All Common Embedded Subtrees for Measuring Tree Similarity

Zhiwei Lin, Hui Wang, Sally McClean, Chang Liu
Faculty of Computing and Engineering
University of Ulster, Northern Ireland, UK
{z.lin,h.wang,si.McClean}@ulster.ac.uk

Abstract

Tree similarity measurement is key to tree-like data mining. In order to maximally capture common information between trees, we consider the problem of computing all common embedded subtrees, and advocate using the number/count *of all common embedded subtrees as a measure of similarity. This problem is not trivial due to the inherent complexity of trees and the ensued large search space. The problem is theoretically analyzed and an effective algorithm for counting all common embedded subtrees is presented. Experimental evaluation shows that the all common embedded subtree similarity is very competitive against tree edit distance, in terms of both efficiency and effectiveness.*

1 Introduction

Tree structures occur in different types of data. For example, in text mining, every (correct) sentence follows some grammar rules and the grammatical information is usually represented as parse trees; in web mining, an XML page has an inherent tree structure; in computational biology, RNA secondary is tree-structured; and in computational chemistry, molecules are also tree structured. Measuring the similarity or dissimilarity (distance) of such structures is key to mining/analyzing these types of data.

Two most studied and widely used measures are *largest common subtree*(LCT)[1, 6] and *tree edit distance*(TED)[2, 3, 4, 7, 8]. As similarity measures, LCT advocates using the size of the largest common subtree as an indication of similarity between trees and TED advocates using the shortest sequence of edit operations needed to transform one tree to another as an indication of similarity between trees. As computer science problems they both study efficient, polynomial algorithms to compute LCT and TED respectively.

These two measures are intuitively sound, but they both have the same problem of relying completely on some special elements in trees. TED is concerned with the shortest sequence of edit operations while LCT is concerned with the largest common subtree. There is no doubt that the largest common subtree contains most of the common information between trees, but the second largest and in general the n-th largest common subtrees all contain common information to some degree. This argument also applies to TED. Consequently their performances can be compromised. If we want to maximally capture the commonality between trees we would need to consider all common substructures, not just the largest common substructure.

The notion of subtree is traditionally defined as a portion of a tree, and it has been used in LCT. This definition has, however, limitations when used in some applications, and alternative definitions have been introduced, e.g., *induced subtree*, *bottom-up subtree* and *embedded subtree* [5]. Embedded subtree is broader than other definitions including the traditional definition. It additionally considers the context and topological structures of a tree. In order to capture as much information in a tree as possible we use embedded subtree and consider the problem of how to find *all common embedded subtrees* (ACETs), not all common subtrees.

In this paper, we propose to use the *number of all common embedded subtrees* as a measure of tree similarity (the ACET similarity). However the ACET problem is not trivial due to the inherent complexity of trees and the ensued large search space. We take a divide and conquer approach where a problem is broken down into sub-problems, which are then solved recursively. In order to avoid solving a subproblem more than once we conduct a theoretical study on the relationship between a problem and its sub-problems. Based on the theoretical results, we have developed an effective algorithm to compute the number of all common embedded subtrees. This algorithm has polynomial complexity in time and space. We conducted some experiments with parse trees obtained from some text documents in a TREC collection. Results show that ACET similarity is significantly more efficient and also significantly more effective than TED.

This paper is organized as follows. In Section 2, we present some notations used in the paper followed by an extended definition of subtree - *embedded subtree* - the subject of this paper. This is followed by the *all common embedded subtrees* problem and a polynomial algorithm in Section 3. In Section 4, we present experimental results to show the efficiency and effectiveness of ACET. Finally, the paper is concluded with a summary.

2 Preliminaries and Notation

An *ordered tree* is a labeled tree $T = (V, E)$, where V is a set of nodes, E is a set of edges; T has a *root* node, denoted by $root(T)$; and there is left-right order over the children of each node. In the rest of this paper, we assume trees are ordered unless otherwise stated.

