Tree Isomorphism and Some Other Complete Problems for Deterministic Logspace

Birgit Jenner* Universität Ulm

Klaus-Jörn Lange† Universität Tübingen

Pierre McKenzie‡ Université de Montréal

March 27, 1997

Abstract

Several new tree problems are shown complete for deterministic logarithmic space. These include the tree centroid problem and the tree isomorphism problem, which thus becomes the first isomorphism problem of a combinatorial nature shown complete for a fundamental resource-based complexity class. The crucial role of the input representation of trees, as edge lists or as bracketed expressions, on the hardness of tree problems, is also discussed.

Résumé

Nous démontrons que certains problèmes concernant des arbres sont complets pour la classe de complexité L formée des langages reconnus en espace logarithmique. Un de ces problèmes est celui de déterminer si deux arbres sont isomorphes. Il s'agit donc dans ce cas d'un premier problème d'isomorphisme à caractère combinatoire dont on puisse démontrer la complétude pour une classe de complexité robuste définie par bornes de ressources. Un autre problème dont nous prouvons la L-complétude est celui de déterminer si le retrait d'un sommet donné d'un arbre de n sommets résulte en sous-arbres d'au plus $\lceil \frac{n}{2} \rceil$ sommets. Nous signalons finalement l'effet du choix de la représentation des arbres sur la complexité de certains problèmes de calculs de la classe L.

*Abt. Theoretische Informatik, Universität Ulm, Oberer Eselsberg, 89075 Ulm. During academic year 1996-97 on vacation from Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D–72076 Tübingen, Germany. E-mail: `jenner@informatik.uni-ulm.de`

†Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D–72076 Tübingen, Germany. E-mail: `lange@informatik.uni-tuebingen.de`

‡Dép. d'informatique et recherche opérationnelle, Université de Montréal, C.P. 6128, Succursale Centre-ville, Montréal (Québec), H3C 3J7 Canada. Work supported by NSERC of Canada and by FCAR du Québec. E-mail: `mckenzie@iro.umontreal.ca`

1 Introduction

The parallel complexity class NC defined by uniform polynomial-size circuits with polylogarithmic depth characterizes those problems in P that in principle are efficiently parallelizable, as opposed to the problems that are P-complete. A long-known subclass of NC is the class L of problems solvable deterministically in logarithmic space. The class L lies "between" the class NL, nondeterministic logarithmic space, and the class NC^1 defined by NC circuits of logarithmic depth:

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC \subseteq P.$$

It is suspected but not known that the five above inequalities are proper.

The class L is robust. Indeed, L was defined above using the Turing machine, but L is also characterized by k-head finite-automata, by (uniform) polynomial-size deterministic branching programs [20], and by logical formalisms [13]. Furthermore, L and its functional variant FL, the class of functions computable in logspace, have a variety of natural complete problems arising from formal language theory, from algebra, from graph theory, etc. [16, 6, 8, 9]. An up-to-date collection of L- and FL-complete problems is in preparation [15].

In this paper we add four graph-theoretic problems to the list of L-complete problems. These are the isomorphism problem for trees, the node subtree isomorphism problem, the tree size problem, and the centroid problem. We also discuss functional variants of these problems that are FL-complete.

It was shown by Lindell [19] that tree isomorphism (TI) is contained in L. We show here that TI is actually complete for L with respect to (disjunctive) AC^0-Turing reductions and that the node subtree isomorphism problem NSTI is complete for L with respect to AC^0-many-one reductions. TI thus becomes one the select few isomorphism problems found *complete* for a robust complexity class. To the best of our knowledge, the one example of a complete isomorphism problem is the isomorphism problem for two given subgroups of a free group, shown P-complete by Avenhaus and Madlener [2]. Isomorphism problems defined from computational models, as for example, Boolean circuits or branching programs, are not known to be complete for a complexity class (see for example the introduction of [22]). A more prominent example, the graph isomorphism problem, is trivially contained in NP but is probably not NP-hard (see [18]). Indeed it seems that the existential quantifier underlying the definition of NP is stronger than the existential quantifier implicit in the graph isomorphism problem. This "weakness" of the problem can be compensated by considering the *sub*graph isomorphism problem, which is NP-complete [17]. Although L is not a typical nondeterministic class, our results on tree isomorphism reflect a somewhat similar situation: it seems that the existential quantifier implicit in TI is too weak to capture the whole complexity of L via many-one reducibility, but this lack of problem complexity can be compensated by a further existential quantification in the disjunctive Turing reduction. Alternatively, this lack of complexity of the tree isomorphism problem can be compensated by considering the *sub*tree isomorphism problem NSTI.

