A Note on the Hardness of Tree Isomorphism

Birgit Jenner
Tübingen and Ulm

Pierre McKenzie *
Montréal and Tübingen

Jacobo Torán
Ulm

December 5, 1997

Abstract

In this note we prove that the tree isomorphism problem, when trees are encoded as strings, is NC^1-hard under DLOGTIME-reductions. NC^1-completeness thus follows from Buss's recent NC^1 upper bound. By contrast, we prove that testing isomorphism of two trees encoded as pointer lists is L-complete.

1 Introduction

GI, the graph isomorphism problem, is one of the most intensively studied problems in Theoretical Computer Science. Historically, the main source of interest in GI has been the evidence that GI is probably neither in P nor NP-complete, but other sources of interest include the sophistication of the tools developed to attack the problem (for example [2, 17]), and the connexions between GI and structural complexity (see [14]).

Understandably, many GI restrictions have been considered. For example, P upper bounds are known in the cases of planar graphs [12] or graphs of bounded valence [17]. However, none of these GI restrictions are known to be complete for a natural complexity class, and it seemed that the problem lacks the structure needed for a hardness result.

In the special case of trees, a linear sequential time algorithm for the problem was already known in 1974 to Aho, Hopcroft and Ullman [1]. In 1991, an NC algorithm was developed by Miller and Reif [18]. One year later, Lindell [15] obtained an L upper bound. Finally, in 1997, a subtle algorithm able to test two trees for isomorphism in NC^1 was devised by Buss [7].

Buss in [7] asks whether tree isomorphism is NC^1-hard. Here we answer this question affirmatively showing the hardness of the problem under DLOGTIME many-one reducibility. Hence tree isomorphism is NC^1-complete. Trees thus provide the first class of graphs for which the isomorphism problem captures a natural complexity

*Corresponding author: Pierre McKenzie, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D–72076 Tübingen, Germany. Phone: +49 (7071) 29 74284. Fax: +49 (7071) 29 5061. Email: mckenzie@informatik.uni-tuebingen.de

class. Moreover, so far, the problem of evaluating a Boolean formula [6] and the problem of multiplying permutations on five points [3] (and some of their variations) were the only two NC^1-complete problems known. Tree isomorphism is a third such problem.

As noted by Buss, choosing a graph representation is critical when working at the level of NC^1. Buss uses Miller and Reif's *string* representation for trees on the grounds that, when the *pointer* representation is used, the "deterministic transitive closure problem can be reduced to the descendant predicate, and the former is known to be complete for logspace" [7, Section 2].

We prove here that the tree isomorphism problem is L-hard under many-one NC^1-reducibility when the pointer representation is used. Hence, tree isomorphism in the pointer representation is L-complete by Lindell [15] or by Buss [7]. Tree isomorphism thus captures in this way another very natural complexity class.

2 Preliminaries

We assume familiarity with basic notions of complexity theory such as can be found in the stardard books in the area. In particular, we simply recall that

$$\mathrm{AC}^0 \subset \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L},$$

where L is the set of languages accepted by deterministic Turing machines using logarithmic space, NC^1 is the class of languages recognized by DLOGTIME-uniform families of logarithmic depth, polynomial size, Boolean circuits of bounded fan-in over the basis $\{\wedge, \vee, \neg\}$, and AC^0 (resp. TC^0) is the set of languages recognized by DLOGTIME-uniform families of constant depth, polynomial size, Boolean circuits of unbounded fan-in over the basis $\{\wedge, \vee, \neg\}$ (resp. over the basis consisting solely of the *MAJORITY* function).

For simplicity, we will often not distinguish between a class of Boolean functions like NC^1 and the corresponding class of functions from $\{0,1\}^*$ to the natural numbers, sometimes called FNC^1.