The *size* of T, denoted by $|T|$, is the number of the nodes in tree T, i.e., $|T| = |V(T)|$. An *empty tree* is denoted by ϵ which has a size of 0. A *descendant* of v is a child of v or a descendant of a child of v; similarly an *ancestor* of v is its parent or an ancestor of its parent. A *subtree* rooted at v includes v and all descendants of v and is denoted by $T(v)$. We use $T(v, w)$ to represent a *subtree* of T, containing all nodes from v to w in the pre-order traversal of T. Two nodes n_1 and n_2 are *siblings* if there exists $n \in T$ such that $n_1 \in T(n)$, $n_2 \in T(n)$, $n_1 \notin T(n_2)$, and $n_2 \notin T(n_1)$.

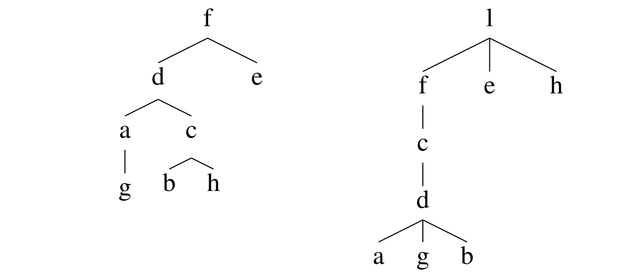
Let $v \in T$ and $v \neq root(T)$, $T - v$ refers to the *delete operation* on v, having two cases (1)if v is a leaf, remove v from T; (2) otherwise, remove v from T as well as move the children of v as the new children of the parent of v from the place of v with left-right order. An *embedded subtree* $T'(v)$ of tree T is obtained by deleting some non-root nodes from $T(v)$, denoted by $T'(v) \preceq T$. It is not hard to see that the $\preceq$ relation is reflexive and transitive.

3 All common embedded subtrees as a similarity measure

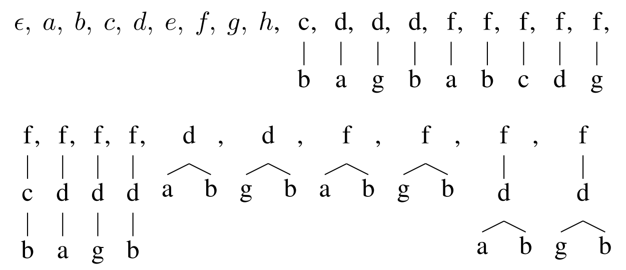
If two trees resemble each other there must be some commonality between them. As a tree describes structures among a set of objects (nodes), tree commonality should be structural. In other words these two trees should have some common sub-structures. Therefore, it is intuitively true that the more common embedded subtrees there are between two trees, the higher the commonality is. This understanding leads to the *all common embedded subtrees* (ACET for short) similarity – the subject of this paper.

Definition 1 *The* ACET similarity *of two trees is the number of all common embedded subtrees between them, where the definition of embedded subtree is presented in Section 2.*

We will use $ACET(T_1, T_2)$ as a similarity function for trees T_1 and T_2.



(1) T_1 and T_2


ε, a, b, c, d, e, f, g, h,
c, d, d, d, f, f, f, f, f,
b a g b a b c d g
f, f, f, f, d, d, f, f, f, f
c d d d
b a g b
a b g b a b g b
d d
a b g b

(2) All common embedded subtrees of T_1 and T_2

Figure 1. Two trees and all common embedded subtrees.

Example 1 *Consider the two trees T_1 and T_2 in Figure 1(1). All common embedded subtrees of T_1 and T_2 (containing empty tree ϵ) are illustrated in Figure 1(2) and the total number is 28.*

To compute common embedded subtrees we need to traverse both trees and compare the nodes from the two trees. If we find a match, i.e., a common node, we need to know what additional common embedded subtrees should be generated. The following lemma describes the conditions under which new common embedded subtrees can be generated.

Lemma 1 *Consider two trees T_1 and T_2. Let v_k, v_{k_i}, $w \in T_1$, and v_k, v_{k_i}, $w \in T_2$, where v_k is an ancestor of w in both trees, and $v_{k_i} \in T_1(v_k, w)$ and $v_{k_i} \in T_2(v_k, w)$, i.e., v_{k_i} is a node between v_k and w in the pre-order traversals of both trees. Then the following two statements are true:*

1. *if w is a sibling of v_{k_i} in both T_1 and T_2, then v_k, v_{k_i}, and w can form a common embedded subtree rooted at v_k.*
2. *if w is a descendant of v_{k_i} in both T_1 and T_2, then v_k, v_{k_i}, and w can form a common embedded subtree rooted at v_k.*

We refer to the new common embedded subtree as $CST(v_k, v_{k_i}, w)$, which contains three nodes.

