Closed and Maximal Tree Mining Using Natural Representations

José L. Balcázar BALQUI@LSI.UPC.EDU
Albert Bifet ABIFET@LSI.UPC.EDU
Antoni Lozano ANTONI@LSI.UPC.EDU
Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics

1. Introduction

Mining frequent trees is becoming an important task, with broad applications including chemical informatics, computer vision, text retrieval, bioinformatics, and Web analysis. Many link-based structures may be studied formally by means of unordered trees.

Closure-based mining on purely relational data, that is, itemset mining, is, by now, well-established, and there are interesting algorithmic developments. Sharing some of the attractive features of frequency-based summarization of subsets, it offers an alternative view with both downsides and advantages; among the latter, there are the facts that, first, by imposing closure, the number of frequent sets is heavily reduced and, second, the possibility appears of developing a mathematical foundation that connects closure-based mining with lattice-theoretic approaches like Formal Concept Analysis.

Yan and Han (Yan & Han, 2002; Yan & Han, 2003) proposed two algorithms for mining frequent and closed graphs. The first one is called gSpan (graph-based Substructure pattern mining) and discovers frequent graph substructures without candidate generation. gSpan builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the depth-first search strategy to mine frequent connected subgraphs. The second one is called CloseGraph and discovers closed graph patterns. CloseGraph is based on gSpan, and is based on the development of two pruning methods: equivalent occurrence and early termination. The early termination method is similar to the early termination by equivalence of projected databases method in CloSpan (Yan et al., 2003), an algorithm for mining closed sequential patterns in large datasets of the same authors. However, in graphs there are some cases where early termination may fail and miss some patterns. By detecting and eliminating these cases, CloseGraph guarantees the completeness and soundness of the closed graph patterns discovered.

In the case of trees, there are two broad kinds of subtrees considered in the literature: subtrees which are just induced subgraphs, called *induced subtrees*, and subtrees where contraction of edges is allowed, called *embedded subtrees*. A survey of works on frequent subtree mining can be found in (Chi et al., 2001a).

These mining processes can be used for a variety of tasks. For example, consider web search engines. Already the high polysemy of many terms makes sometimes difficult to find information through them; for instance, a researcher of soil science may have a very literal interpretation in mind when running a web search for "rolling stones", but it is unlikely that the results are very satisfactory; or a computer scientist interested in parallel models of computation has a different expectation from that of parents-to-be when a search for "prams" is launched. A way for distributed, adaptive search engines to proceed may be to distinguish navigation on unsuccessful search results, where the user follows a highly branching, shallow exploration, from successful results, which give rise to deeper, little-branching navigation subtrees.

As an example, consider the KDD Cup 2000 data (Kohavi et al., 2000). 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 used the 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 visits 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. On them, an unlabeled tree variation of our algorithms (Balcázar et al., 2007b) was considerably

faster than the only alternative algorithm now available for the task, CMTreeMiner(Chi et al., 2001b), as Figure 1 shows.

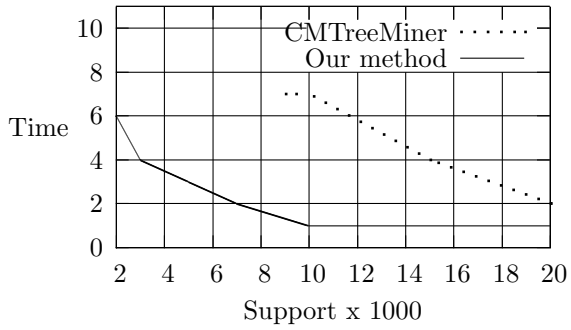

Figure 1. Gazelle experimental results on Unordered Trees: Support versus Running Time(seconds)

Our focus is on 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 present a new algorithm for mining frequent closed and maximal, ordered and unordered, labeled trees.

2. Labeled Natural Representations

In previous work (Balcázar et al., 2007b), we represent each unlabeled tree as a sequence over a countably infinite alphabet, namely, the set of natural numbers. To extend this representation to labeled trees, we will represent each node as pair composed of a natural number and a label, and each tree as a sequence over the set of pairs of natural numbers and labels; we will concentrate on a specific language, whose strings exhibit a very constrained growth pattern. Some simple operations on strings of nodes are:

Definition 2.1. Given two sequences of nodes x, y, we represent by by $|x|$ the number of pairs of x, by $x \cdot y$ the sequence obtained as concatenation of x and y, and x^+ the sequence obtained adding 1 to the first components of each pair in x.

The language we are interested in is formed by sequences which never "jump up": each time the natural number component of the pair grows is only by one. This kind of sequences will be used to describe trees.

Definition 2.2. A *labeled natural sequence* is a sequence $((x_1, l_1) \ldots, (x_n, l_n))$ of pairs of natural numbers and labels 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, $x = ((0, A), (1, B), (2, A), (3, B), (1, C))$ is a natural sequence that satisfies $|x| = 6$ or $x = ((0, A)) \cdot ((0, B), (1, A), (2, B))^+ \cdot ((0, C))^+$. Now, we are ready to represent trees by means of natural sequences.