An interesting restriction of NSTI is the problem of whether the subtree rooted at a given node of a tree (or forest) has a given size. We also show this problem to be L-complete.

The fourth problem that we prove L-complete is the centroid problem. A centroid of a tree with n nodes is a node whose deletion leaves no subtree containing more than $\frac{n}{2}$ nodes ([24]). In 1869 Jordan showed that any tree has a centroid (see [12]). The computation of centroids is a tool of many algorithms of practical relevance (see e.g. [10]).

All the L-complete problems that we present are defined on directed (and hence rooted) trees, but can be extended to undirected trees or forests (i.e., undirected acyclic graphs). We assume here that the trees are given via a listing of the edge relation, i.e., a list of pairs of vertices. This presentation seems to be essential for the completeness results. Buss and Lindell (personal communication, 1996) have shown that binary tree isomorphism is already in NC^1 when the (binary) trees are given as bracketed expressions. A similar situation arises in the case of some other L-complete problems: For example, reachability between nodes of forests is L-complete when the forest is presented via its edge relation, but solvable by counting opening and closing brackets, and hence in NC^1(in fact TC^0), if the forest is presented via a bracketed expression.

The paper is organized as follows. In Section 2, we define the reducibilities used in this paper and we recall some well-known L-complete problems. In Section 3, we discuss the representation of graphs and trees in more detail, we recall some other known L-complete problems, and we show that our problems, when defined properly, are not promise problems. Section 4 contains our completeness results for the various isomorphism problems. In Section 5 we show L-completeness for the tree size problem, and in Section 6 L-completeness for the centroid problem.

2 Reducibilities and Some Known L-Complete Problems

In this section we present the reducibilities and a first group of L-complete problems required in later sections.

Reducibilities

There are a variety of reducibility notions suitable for logspace classes. These notions vary in the amount of resources they consume and in the the way the reduction is computed. The suitable reducibilities must consume less (or supposedly less) resources than logspace. Hence the most common resources chosen are NC^1, AC^0, DLOGTIME, and quantifier-free projections. Here $\mathrm{NC}^1(\mathrm{AC}^0)$

denotes the class of functions computable with DLOGTIME-uniform (semi-unbounded fan-in) Boolean circuit families of polynomial size and logarithmic (constant) depth (see e.g. [3]). DLOGTIME is the class of problems computable by logarithmic time-bounded Turing machines (see e.g.[1]); for quantifier-free projections see [14].

For our completeness results, we will use two variants of AC^0-reducibility, namely, AC^0-many-one reducibility and AC^0-Turing reducibility. The former will be used only for reducing between languages, that is, for decision problems, and the latter will serve for both decision and functional problems.

Formally, *AC^0-many-one reducibility* is defined as follows: For problems $A, B \subseteq \Sigma^*$, $A \leq_m^{\mathrm{AC}^0} B$ holds if there exists a function $f \in \mathrm{AC}^0$ having the property that, for all $x \in \Sigma^*$, $x \in A$ iff $f(x) \in B$.

In the Turing case, the reduction is realized by AC^0-circuits which contain so-called oracle gates that solve the incoming problem instances at unit cost: The size and depth of any such gate is one (see e.g. [23]). For NC^1 this reducibility was introduced by Cook [6, 7]. Turing reducibility is typically used when functional classes like FL are considered, since it allows interreducing functional and decisional problems easily (see e.g. [8]).

More formally, *AC^0-Turing reducibility* is defined as follows: For problems $A, B \subseteq \Sigma^*$, we have $A \leq_T^{\mathrm{AC}^0} B$, if there exists an AC^0-circuit family $\{C_n\}$ with oracle gates for B that computes A.