Lemma 1 is key to the algorithm in this paper. To this end, we present an example to explain Lemma 1.

Example 2 *Consider the two trees in Figure 1(1). Suppose v_k and w are two common nodes in both trees, and v_k is an ancestor of w. A node v_{k_i} may be a sibling or an ancestor of w in both trees or in one of the trees. There are three possible scenarios:*

- *v_{k_i} is a sibling of w in both trees. Let $v_k \overset{\text{def}}{=} d$, $w \overset{\text{def}}{=} b$ and $v_{k_i} \overset{\text{def}}{=} g$ in Figure 1(1). It is clear that node g is a sibling of b in both trees, and d is their common ancestor. In this case, the embedded subtree containing three nodes of d, g and b, can be found by Figure 1(2).*
- *v_{k_i} is an ancestor of w in both trees. Let $v_k \overset{\text{def}}{=} f$, $w \overset{\text{def}}{=} b$ and $v_{k_i} \overset{\text{def}}{=} c$. It is clear that node c is an ancestor of b in both trees, and f is an ancestor of both c and b. In this case, the embedded subtree containing three nodes of f, c and b, can be found by Figure 1(2).*
- *v_{k_i} is a sibling of w in one tree while an ancestor in the other. Let $v_k \overset{\text{def}}{=} d$, and $w \overset{\text{def}}{=} g$ and $v_{k_i} \overset{\text{def}}{=} a$. In this case g is a descendant of a in T_1 and a sibling of a in T_2. As we can see from Figure 1(2), no common embedded subtree can be found.*

From Lemma 1 and the above example, we design an algorithm listed in Algorithm 3.1, which has a polynomial complexity. The algorithm is initiated from line 1 to 12, where the vectors $Ind_A[i]$ and $Ind_B[j]$ indicate whether the element in a tree is common to another tree. Line 13 is to traverse tree T_1 and Line 15 is to find all the common ancestors of T_2. Line 22 uses Lemma 1. The time complexity of the algorithms is $O(|T_1| \times |T_2| \times min\{|T_1|, |T_2|\} \times max\{depth(T_1), depth(T_2)\})$.

Algorithm 3.1: $ACET(T_1, T_2)$

```
   Input: Two trees $T_1$ and $T_2$.
   Output: $act$ – The number of ACETs.
1  Initialization:
2    $act = 1$;   /*empty tree is counted. */
3    $A$ and $B$ are pre-order sequences of $T_1$ and $T_2$
4    $Ind_A[1..|A|] = -1$; $Ind_B[1..|B|] = -1$;
   /*All the members of vectors $Ind_A$
     and $Ind_B$ are initialized with -1 */
5  for ($i = 1; i \leq |A|; i++$) do
       /*Initialize matrix and vectors */
6      for ($j = 1; j \leq |B|; j++$) do
7          if $A_i = B_j$ then
               /*$A_i$ is a common node      */
8              $Ind_A[i] = j$; $Ind_B[j] = i$;
9          end
10         $N[i, j] = 0$;
11     end
12 end
13 for ($i = 1; i \leq |A|; i++$) do
14     if $Ind_A[i] > 0$ then
           /*$Ind_A[i] < 0$ means that $A_i \notin T_2$ */
15         for ($j = 1; j <= Ind_A[i]; j++$) do
16             if $A_i = B_j$ then
                   /*node $A_i$ is a common
                     embedded subtree       */
17                 $N[i, j] = 1$;
18             else
19                 $k = i - 1$;
20                 if $B_j$ is $A_i$'s ancestor in both $T_1$ and
                   $T_2$ then
21                     for ($k; k \geq Ind_B[j]; k--$) do
                           /*The following uses
                             Lemma 1.          */
22                         if ($Ind_A[i] > Ind_A[k]$) and
                           ($CST(B_j, A_k, A_i)$ exists)
                           then
23                             $N[i, j] += N[k, j]$;
24                         end
25                     end
26                 end
27             end
28             $act += N[i, j]$;
29         end
30     end
31 end
```

4 Experimental evaluation

To test the efficiency and effectiveness of Algorithm 3.1, we conducted experiments on classification on a DELL Inspiron 6400 laptop with Intel T2300(1.66GHz) and 1GB memory installed. For this we compare ACET with TED in a classification experiment. In this section we present our experimental results.