Definition 2.3. We define a function $\langle \cdot \rangle$ from the set of ordered trees to the set of labeled natural sequences as follows. Let t be an ordered tree. If t is a single node, then $\langle t \rangle = ((0, l_0))$. 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

$$\langle t \rangle = ((0, l_0)) \cdot \langle t_1 \rangle^+ \cdot \langle t_2 \rangle^+ \cdot \ldots \cdot \langle t_k \rangle^+$$

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

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

Our representation allows us to traverse the whole subtree space by an operation of extension by a single node, in a simple way. The possible places where the new node can be added in our natural representation belong to the right-most branch of the tree.

In (Balcázar et al., 2006) we aim at clarifying the properties of closed trees, providing a more detailed justification of the term "closed" through a closure operator obtained from a Galois connection, along the lines of (Ganter & Wille, 1999)or (Balcázar & Garriga, 2005) for unstructured or otherwise structured datasets. Also, we designed two algorithms for finding the intersection of two subtrees: the first one in a recursive way, and the second one using dynamic programming.

In unordered trees, the children of a given node form sets of siblings instead of sequences of siblings. Therefore, ordered 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 2.4. 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

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

3. Labeled Subtree Testing

Subtree testing of two ordered trees can be obtained by performing a simultaneous preorder traversal of the two trees (Valiente, 2002). In (Balcázar et al., 2007b) we showed an algorithm for unlabeled trees. For the labeled case, we modify the checking of the depth of the node, adding an additional label order checking.

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 $O(n_r n_t^{1.5})$((Valiente, 2002)), where n_r and n_t are the number of nodes 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 this computations at the beginning of the algorithm.

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 = a node tree
6     then if label(r) is in labels(Subcomponents(t))
7             then Return true
8             else  Return false
9  graph ← {}
10 for each s_r in Subcomponents(r)
11       do for each s_t in Subcomponents(t)
12             do if (Unordered_Subtree(s_r, s_t))
13                   then insert(graph, edge(s_r, s_t))
14 if BipartiteMatching(graph)
15    then D(r,t) ← true
16    else  D(r,t) ← false
17 return D(r,t)
```

Figure 2. The Unordered Subtree test algorithm

4. Mining closed and maximal subtrees

Our approach is similar to gSpan: we represent the potential subtrees to be checked for frequent on the dataset in such a way that extending them by one single node corresponds to a clear and simple operation on the representation. The completeness of the procedure is mathematically proved in (Balcázar et al., 2007a) , that is, we argue there that all trees can be obtained in this way with our notion of extension, defined below. This allows us to avoid extending trees that are found to be already nonfrequent.

We propose a new algorithm to mine unordered frequent closed and maximal trees. As a frequent tree is *closed* if none of t's proper supertrees has the same support that t has, a frequent tree t is *maximal* if none of t's proper supertrees is frequent. Note that as all maximal trees are closed, but not all closed trees are maximal, there are more closed trees than maximal.

The method is as follows: beginning with a tree of a single node, it calls recursively the Closed_Maximal_Subtree_ Mining algorithm doing one-step extensions and checking that they are closed or maximal. Note that the unique difference between labeled and unlabeled versions of this method is how the subtree testing methods are implemented to compute tree support. Figure 3 shows the pseudocode of Closed_Maximal_Subtree_ Mining for unordered trees. In the case of ordered trees, we don't need to check that trees are canonical representatives, in lines 1 and 2.

CLOSED_MAXIMAL_SUBTREE_MINING(t, D, min_sup, C, M)

Input: A tree t, a tree dataset D, and min_sup.
Output: The closed frequent tree set C and the maximal tree set M.

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

Figure 3. The Closed Maximal Ordered Labeled Subtree Mining algorithm

References

Balcázar, J. L., Bifet, A., & Lozano, A. (2006). Intersection algorithms and a closure operator on unordered trees. *MLG 2006, 4th International Workshop on Mining and Learning with Graphs.*

Balcázar, J. L., Bifet, A., & Lozano, A. (2007a). Mining frequent closed rooted trees. Submitted.

Balcázar, J. L., Bifet, A., & Lozano, A. (2007b). Subtree testing and closed tree mining through natural representations. *ACKE 2007, Workshop on Advances in Conceptual Knowledge Engineering.*

Balcázar, J. L., & Garriga, G. C. (2005). On Horn axiomatizations for sequential data. *ICDT* (pp. 215–229 (extended version to appear in Theoretical Computer Science)).

Chi, Y., Muntz, R., Nijssen, S., & Kok, J. (2001a). Frequent subtree mining – an overview. *Fundamenta Informaticae*, *XXI*, 1001–1038.

Chi, Y., Xia, Y., Yang, Y., & Muntz, R. (2001b). Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *Fundamenta Informaticae*, *XXI*, 1001–1038.

Ganter, B., & Wille, R. (1999). *Formal concept analysis.* Springer-Verlag.

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

Valiente, G. (2002). *Algorithms on trees and graphs.* Berlin: Springer-Verlag.

Yan, X., & Han, J. (2002). gSpan: Graph-based substructure pattern mining. *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)* (p. 721). Washington, DC, USA: IEEE Computer Society.

Yan, X., & Han, J. (2003). CloseGraph: mining closed frequent graph patterns. *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 286–295). New York, NY, USA: ACM Press.

Yan, X., Han, J., & Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large databases. *SDM.*