Algorithms for Computing Maximum Agreement Subtrees

Nikolaj Skipper Rasmussen 20114373

Thomas Hedegaard Lange 20113788

Master's Thesis, Computer Science

February 2016

Advisor: Christian Nørgaard Storm Pedersen

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

►in English. . . ◄

Resumé

►in Danish. . . ◄

Acknowledgements

►. . . ◄

Karsken Bælg,
Aarhus, February 29, 2016.

Contents

Chapter 1

Introduction

The Maximum Agreement Subtree problem (MAST) provides a measure of similarity, and is defined as such: Given two rooted trees, T1 and T2, created over the same leaf-set {1,2,3...,n}, determine the largest possible subset of leaves inducing an agreeing subtree of T1 and T2. For a set of leaves to induce an agreeing subtree for T1 and T2, the subtrees restricted to the set of leaves must be isomorphic, which means that they are structurally equivalent.

Let us start by motivating the interest in MAST by giving an example of its application. Suppose that we are interested in inspecting the relationship between DNA obtained from different species. This is typically done by the use of Hierachical Clustering (REF) or Neighbor Joining (REF) to construct evolutionary trees. However, finding the true evolutionary tree is often hard (find another way of expressing hard), and evidence is required to support any suggested tree topology.

The MAST problem is one of several ways of defining tree distances. -which one is supeiror?

The MAST problem applies to all trees, but we will choose to focus on the rooted, binary trees given that the motivation for the problem is primarily rooted in biology and linguistics, where these trees are most common.

$$\begin{aligned} \frac{3x+y}{7} &= 9 && \text{given} \\ 3x+y &= 63 && \text{multiply by 7} \\ 3x &= 63-y && \text{subtract y} \\ x &= 21-\frac{y}{3} && \text{divide by 3} \end{aligned}$$

$$f(T_a, T_x) = Max \begin{cases} f(T_b, T_y) + f(T_c, T_z) & \text{if TypeOf}(T_a)\text{=Type Of}(T_x)\text{=Internal Node} \\ f(T_b, T_z) + f(T_c, T_y) & \text{if TypeOf}(T_a)\text{=Type Of}(T_x)\text{=Internal Node} \\ f(T_a, T_y) & \text{if TypeOf}(T_x)\text{=Internal Node} \\ f(T_a, T_z) & \text{if TypeOf}(T_x)\text{=Internal Node} \\ f(T_b, T_x) & \text{if TypeOf}(T_a)\text{=Internal Node} \\ f(T_c, T_x) & \text{if TypeOf}(T_a)\text{=Internal Node} \\ 1 & \text{if TypeOf}(T_a)\text{=TypeOf}(T_x)\text{=Leaf} \wedge T_a{=}T_x \\ 0 & \end{cases}$$

Algorithm 1 Naive solution

```
1: procedure MyProcedure
2:     stringlen ← length of string
3:     i ← patlen
4: top:
5:     if i > stringlen then return false
6:     end if
7:     j ← patlen
8: loop:
9:     if string(i) = path(j) then
10:        j ← j − 1.
11:        i ← i − 1.
12:        goto loop.
13:        close;
14:    end if
15:    i ← i + max(delta1(string(i)), delta2(j)).
16:    goto top.
17: end procedure
```

1.1 Induced subtree

Recall that the centroid decomposition of the main algorithm splits T_1 and T_2 into a forest of subtrees. We are in the second step of the algorithm required to determine the subtrees of T_2 induced by the leaves of $T_1's$ subtrees. We will now show how to achieve this in linear time with respect to the number of leaves, provided that T_2 has been prepossed in linear time.

1.1.1 Preprocessing T_2

To induce subtrees in linear time, two requirements must be met. The Least Common Ancestor (LCA) between two leaves should be computed in constant time, and the depth of each vertex in the tree must be known. Both of these requirements can be achieved in linear time with respect to the size of the tree.

The vertex depth can be found in linear time by a simple pre-order tree traversal (REF?), where each node sets its children's depth to it's own depth+1. The depth of the root is initializes to zero.
Computing LCA in $O(1)$ is a tricky afair, and is covered in Section/Chapter ??