4.1 Data set

The data set in this experiment is based on a TREC collection – Text Research Collection Volume 2, Revised March 1994. This collection contains text files from the Wall Street Journal (1990, 1991, and 1992). The TREC collections are prepared for use at TREC conferences and each of the TREC collections consists of a set of documents, a

k	ACET (%)	TED (%)
1	86.5	41.5
2	86.5	41.5
3	86.5	37.5
4	85.5	38.0
5	85.5	35.0
6	85.0	35.0
7	81.5	36.0
8	81.0	36.0

Table 1. Comparison of ACET and TED in terms of classification accuracy.

set of topics (queries), and a corresponding set of relevance judgments (right answers).

In this experiment we randomly selected three topics – 251, 289 and 291. Based on the relevance judgments we selected 70 documents each for topics 251 and 289, and 60 documents for topic 291. Altogether we get 200 documents. Then we extract the title of each document and associate each title with the topic that the document belongs to. We thus construct a set of 200 labeled document titles, where a label is a topic.

Once the document titles are ready, we move on to construct representations for the titles. We employ Minipar [1] to parse all the titles, resulting in 200 parse trees as a representation of the 200 titles. We put all the parse trees together along with their topic labels, giving rise to our data set D.

4.2 Experimental details and results

Since our aim is to evaluate ACET we decide to use the k-*nearest neighbor* classifier, where the similarity measures are ACET and TED [4]. As the data set is not very large we decide to use the *leave-one-out* evaluation method.

For each similarity measure we run the experiment 8 times with different k values. The experiment result is shown in Table 1. It is very clear that ACET is significantly and consistently outperforming TED under various k values. Table 2 shows the time needed to compute the similarity by these two algorithms, where ACET uses 5 seconds while TED use 31 seconds. Apparently, ACET is more time-efficient than TED.

It is clear from the two sets of experiment that ACET outperforms TED consistently, both in terms of efficiency and effectiveness. This provides evidence that ACET is a strong alternative to TED, the de facto standard of tree similarity measure.

[1] http://www.cs.ualberta.ca/ lindek/minipar.htm

ACET	TED
5 seconds	31 seconds

Table 2. Comparison of ACET and TED in terms of running time.

5 Conclusions

In this paper we consider the problem of computing *all common embedded subtrees* (ACETs) when the concept of subtree is defined in a broader sense, which is aimed to maximally capture the common information in trees by considering all common embedded subtrees. We design an algorithm to solve the ACET problem. The algorithm has polynomial complexity in both time and space. Experiment results show that ACET outperforms TED consistently in terms of both efficiency and effectiveness. Future work includes applications of ACET to various problems where tree structured data abound.

References

[1] T. Akutsu and M. M. Halldorsson. On the approximation of largest common point sets and largest common subtrees. In *Proc. of 5th Ann. Int. Symp. on Algorithms and Computation*, pages 405–413, 1994.

[2] P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3), 2005.

[3] S.S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom, Change Detection in Hierarchically Structured Information. In *Proceedings of the ACM SIGMOD Conference*, USA, Pages 493-504, 1996,

[4] S.S. Chawathe. *Comparing hierarchical data in external memory*. In *Proceedings of the VLDB Conference*, pages 90-101, Edinburgh, UK 1999.

[5] Y. Chi. R.R. Muntz. S. Nijssen. J.N. Kok. Frequent subtree mining – an overview. *Fundamenta Informaticae*, Pages 161–198, 66(1-2),2005.

[6] S. Khanna, R. Motwani, and F. F. Yao. Approximation algorithms for the largest common subtree problem. Technical report, Stanford University, 1995.

[7] S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6:184–186, Dec 1977.

[8] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18:1245–1262, Dec 1989.