Special cases of AC^0-Turing reducibility result from restricted the reducing circuits. *Disjunctive AC^0-Turing reducibility* is obtained by restricting the reducing circuit family $\{C_n\}$ to consisting of $\vee$-gates. One last special case is relevant to the study of classes L or FL. Any of the reductions given by Cook and McKenzie in [8] between L- or FL-complete problems are in fact such that in the reducing AC^0-circuits, any path from the input gates to the output gate(s) contains at most one oracle gate.[1] This is the special notion of AC^0-reducibility used by Chandra, Stockmeyer, and Vishkin in [5].

Throughout this paper, unless specifically mentioned otherwise, L-completeness will refer to $\leq_m^{\mathrm{AC}^0}$, and FL-completeness will refer to $\leq_T^{\mathrm{AC}^0}$.

Some L-Complete Problems

L has many natural complete problems from different areas including formal language theory, algebra, and graph theory. A partial list of such problems can be found in [8], and an up-to-date collection will appear in [15]. Here we recall some of the examples which we will require later in our hardness proofs. We adopt the presentation format used by Greenlaw, Hoover, and Ruzzo [11].

[1]In fact, Cook and McKenzie use NC^1-circuits instead of AC^0-circuits. But with the exception of the reduction DFA $\leq$ BFS, where counting and thus at least TC^0 is needed, any of their reductions can be computed by AC^0 circuits.

Directed Two Tree Accessibility (DTTA)

Given: A digraph consisting of two directed trees T_{good}, T_{bad} with roots *good* and *bad*, respectively, and a node t.
Problem: Determine whether t belongs to the tree with root *good*.
Classification: L-complete.
Reference: [8]
Hint: This is basically the problem Directed Forest Accessibility (DFA) of [8], where for a given acyclic directed graph G of outdegree zero or one with exactly two weakly connected components, and nodes u and v, it is required to determine whether there is a path between u and v. Simply reverse the edges in G to obtain two directed (or rooted undirected) trees and choose u to be the root of T_{good}.
Remarks: The trees can also be unrooted. This is the problem UFA of [8].

Iterated Multiplication in S_n (ITS$_n$)

Given: Permutations $p_1, \ldots, p_k$ of S_n presented pointwise as $(p_i(1), \ldots, p_i(n))$, and two designated elements i', j' with $1 \leq i', j' \leq n$.
Problem: Is i' mapped to j' by the product of permutations $\Pi_{i=1}^{k} p_i$?
Classification: L-complete.
Reference: [8, 14]
Remarks: In [8] this problem is called Permutation Product (PP). PP is the functional variant that computes the resulting permutation. Hardness is proved with respect to NC^1-Turing reducibility. [14] proves hardness with respect to quantifier-free projections.

Order between Vertices (ORD)

Given: A digraph $G = (V, E)$ that is a line, and two nodes $i, j \in V$.
Problem: Determine whether $i < j$ in the total order on V induced by G.
Classification: L-complete.
Reference: [9]
Remarks: Hardness proved with respect to quantifier-free projections. Observe that it can be checked in L whether a digraph is a line (see problem IL of Section 3).
Hint: For containment, an L-machine simply traverses along the unique outgoing edges starting from the unique node with indegree zero to check whether node s is reached before node T. For hardness, reduce from ITS$_n$.

3 Representations of Graphs and Trees

Several known L-complete problems involve graphs like forests, trees or lines. We use the word "forest" in the context of both directed and undirected graphs. When applied to an undirected graph, a forest is simply an acyclic graph. When applied to a directed graph, a forest is an acyclic directed graph with maximal indegree one. A tree is a forest with one connected component, and a line is a graph that consists of a sequence of edges. Now, acyclicity is only known to be computable in L for undirected graphs (in fact, it is L-complete) [6]. For directed graphs acyclicity is well-known to be NL-complete [16]. Are all the problems involving directed graphs below therefore *promise problems*[2]? The answer is no, since the indegree conditions on the directed graphs input to these problems reduce testing acyclicity to testing acyclicity in undirected graphs. This is made precise in the comments accompanying the problem definitions below.

