Labelled Isomorphism of Trees

An instance of *Labelled Isomorphism* consists of two graphs G_1 and G_2 and a non-negative integer label for each vertex. We are asked to determine if there is a bijection f from $V(G_1)$ to $V(G_2)$ such that xy is an edge of G_1 precisely if $f(x)f(y)$ is an edge of G_2, and such that for every vertex x of G_1, the label of x and $f(x)$ agree. The size of the instance is the sum of the number of vertices in the two trees and the sum of the logarithms (base 2) of the labels.

We can check if the graphs have the same number of vertices in linear time by scanning through the input once. If not they are obviously not isomorphic and we return this fact. Otherwise we let n be $|V(G_1)|=|V(G_2)|$. If there is a non-negative integer k which is not used as a label and an integer j>k which is used as a label, then we can replace all the labels which are j by k without changing the problem. In linear time we can determine which integers<n are used as labels and which integers>n are not used and replace the latter by the former , thus we can assume all the labels lie between n and log n.

We present here an algorithm for Labelled isomorphism which runs in polynomial time. Indeed it requires O(n log n) simple steps on instances of size n provided we treat comparisions of numbers which are at most n as a simple step.

A separator for a tree T is a node v such that such that every component of T-v contains at most half the vertices of T As discussed in class, every tree T either contains a unique edge xy such x and y are the only separators for T or has a unique separator v. Furthermore, we can find such an xy or v in O(|V|) time as follows . We root the tree and traverse it in post-order. We compute for each node u in turn, the number of nodes in the tree T_u formed by u and its descendant, by adding 1 to the sum of the corresponding values for its children. We let x be the first node for which this sum is at least |V|/2. If the sum is exactly |V|/2 then we let y be the parent of x and return xy, otherwise we set v=x and return v.

So for i in {1,2}, we either find a unique separator v_i of T_i or a pair (x_i,y_i) of separators. Now, for any isomorphism between T_1 and T_2, if z is a separator of T_1 then f(z) is a separator of T_2 because being a separator does not depend on the names of the vertices. So, we can pick a separator z of T_1 and ask for the at most two choices of a separator z' for T_2 whether there is a labelled isomorphism from T_1 to T_2 mapping z to z'. There is a labelled isomorphism between T_1 and T_2 if and only if the answer is yes for one of the at most two choices for z'.

So we need only consider rooted isomorphism on trees, i.e. a labelld isomorphism problem where ach tree has a root and the isomorphism maps one root to the other. For each node t, in one of the two trees, we let T_t be the subtree consisting of t and its descendants. We can label each node by its distance from the root, by traversing the trees using a depth first search. We can build a list L_i of the set S_i of nodes at

distance i from the root of the tree they are in, We actually determine for every two vertices u and v of L_i, whether or not there is a rooted isomorphism between T_u and T_v which maps u to v. More strongly, we partition S_i up into k non-empty equivalence classes, for some k, such that two vertices u and v are in the same equivalence class precisely if T_u and T_v are isomorphic. We then k-colour the elements of S_i so that elements in the same equivalence class get the same colour (which we also call a label).

Having done this for S_{i+1}, we proceed on S_i as follows. We can think of each vertex of S_i with l children as a multiset of l of the labels used on S_i where T_u and T_v have a rooted isomorphism precisely if these multisets are the same. We can use a bucketsort, to order the list, for every u in S_i, in nondecreasing order, in $O(|S_{i+1}|)$ time. We then partition L_i up into smallers list so that the elements in each sublist have the same number of children, in $O(|S_{i+1}|)$ time. We can then order the multisets in each of these smaller lists lexicographically in $O(|S_{i+1}| \log n)$ total time (if there are k_l vertices of Si which have l children then these children are lk_l vertices of S_{i+1} and we need to sort then on each of l indices which takes at most $l(k_l \log n)$ time.