2.1 Reducibilities

We use the following reducibilities:

- DLOGTIME reducibility ($\leq^{DLT}$)

We use this reducibility for the NC^1-completeness results. Following [6, 8] we say that for languages $A, B \subseteq \Sigma^*$, a function f many-one reducing A to B is a DLOGTIME reduction if (1) f increases the length of strings only polynomially ($|f(x)| \leq p(|x|)$ for a polynomial p), and (2) some DLOGTIME Turing machine can decide given an input of (c, i, x), whether the ith symbol of $f(x)$ is c. We write $A \leq^{DLT} B$, if there exists a DLOGTIME reduction reducing A to B. (Recall that it is customary in the case of Turing machines operating in sublinear time, to access

the input via a special input index tape, for more details on DLOGTIME Turing machines see for example [4].)

- NC^1 many-one reducibility ($\leq_m^{\mathrm{NC}^1}$)

We use this reducibility for the L-completeness and all other results. Here the many-one reduction f is required to be computable in FNC^1.

2.2 Trees and representations

All trees discussed in this paper are rooted (hence implicitly directed) and unordered (i.e. the ordering of the descendants of a node does not matter). In some of our constructions we use colored trees. A tree with n nodes is said to be *colored* if each node in the tree is labelled with a positive integer no greater than n. An *isomorphism* between two colored trees T_1 and T_2 is a bijection between their node sets which maps the root of T_1 to the root of T_2, preserves the colors, and preserves the edges. Two colored trees are isomorphic, denoted $T_1 \simeq T_2$, iff an isomorphism exists between them. These notions apply mutatis mutandis to non-colored trees. The main computational problem of interest in this paper is:

[Colored] Tree Isomorphism ([C]TI)
Given: Two [colored] trees T_1 and T_2.
Problem: Determine whether $T_1 \simeq T_2$.

We consider two different representations for encoding trees: the string representation and the pointer representation. As we will see, in the small complexity classes we are dealing with, the representation used might change the complexity of the problem.

In the *string representation* [18], trees are represented over an alphabet containing opening and closing parentheses. A string representation of a tree T is defined recursively in the following way: The tree with a single node is represented by the string "()", and if T is a tree consisting of a root and subtrees $T_1, \ldots, T_k$ (in any order), with string representations $\alpha_1, \ldots, \alpha_k$, a representation of T is given by "$(\alpha_1, \ldots, \alpha_k)$". Observe that a tree might have different string representations, depending on the order of the descendants of any of its nodes. Colored trees can be encoded in the same way using colored parentheses. Let C be the set of colors. An opening parenthesis in a colored tree is represented by "(" followed by $\log(|C|)$ bits encoding the color. Note that we only need to color the opening parentheses.

The *pointer representation* is a more standard way to encode graphs. In this case we consider the trees given by a list of pairs of nodes representing directed edges. As in the previous case, if we deal with colored trees, with the representation of a node we include $\log(|C|)$ bits to encode its color.

For many tree problems in NC^1 and L, completeness results seem to depend on the representation used. For example, the reachability problem on forests, which

is L-complete in the pointer representation [10], can be solved in NC^1 (and even TC^0) in the string representation. Analogously, the Boolean formula value problem, which is complete for NC^1 in the string representation [7], becomes L-complete when described using trees given in the pointer representation [5]. In fact, changing from pointer to string representation is FL-complete [13].

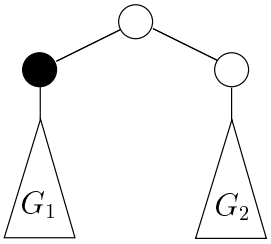
Another important observation is that it is possible to test within the classes NC^1 and L whether a given input is a correct encoding of a tree in the string representation, and respectively, in the pointer form.

3 Tree Isomorphism for Trees Given by Strings

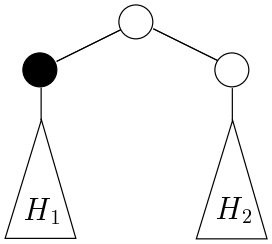
We show first that deciding whether two trees given in string representation are isomorphic is NC^1-complete. We first consider colored trees for which the hardness proof is simpler.