Is Forest (IF)
Given: A (di)graph $G = (V, E)$.
Problem: Determine whether G is a forest, i.e., an acyclic undirected graph or an acyclic directed graph with maximal indegree one.
Classification: L-complete.
Reference: [15]
Hint: For undirected graphs, G is a forest iff G is acyclic, that is, G does not contain any cycles, which is the L-complete problem CYCLE of [6]. For directed graphs, we can reduce IF to CYCLE, since the indegree one bound ensures that removing the directions of the edges does not increase the number of cycles. Also, the graph constructed for hardness in [6] obeys the indegree bound.

Is Tree (IT)
Given: A (di)graph $G = (V, E)$.
Problem: Determine whether G is a tree.

[2]A promise problem is the formulation of a decision problem that has the structure (x, Q, R), where x is the input (for example, a graph G), Q is a promise made about x (for example, that G is a tree), and R is the property that has to be decided by the algorithm (for example, whether the tree has a particular depth). An algorithm solving the promise problem is only required to decide $R(x)$ under the assumption that x in fact satisfies predicate Q, and it may answer arbitrarily on inputs for which $\neg Q(x)$ (see [21] or [18, p. 39]). The predicate Q is appropriately called a promise only when the complexity of Q is not known to be bounded by the complexity of R. In our case, if Q specifies a property that is in fact checkable within logspace, then the relevant problem is in reality not a promise problem.

Classification: L-complete
Reference: [15]
Hint: For containment, we check acyclicity and the existence of at most one connected component. The latter problem, 1CC, is L-complete as shown in [15]. Hardness follows from the hardness of the problem Single Cycle Permutation of [8] by simply removing one edge. If there was a single cycle, then the result is a line, which is a tree; if there were two or more cycles, then removing an edge leaves at least one cycle and thus no tree.

Is Line (IL)
Given: A digraph $G = (V, E)$.
Problem: Determine whether G is a line graph, i.e., a sequence of nodes $v_1, \ldots, v_n$ such that $E = \{(v_i, v_{i+1}) \mid 1 \le i \le n-1\}$.
Classification: L-complete.
Hint: For containment, just check acyclicity and the necessary in- and outdegree conditions. Hardness follows with the same reduction as for IT above.

The functional variant of IL is the following problem.

Compute Line (CL)
Given: A (di)graph $G = (V, E)$ that is a line.
Problem: Compute the sequence of nodes of G in the total order on V induced by G, that is, starting with the indegree zero node up to the outdegree zero node.
Classification: FL-complete.
Reference: [15]
Hint: For containment, just check that there is no cycle in G. For hardness, reduce IL to CL.

Unless specified otherwise, we assume that all forests or trees in our problems have specified roots (which always is ensured if the edges are directed). Our forest or trees are presented via a listing of the edge relation, that is, by all the pairs of nodes that define an edge of T, which is one of the common representations of graphs. If not specified, trees are either directed or undirected.

Alternative representations for forests or trees are obtained using bracketed expressions. Let $<$ denote the (total) lexicographical order $<$ between strings over $\{0, 1\}$ (namely the names of the nodes of the tree). The (lexicographically first) *bracketed expression* $E(T)$ of a tree T is a string over $\{[,], 0, 1\}$ defined inductively as follows:

1. For a one node tree $v \in \{0, 1\}$ the bracketed expression is $[v]$.
2. For a tree T with root v and subtrees $T_1, \ldots, T_k$ with roots $v_1, \ldots, v_k$ such that $v_1 < \ldots < v_k$, the bracketed expression is $[v\, E(T_1) \cdots E(T_k)]$.

The bracketed expression $E(F)$ of a forest F that consists of trees $T_1, \ldots, T_k$ with roots $v_1, \ldots, v_k$ such that $v_1 < \ldots < v_k$ is simply the concatenation of the bracketed expressions of its trees in lexicographical order of the roots, that is, $E(F) = E(T_1) \cdots E(T_k)$.

