 attributes. We use the file attribute 'Session ID' to associate to each user session a unique tree. The trees record the sequence of web pages that have been visited in a user session. Each node tree represents a content, assortment and product path. Trees are not built using the structure of the web site, instead they are built following the user streaming. Each time a user visit a page, if he has not visited it before, we take this page as a new deeper node, otherwise, we backtrack to the node this page corresponds to, if it is the last node visited on a concrete depth. The resulting dataset consists of $225,558$ trees.

Figures 6 and 7 show the results of our experiments on these real-life data: we see that our method is better than CMTreeMiner at all values of support, both for ordered and unordered approaches.

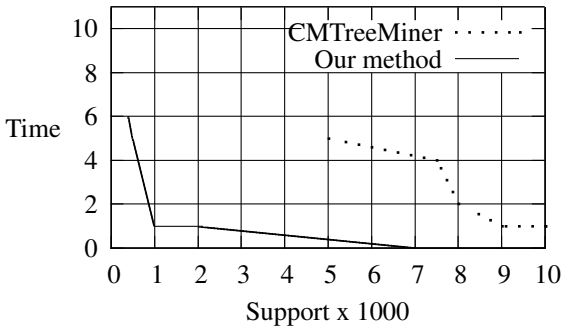

Figure 6. Real data experimental results on Ordered Trees: Support versus Running Time in seconds

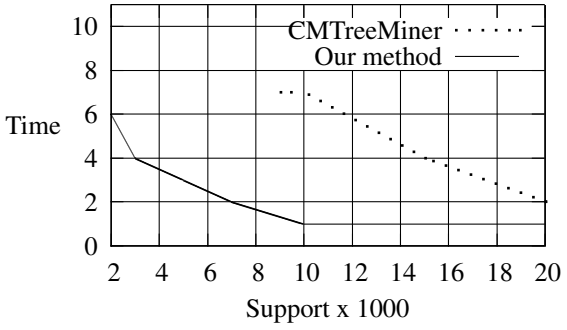

Figure 7. Real data experimental results on Unordered Trees: Support versus Running Time in seconds

6 Conclusions

Through our proposed representation of ordered trees, we have adapted basic algorithmics on the data type such as subtree test, and also our previous algorithms for mining ordered and unordered frequent closed trees. Further applications of these algorithms are envisaged, and a number of variants suggest themselves for further study. Embedded subtrees, the case where the root is not necessarily preserved, and extensions to labeled trees, all should range from being rather straightforward up to at least feasible.

We believe that the sequential form of our representation, where the number-encoded depth furnishes the two-dimensional information, is key in the fast processing of the data, and will be useful in further studies, algorithms, and applications of similar techniques.

References

[1] H. Arimura and T. Uno. An output-polynomial time algorithm for mining frequent closed attribute trees. In *ILP*, pages 1–19, 2005.

[2] J. L. Balcázar, A. Bifet, and A. Lozano. Intersection algorithms and a closure operator on unordered trees. In *MLG 2006, 4th International Workshop on Mining and Learning with Graphs*, pages 1–12, 2006.

[3] J. L. Balcázar, A. Bifet, and A. Lozano. Mining frequent closed unordered trees through natural representations. *To appear in Proceedings of ICCS 2007, 15th International Conference on Conceptual Structures*, 2007.

[4] Y. Chi, R. Muntz, S. Nijssen, and J. Kok. Frequent subtree mining – an overview. *Fundamenta Informaticae*, XXI:1001–1038, 2001.

[5] Y. Chi, Y. Xia, Y. Yang, and R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *Fundamenta Informaticae*, XXI:1001–1038, 2001.

[6] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000.

[7] A. Termier, M.-C. Rousset, and M. Sebag. DRYADE: a new approach for discovering closed frequent trees in heterogeneous tree databases. In *ICDM*, pages 543–546, 2004.

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

[9] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 721, Washington, DC, USA, 2002. IEEE Computer Society.

[10] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295, New York, NY, USA, 2003. ACM Press.

[11] M. J. Zaki. Efficiently mining frequent trees in a forest. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, New York, NY, USA, 2002. ACM Press.