Lemma 3.1 *In the string representation, CTI is NC^1-hard under $\leq^{DLT}$.*

Proof. Following Barrington, Immerman and Straubing [4, Lemma 6.2], it suffices to reduce the problem of evaluating a balanced DLOGTIME-uniform family of Boolean expressions made up of alternating layers of ANDs and ORs, to CTI. The core of the reduction is the simple construction from [16, 9] described in [14] (page 45), for the purpose of simulating ANDs and ORs using graph isomorphism questions. We adapt this construction as follows. Consider four colored trees G_1, G_2, H_1, and H_2. Then the colored trees

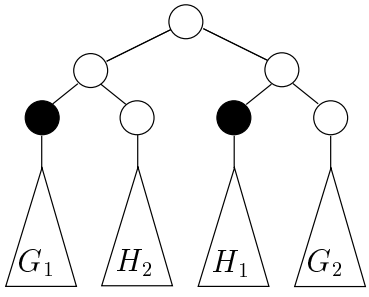


Tree $G_\wedge$

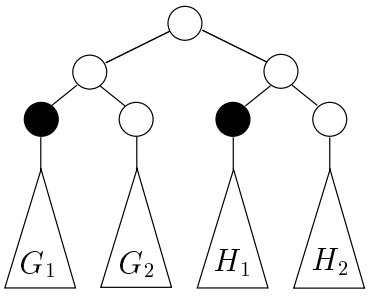


Tree $H_\wedge$

have the property that $G_\wedge \simeq H_\wedge$ iff $[(G_1 \simeq G_2) \wedge (H_1 \simeq H_2)]$, and the colored trees



Tree $G_\vee$



Tree $H_\vee$

have the property that $G_\vee \simeq H_\vee$ iff $[(G_1 \simeq G_2) \vee (H_1 \simeq H_2)]$. Observe that the OR-construction doubles the size of the initial trees for $G_1 \simeq G_2$ and $H_1 \simeq H_2$, that is, from 4 trees each having the same size, s, the OR-construction would produce 2 trees each having size $4s + 7$.

Now pick two single-node trees T_1 and T_2 and assign them different colors. Starting from the CTI instance (T_1, T_1) to represent a TRUE input and the CTI instance (T_1, T_2) to represent a FALSE input, it is trivial to construct, from a depth-d Boolean expression f with no negation gates, two colored trees G and H having $O(4^d)$ nodes (this is a rough upper bound) and verifying the property that $G \simeq H$ iff f evaluates to TRUE.

To see that this yields a DLOGTIME reduction, note it is possible to add a few dummy nodes in order to modify the constructs above in such a way that, if G_1, G_2, H_1 and G_2 are full binary (colored) trees, then so are $G_\wedge$, $H_\wedge$, $G_\vee$ and $H_\vee$, and moreover, the color occurrences in these respective constructs are the same. Because the Boolean expression is balanced, its depth is $O(\log n)$. And because the expression is built from alternating levels of ANDs and ORs, the argument is completed as in the proof of [4, Lemma 6.2]. ■

Remark. The complete simulation used in the proof of Lemma 3.1 in fact requires only two distinct colors. A similar construction could be devised in the absence of colors as well.

Lemma 3.2 *In the string representation, CTI* $\leq^{DLT}$ *TI.*

Proof. The obvious idea is to simulate the colors by attaching color-dependent gadgets at each node. Suppose that the trees in the CTI instance have n nodes. Then it suffices to attach at each c-colored node, $1 \leq c \leq n$, a new node which is root to a height-one subtree having $n + c$ leaves. In detail, at the string encoding level, the color binary number c occurring after the opening bracket which specifies the occurrence of the c-colored node is simply replaced with the encoding of the c-color gadget. To ensure DLOGTIME-computability, the color gadgets are modified to contain an identical number of nodes: it is easy to implement the idea with n non-isomorphic gadgets each having $2n + 1$ nodes. ■

Theorem 3.3 *In the string representation, CTI and TI are* NC^1*-complete under* $\leq^{DLT}$.