L-completeness results seem to depend on the presentation of the forests or trees. Many known L-complete graph problems, like for example, reachability between nodes of a forest as in the problem DTTA, are solvable in NC^1 (or even TC^0) if the input forest F is given by a bracketed expression $E(F)$, because then we can find the subtree containing the designated node by simply counting opening and closing brackets in $E(F)$. Similarly, Buss and Lindell (personal communication, 1996) have shown that binary tree isomorphism is already in NC^1 when the (binary) trees are given as bracketed expressions. Vice versa, the NC^1-complete Boolean Formula Value Problem becomes L-complete, when defined over trees that are presented via pairs of nodes (see [4, p. 445] or [15]). Accordingly, changing the representation of forests or trees is itself FL-complete:

Compute Bracketed Expression (CBE)
Given: A forest $F = (V, E)$.
Problem: Compute the bracketed expression $E(F)$ of F.
Classification: FL-complete.
Reference: [4, 15]
Remarks: Hardness is via NC^1-Turing reducibility. Transforming the bracketed expression into a list of the edge relation is solvable in TC^0.

Cook and McKenzie [8] pointed out similar representation-dependent complexity variations for permutation problems and showed that the transformation of a permutation π presented pointwise (as $\pi(1), \ldots, \pi(n)$) to the disjoint cycle representation of π is L-complete, while the reverse direction is computable in NC^1 (in fact, it is computable in AC^0).

4 Tree Isomorphism

First we define tree isomorphism and subtree isomorphism formally (see also [19]). Let S and T be two trees. We say that $S < T$ in *isomorphism order*, if

1. $|S| < |T|$, i.e., the size of S (the number of nodes in S) is smaller than the size of T, or

2. $|S| = |T|$ and $\#child(s) < \#child(t)$, i.e., the number of children of the root s of S is smaller than the number of children of the root t of T, or

3. $|S| = |T|$, $\#child(s) = \#child(t) = k$, and $(S_1, \ldots, S_k) < (T_1, \ldots, T_k)$ lexicographically, where (it is inductively assumed that) $S_1 \leq \ldots \leq S_k$ and $T_1 \leq \ldots \leq T_k$ are the ordered subtrees of S and T.

If neither $S < T$ nor $T < S$, S and T have the same isomorphism order $S = T$, and are said to be *isomorphic* ($\hat{=}$).

As a canonical representation of any tree isomorphism class we can use so called *tree canons*, i.e., an infix notation of trees, where the embedded subexpressions are ordered. The *canon $c(T)$ of a tree T* is a string over $\{[,],*\}$, that provides a name for T that is invariant under isomorphism. $c(T)$ is defined inductively as follows:

1. The canon of a one node tree is $*$.

2. The canon of a tree T with subtrees ordered $T_1 \leq \ldots \leq T_k$ is $[c(T_1) \cdots c(T_k)]$.

For example, an edge between two nodes has the canon $[*]$, a line graph with three nodes has the canon $[[*]]$, a parent-two-children fork has the canon $[**]$, and the tree $((1,2),(1,3),(2,4),(2,5),(5,6),(5,7))$ has the canon $[*[*[**]]]$.

The following problem was shown by Lindell [19] to be contained in L.

Tree Isomorphism (TI)
Given: Two trees T_1 and T_2.
Problem: Determine whether T_1 is isomorphic to T_2.
Reference: [19]
Remarks: Lindell shows that for a given tree the canon can be computed in logspace (see problem CC below), which yields TI $\in$ L. TI can be generalized to the problem Forest Isomorphism, if forests instead of trees are given. Lindell showed that FI is also contained in L.

We first show

Theorem 4.1 *TI is L-complete with respect to disjunctive $\leq_T^{\mathrm{AC}^0}$-reducibility.*

Proof. For containment, see the article of Lindell [19] who shows that the canon $c(T)$ of a (rooted) tree T, can be computed in logspace. Hence, for two given trees we determine isomorphism by computing and comparing their canons symbol by symbol.