1.1.2 The Algorithm

We are given a tree and an ordered set of leaves, $L = l_1, l_2, l_3, ..., l_n$,and must find the subtree induced by this set. The algorithm works in 2 steps:

1. The LCA of each pair of leaves such that $A_i = LCA(l_i, l_i + 1), where 1 \leq i \leq n - 1$

2. The second item

3. The third etc ...

subtrees given an ordered set of nodes in linear time with respect to the nodes.

►...◄

Chapter 2

►. . . ◄

►example of a citation to primary literature: [1], and one to secondary literature: [?]◄

Chapter 3

►The NSquared algorithm◄

Goddard et. al.[1] describes a **►O(n2)◄** algorithm for finding the largest agreement subtree for two rooted binary trees. Given two trees T and U of size m and n, the idea is to iteratively find the largest agreement subtree and its size for every pair of subtrees from T and U. This can be done in quadratic time by using Lemma 1: **►...◄**

3.1 Implementation

We implemented the algorithm in java (using the forrester [**?**] library to represent the trees?). We computed the agreement subtrees for each pair of subtrees in the two trees by doing a postorder traversal of the first tree and for each node did a postorder traversal of the second tree. For each pair of nodes `a` and `w` we computed the agreement subtree $A_{a,w}$ for the two subtrees T_a and T_w having respectively `a` and `w` as roots. For such a pair of nodes there are three cases.
The first case is that `a` and `w` are both leaves. If the leaves are equal, i.e. they have the same name, then $A_{a,w}$ will be the tree consisting of exactly one leaf with that name. Otherwise $A_{a,w}$ is empty.
The second case is that we have a leaf and an internal node. Let's say `w` is the internal node having `x` and `y` as children. If the leaf corresponding to `a` is contained in T_w, it will also be contained in either T_x or T_y and $A_{a,w}$ will be the same as either $A_{a,x}$ or $A_{a,y}$. Since we do a postorder traversal of the trees, $A_{a,x}$ and $A_{a,y}$ have already been computed and $A_{a,w}$ can just be set to the largest of the two.
The third case is that both `a` and `w` are internal nodes, where `a` have children `b` and `c` and `w` have children `x` and `y`. In this case we can use Lemma 1. Again, all the agreement subtrees to consider have already been computed. If the largest subtree is either $A_{b,x} + A_{c,y}$ or $A_{b,y} + A_{c,x}$, $A_{a,w}$ will be the tree having the two subtrees as children.

All these agreement subtrees were stored in a matrix together with their sizes. The last cell in the matrix would then at the end contain the agreement subtree for the two input trees.

How did we verify the algorithm? E.g. bruteforce MAST and compare results.
Show trees and MAST outputted by the program.

►. . . ◄

Chapter 4

The NLogN algorithm

4.1 Centroid decompositions

The first step of the algorithm is to compute the `Centroid decomposition`s for the two trees. A centroid decomposition consists of an amount of disjoint paths through the tree, called `Centroid path`s. Such a path starts at a node in the tree and contains the edge to the child node holding the largest amount of leaves in its subtree. In case of a tie, an arbitrary edge is picked. The path will continue until reaching a leaf.

For the first tree, the centroid decomposition consists of only the centroid path starting at the root. For the second tree, the centroid decomposition consists of all possible disjoint centroid paths.

The implementation of this principle was done by storing the number of leaves in the subtree of a node at each node in the two trees such that this number could be looked up in constant time. Now for the centroid decomposition of the first tree, we simply started at the root node and picked the edge to the node with the highest number stored. For the second tree, the same thing was done, but when picking the edge to a child node, we also started another path at the second child if it wasn't a leaf.

Chapter 5

Conclusion

►. . . ◄

Bibliography

[1] Wayne Goddard, Ewa Kubicka, Grezegorz Kubicki, and F. R. McMorris. The agreement metric for labeled binary trees. *Mathematical Biosciences*, 123(2):215–226, October 1994.