Proof. CTI is NC^1-hard (Lemma 3.1) and CTI $\leq^{DLT}$ TI (Lemma 3.2). Buss [7] shows in a delicate argument that TI $\in$ NC^1. (Buss in fact points out that his NC^1 algorithm applies directly, as well, to the case of labelled trees.) Hence CTI is NC^1-complete.

Since $\leq^{DLT}$ is in general not transitive, NC^1-hardness of TI needs to be argued by combining the reductions of Lemma 3.1 and Lemma 3.2, or by adapting the proof of Lemma 3.1 to the case of trees without colors. Hence TI is NC^1-complete. ■

4 Tree Isomorphism for Trees Given by Pointers

Recall Lemma 3.2, which states that in the string representation, colored tree isomorphism reduces to tree isomorphism by the introduction of appropriate gadgets for the colors. These gadgets are clearly NC^1-computable in the pointer representation, proving the following:

Lemma 4.1 *In the pointer representation, CTI* $\leq_m^{NC^1}$ *TI.* ■

Theorem 4.2 *In the pointer representation, TI is L-complete under many-one* NC^1*-reducibility.*

Proof. The containment follows from Lindell [15], who shows that a canonical form $c(T)$ of a (rooted) tree T can be computed in logspace. Hence, for two given trees, we determine isomorphism by computing and comparing their canonical forms symbol by symbol. (Alternatively, the string representation of the trees could be computed in L, and then Buss's NC^1 algorithm could be used.)

We prove hardness of TI by reducing the L-complete problem ORD [11] to Colored Tree Isomorphism, and appealing to Lemma 4.1:

Order between Vertices (ORD)

Given: A digraph $G = (V, E)$ that is a line, and two nodes $v_i, v_j \in V$.
Problem: Decide whether $v_i < v_j$ in the total order induced on V.

Let a line graph $G = (V = \{v_1, \ldots, v_n\}, E)$ and two designated nodes v_i and $v_j \in V$ be given. We assume without loss of generality that v_n is the outdegree zero node and that v_i, v_j and v_n are three different nodes. We first determine whether $v_i < v_j$ in the order induced by E with the help of oracle queries to CTI. Later we will show that the oracle queries can be merged to yield a many-one reduction.

We consider the instances $(T', T_{k,l})$ of the CTI problem. Here T' is the colored tree that results from G by coloring the node v_i with color 1, v_j with color 2, v_n with color 3, and the rest of the nodes with color 0.

For $1 \leq k < l < n$, the colored tree $T_{k,l}$ is defined as $(V, \{(v_m, v_{m+1}) | 1 \leq m < n\})$, with the nodes v_k, v_l and v_n colored with colors 1,2 and 3 respectively, and the rest of the nodes colored by 0.

It is not hard to see that $v_i < v_j$ in the order induced on V by G if and only if $\exists k, l$ with $1 \leq k < l < n$ such that $T' \simeq T_{k,l}$. This therefore describes a disjunctive reduction to CTI. In order to transform it into a many-one reduction, we use the OR-operator from Lemma 3.1 to combine all the $O(n^2)$ many queries to CTI into a single one. The OR-operator has to be applied carefully since it doubles the size of the trees involved every time it is used, and we are considering the disjunction of a polynomial number of trees. However, the operator can be applied in a divide and conquer manner and the resulting pair of trees has size polynomial in n (see [14]

page 50 for the details). Putting the two parts of the reduction together we have that ORD is NC^1 many-one reducible to CTI. ■

Remark. The reduction used in Theorem 4.2 can be made much finer than $\leq_m^{\mathrm{NC}^1}$, but obtaining the tightest possible reduction is not the focus of this note.

5 Concluding remarks

Complementing the harder results of Buss and Lindell, we have shown that tree isomorphism, depending on the tree representation, captures two robust complexity classes, namely NC^1 and L. These are the first hardness results for a natural restriction of the graph isomorphism problem, for which so far no hardness result was known. A modest L-hardness result for GI is implied by our L-hardness result for the tree case. An interesting open question is whether the isomorphism problem for general graphs is also hard for NL or even for P.