For hardness, we reduce ORD, which is L-complete (see Section 2) to TI as follows. Let a line graph $G = (V = \{v_1, \ldots, v_n\}, E)$ and two designated nodes

$s = v_i, t = v_j \in V$ be given. Let v_1, v_n denote the first (i.e., indegree zero) and respectively, last node (i.e., outdegree zero) node in V. We determine whether $i < j$ with the help of oracle queries to TI querying instances $(T', T_{i,j})$. Here $T' = (V \cup V', E \cup E')$ is the tree that results from G by adding six new nodes $v_i', v_j', v_j'', v_n', v_n'', v_n''' \in V'$ and the six edges:

$$E' := \{(v_i, v_i'), (v_j, v_j'), (v_j, v_j''), (v_n, v_n'), (v_n, v_n''), (v_n, v_n''')\}.$$

This construction ensures that the tree rooted at v_1 is minimal with respect to isomorphism order (for the unrooted case). Furthermore, if $i < j$ then the canon of T' depends only on i and j. It is

$$[^i * [^{j-i} * * [^{n-j} * * *]^n.$$

If $i > j$ the canon of T' will be different, namely

$$[^i * * [^{j-i} * [^{n-j} * * *]^n.$$

Since we do not know the distances i, j, we define for $1 < i < j < n$ trees $T_{i,j} = (V_{ij}, E_{ij})$ with nodes

$$V_{ij} := \{1, \ldots, n\} \cup \{i^1, j^1, j^2, n^1, n^2, n^3\}$$

and edges

$$E_{ij} := \{(k, k+1) \mid 1 \leq k < n\} \cup \{(i, i^1), (j, j^1), (j, j^2), (n, n^1), (n, n^2), (n, n^3)\},$$

according to the canon in the case $i < j$. Now if $s = v_i < t = v_j$, then $T_{i,j}$ is isomorphic to T'; on the other hand, if $s = v_i > t = v_j$, no $T_{i,j}$ can be isomorphic to T'. That is, querying $(T_{i,j}, T') \in$ TI for all i, j such that $1 < i < j < n$, we can decide $(G, s, t) \in$ ORD. Hence, ORD is disjunctive $\leq_T^{\mathrm{AC}^0}$-reducible to TI. □

The proof shows that the problem Test Isomorphism Order, the special case of TI in which one of the two given trees is already a canon, is L-complete. But it seems that disjunctive Turing reducibility is the strongest reducibility we can expect for the L-completeness of Tree Isomorphism, since trees do not seem to be closed under "disjunctive changes". Many-one reducibility can be used to show completeness for the slightly more general problem NSTI, which incorporates in its definition the disjunctive reduction via the subisomorphism question.

Node Subtree Isomorphism (NSTI)
Given: Two trees T_1 and T_2.
Problem: Determine whether there is a node v of T_1 such that T_2 is isomorphic to the complete subtree of T_1 rooted at v.

NSTI is not as general as the

> **Subtree Isomorphism (STI)**
> **Given:** Two trees T_1 and T_2.
> **Problem:** Determine whether there is subgraph of T_1 that is isomorphic to T_2.

STI is known to be in solvable in $\mathrm{P} \cap \mathrm{RNC}$, but neither known to be P-complete [11] nor to be computable in NC. Our variant NSTI does not search for an arbitrary subgraph of T, but limits its quest to subgraphs that are *complete subtrees subtended by particular nodes* of T. Clearly, the NSTI question is an easier question, because the subgraphs relevant to NSTI have a very short description (of size $\log n$ for a tree with n nodes) that consists in their root only.

Theorem 4.2 *NSTI is L-complete with respect to $\leq_m^{\mathrm{AC}^0}$-reducibility.*

Proof. For containment, simply check for all nodes v of T_1 whether its subtree is isomorphic to T_2 using Lindell's algorithm that shows $\mathrm{TI} \in \mathrm{L}$ [19]. To mark the current node v only logspace is necessary, and for any node u it can be checked in logspace whether u belongs to the subtree of v.

For hardness, we reduce ORD to NSTI as in the proof of Theorem 4.1. As in this proof, let a line graph $G = (V = \{v_1, \ldots, v_n\}, E)$ and $s = v_i, t = v_j$ be given. But now we simulate the disjunctive Turing reduction to instances $(T', T_{i,j})$ of TI by just one query to the instance (T_{coll}, T') of NSTI, where T_{coll} is a complex directed tree that collects the trees $T_{i,j}$ of the proof of Theorem 4.1. More precisely T_{coll} consists of a root node v_{coll} that has the $\frac{(n-2)\cdot(n-3)}{2}$ equal-size trees $T_{i,j}$, $1 < i < j < n$, as subtrees. Then, by construction, if $i < j$ then the subtree $T_{i,j}$ is isomorphic to T'. On the other hand, if $i > j$, then no subtree T_v rooted at node v of T_{coll} can be isomorphic to T', because the only nodes that have subtree size equal to that of T' are the roots of the trees $T_{i,j}$, none of which is isomorphic to T'. $\square$

A slightly more complex construction yields the following corollary.

Corollary 4.3 *STI (for undirected trees) is $\leq_m^{\mathrm{AC}^0}$-hard for L.*

Proof. We again simulate the Turing reduction from the proof of Theorem 4.1, using a more complicated tree. Call this new tree T_{push}. T_{push} is as T_{coll}, but to ensure that no subgraph containing v_{coll} can be isomorphic to T', any one of the roots of the $T_{i,j}$ subgraphs is pushed away from v_{coll} by a chain of n nodes. Then, by construction, if $i < j$, the subtree $T_{i,j}$ is isomorphic to T'. But if $i > j$, no subtree of T_{push} that contains node v_{coll} can be isomorphic to T', because the distance of this node to any other node of outdegree at least three in T_{push} is too large; and similarly, the subtrees of T_{push} that do *not* contain v_{coll} are the graphs $T_{i,j}$, possibly enlarged by chains of up to n nodes, none of which results in a subtree that can be isomorphic to T'. $\square$

There is one prominent functional variant of Tree Isomorphism, namely the problem of computing the isomorphism order of a given tree (or a forest).

Compute Canon (CC)
Given: A tree T.
Problem: Compute the canon of T.
Reference: [19]
Remarks: [19] shows that CC $\in$ FL.

Proposition 4.4 *CC is FL-complete with respect to $\leq_T^{\mathrm{AC}^0}$ reducibility.*

Proof. Containment follows with Lindell [19]. For hardness, reduce TI to CC. Two oracle gates are queried in parallel for the canons of the given trees T_1, T_2. Afterwards, the canons are simply compared symbol by symbol. $\square$

5 Tree Size

One of the subproblems that arise when the canon of a forest F is computed is to determine the size of a subtree of F rooted at a particular node. In this section we show that this problem is in fact L-complete.

Tree Size (TS)
Given: A forest F, a node v of F, and a positive integer k.
Problem: Is k the size of the subtree of F rooted at v?

(The size of a tree T is the number of nodes in T.)
The functional variant of TS is the following problem.

Compute Tree Size (CTS)
Given: A forest F and a node v of F.
Problem: Compute $s(T_v)$, the size of the tree rooted at v.

Theorem 5.1 *TS is L-complete, and CTS is FL-complete.*

Proof. We only sketch the proof of the L-completeness of TS, which implies the proof of FL-completeness of CTS. For containment of TS in L, just compute a tour through F starting and ending in v to obtain the size of the tree rooted at v. For the L-hardness of TS, we reduce DTTA to TS as follows. For a given DTTA instance with trees T_{good} and T_{bad} and designated node t take three copies of T_{good} and T_{bad}, and construct a tree T with root r with successors r_+ and r_-, such that root r_+ has successors $good_1$ and bad_2 and root r_- has successors $good_2$ and bad_1. Furthermore, the designated node $t = t_1$ (that belongs either to the subtree of $good_1$ or bad_1) has successors $good_3$ and bad_3. Then the tree size corresponding to r_+ is $2n+1$, if t lies in T_{good} and is $n+1$, if t lies in T_{bad}, which reduces DTTA to TS. □

6 Centroid Problems

A centroid of a tree T of size n is a node of T that when deleted decomposes T into subtrees of size at most $\lceil \frac{n}{2} \rceil$ each ([24]). In 1869 Jordan showed that any tree has a centroid (see [12]). The computation of centroids is a tool in many algorithms of practical relevance (see e.g. [10]).

The following problems correspond to checking and computing centroids for a given tree.

Is Centroid (IC)
Given: A tree T and one of its nodes v.
Problem: Is v a centroid of T?

The functional version version of IC is:

Compute All Centroids (CAC)
Given: A tree T.
Problem: Compute all centroids of T.