The level of sophistication of Buss's NC^1 algorithm for TI [7] is comparable to that of his simplified NC^1 algorithm for the Boolean expression value problem FVP [6]. Are these two upper bounds independent? In other words, is there a reduction from TI to FVP or vice versa which is simpler than either of Buss's two upper bounds?

It is interesting to consider FVP $\leq_m^{\mathrm{NC}^1}$ TI. Proving that FVP $\leq_m^{\mathrm{NC}^1}$ TI has required three ingredients: (1) the NC^1 upper bound for FVP, (2) the characterization of NC^1 in terms of balanced Boolean expressions, and (3) our simple Lemma 3.1. Lemma 3.1 directly constructs trees from Boolean formulas, but the ensuing direct reduction is from *Balanced*-FVP to TI. How can Lemma 3.1 be strengthened?

The bottleneck to a strengthening of Lemma 3.1 is the handling of a Boolean OR. Lemma 3.1 can only handle balanced Boolean expressions because the trees $G_\vee$ and $H_\vee$ depicted in its proof each require a copy of G_1, G_2, H_1, and H_2. Hence an open question is whether Lemma 3.1 can be proved using simpler constructs $G_\vee$ and $H_\vee$, still simulating the Boolean OR, but only adding a small number additional nodes. If so, the NC^1 upper bound for FVP is redundant, i.e., the NC^1 upper bound for FVP follows from the NC^1 upper bound for TI.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[2] L. Babai. Moderately exponential bounds for graph isomorphism. In *Fundamentals of Computation Theory 81*, Lecture Notes in Computer Science #117, Springer-Verlag, pp. 34–50, 1981.

[3] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. System Sci.*, 38:150–164, 1987.

[4] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within $\mathcal{NC}^1$. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[5] M. Beaudry and P. McKenzie. Circuits, matrices and nonassociative computation. *Journal of Computer and System Sciences*, 50(3):441–455, 1995.

[6] S. R. Buss. The Boolean formula value problem is in ALOGTIME. In *19th Annual ACM Symposium on Theory of Computing*, 123–131, 1987.

[7] S. R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Computational Logic and Proof Theory, 5th Kurt Gödel Colloquium'97*, Lecture Notes in Computer Science #1289, Springer-Verlag, 1997, pp. 18-33.

[8] S. R. Buss and S. A. Cook and A. Gupta and V. Ramachandran, An optimal parallel algorithm for formula evaluation, *SIAM Journal on Computing*, 21:4, pp. 755-780, 1992.

[9] R. Chang and J. Kadin, On computing Boolean connectives of characteristic functions. *Math. Systems Theory* 28, 173–198, 1995.

[10] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8:385–394, 1987.

[11] K. Etessami. Counting quantifiers, successor relations, and logarithmic space. In *Proc. of the 10th Structure in Complexity Theory Conf.*, pages 2–11. IEEE, 1995.

[12] J. E. Hopcroft and R. E. Tarjan. A V^2 algorithm for determining isomorphism of planar graphs. *Information Processing Letters*, 32–34, 1971.

[13] B. Jenner. *Between NC^1 and NC^2: Classification of Problems by Logspace Resources.* Manuskript of Habilitation thesis, 1997.

[14] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem — Its Structural Complexity.* Progress in Theoretical Computer Science. Birkhäuser, Boston, 1993.

[15] S. Lindell. A logspace algorithm for tree canonization. In *Proc. of the 24th STOC*, 400–404. ACM, 1992.

[16] A. Lozano and J. Torán. On the nonuniform complexity of the Graph Isomorphism problem. *Complexity Theory: Current Research*, pp. 245–273, 1993. Edited by K. Ambos-Spies, S. Homer, and U. Schöning. Also in *Proceedings of the 7th Structure in Complexity Theory Conference*, pp. 118–129, June 1992.

[17] E. Luks. Isomorphism of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25:42–65, 1982.

[18] G.L. Miller and J.H. Reif. Parallel tree contraction part 2: Further applications, *SIAM Journal on Computing*, 20:1128–1147, 1991.