The complexity of centroid problems is captured by the following results.

Theorem 6.1 *IC is L-complete with respect to* $\leq_m^{\mathrm{AC}^0}$ *reducibility.*

Proof. For containment, delete node v from T and determine the sizes of any of the resulting subtrees by computing for every node u different from v the size of a tour starting and ending in u. Accept, iff these sizes never exceed $\lceil \frac{n}{2} \rceil$.

For hardness, we reduce DTTA to IC. Let an arbitrary DTTA instance be given, that is, a digraph G with n nodes consisting of two trees T_{good}, T_{bad} and a node t. We will reduce G to a complex new tree T_c of size $4n$ that contains three copies of G and some additional nodes. Let G_1, G_2 and G_3 be the three copies of G. T_c has root r with three subtrees T_{r_+}, T_{r_-}, and T_{r_*}. The subtree T_{r_*} is a simple chain of $n-3$ nodes used to control the size of T. The subtree T_{r+} has root r_+ and the subtree T_{r-} has root r_- (later, these will be the two only possible centroids of T_c, depending on whether t belongs to the tree T_{good} or not). These two roots branch to G_1 and G_2 as follows: r_+ has children $good_1$ and bad_2, and r_- has children $good_2$ and bad_1. Finally, we connect node t_1 (the terminal node of G_1) to both $good_3$ and bad_3.

Now we claim that t is connected to *good* in the given instance G if and only if r_+ is a centroid of T_c. First observe that T_c has size $4n$ and deleting a centroid will leave only subtrees of size up to $2n$. Now, if t was in the *good* tree of G then T_{r+} will be of size $2n+1$, and hence r_+ is a centroid of T_c. (In fact, it is the only centroid of T_c, since its heaviest child $good_1$ isn't a centroid.) On the other hand, if r_+ is a centroid of T_c, then t, whose subtree (not counting t) has size n, can not be connected to bad_1, because the size of T_{r_-} must be bounded by $2n$. □

Theorem 6.2 *CAC is FL-complete with respect to $\leq_T^{\mathrm{AC}^0}$ reducibility.*

Proof. For containment, just cycle through all nodes v and check whether v is a centroid (see Problem IC and Theorem 6.1). If so, output v. For hardness, reduce from IC. After querying for the list of all centroids for T, just check whether the given node v is on the list. □

Acknowledgement

The first author thanks Eric Allender for inspiring discussions, and Raymond Greenlaw for pointing out to her the centroid problem.

References

[1] C. Àlvarez and B. Jenner. On adaptive dlogtime and polylogtime reductions. *Theoretical Computer Science*, 1974:175–193, 1995.

[2] J. Avenhaus and K. Madlener. The Nielsen reduction and P-complete problems in free groups. *Theoretical Computer Science*, 32:61–76, 1984.

[3] 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.

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

[5] A. K. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, May 1984.

[6] S. A. Cook. Towards a complexity theory of synchronous parallel computation. *L'Enseignement Mathématique*, XXVII(1–2):75–100, 1981.

[7] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1):2–22, 1985.

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

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

[10] H. N. Gabow. Centroids, representations, and submodular flows. *Journal of Algorithms*, 18:586–628, 1995.

[11] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *A Compendium of Problems Complete for P*. Oxford University Press, 1994.

[12] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[13] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.

[14] N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116:103–116, 1994.

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

[16] N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975.

[17] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

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

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

[20] C. Meinel. The power of nondeterminism in polynomial-size bounded-width branching programs. *Theoretical Computer Science*, 62:319–325, 1988.

[21] A. L. Selman. Promise problems and complexity classes. *Information and Computation*, 78:87–98, 1988.

[22] T. Thierauf. The isomorphism problem for one-time-only branching programs and arithmetic circuits. Technical Report ECCC-TR96-040, http://www.eccc.uni-trier.de/eccc/, 1996.

[23] C. B. Wilson. On the decomposability of NC and AC. *SIAM Journal on Computing*, 19(2):384–396, 1990.

[24] R. J. Wilson and J. J. Watkins. *Graphs: An Introductory Approach.* Wiley, New York, 1990.