ON THE HARDNESS OF GRAPH ISOMORPHISM[*]

JACOBO TORÁN[†]

Abstract. We show that the graph isomorphism problem is hard under DLOGTIME uniform AC^0 many-one reductions for the complexity classes NL, PL (probabilistic logarithmic space), for every logarithmic space modular class Mod_kL and for the class DET of problems NC^1 reducible to the determinant. These are the strongest known hardness results for the graph isomorphism problem, and imply a randomized logarithmic space reduction from the perfect matching problem to graph isomorphism. We also investigate hardness results for the graph automorphism problem.

1. Introduction. The graph isomorphism problem GI consists in the decision of whether two given graphs are isomorphic, or in other words, whether there is a bijection between the nodes of the graphs preserving the edges. This problem has been intensively studied, in part because of its many applications, and in part because it is one of the few problems in NP that has resisted all attempts to be classified as NP-complete, or within P. The best existing upper bound for the problem given by Luks and Zemlyachenko is $\exp\sqrt{cn\log n}$ (cf [7]), but there is no evidence of this bound being optimal, and for many restricted graph classes polynomial time algorithms are known. This is for example the case for planar graphs [19], graphs of bounded degree [29] or graphs with bounded eigenvalue multiplicity [6]. In some cases, like trees [28, 11] or graphs with colored vertices and bounded color classes [30], even NC algorithms for isomorphism are known.

Concerning the hardness of GI, there is evidence indicating that the problem is not NP-complete. On the one hand, the counting version of GI is known to be reducible to its decisional version [31], while for all known NP-complete problems the counting version seems to be much harder. On the other hand it has been shown that if GI were NP-complete then the polynomial time hierarchy would collapse to its second level [9, 36]. Because of these facts, there is a common belief that GI does not contain enough structure or redundancy to be hard for NP. The question of whether GI is P-hard is also open, and moreover, the known lower bounds in terms of hardness results for GI are surprisingly weak. It is only known that isomorphism for trees is hard for NC^1 and for L (logarithmic space) depending on the encoding of the input [23].

In this paper we improve the existing hardness results by showing that GI is hard for all complexity classes defined in terms of the number of accepting computations of a nondeterministic logarithmic space machine.

The key ingredient in the proof of our results, is a graph gadget showing that GI has enough structure to encode a modular addition gate. Using this fact, we are able to give for any ($k \in \mathbb{N}$) an AC^0 many-one reduction from the circuit value problem for addition mod k gates, which is complete for Mod_kL, to GI. Mod_kL is the complexity class corresponding to sets recognized by nondeterministic logarithmic space machines in which the number of accepting computations satisfies a congruence modulo k [10], and it lies within NC^2. We show that a circuit with modular gates can be directly transformed into a graph in which any automorphism of a certain kind maps a special vertex encoding the output gate to a vertex encoding the output of the circuit. The graphs used in the reduction have degree 3 and its vertices can be

[*]A preliminary version of this paper appeared in the conference FOCS 2000.

[†] Abteilung Theoretische Informatik, Universität Ulm, Oberer Eselsberg, 89069 Ulm, Germany, toran@informatik.uni-ulm.de

partitioned into color-classes of size k^2. Luks [30] has given an NC upper bound for the complexity of the isomorphism problem restricted to graphs with bounded color classes. For isomorphism in this class of graphs, the gap between our hardness results and the upper bound given by Luks is therefore small. In fact, in [27] we have recently shown that for graphs of bounded color classes of size 2 and 3, the graph isomorphism problem is complete for symmetric logarithmic space.

By a simple use of the Chinese Remainder Theorem, the hardness results for the modular classes can be transformed into hardness results for NL. It is interesting to observe that the graphs obtained in this reduction have automorphism groups in which the size of the orbits of some of the nodes depend on the input size, and therefore these graphs do not have classes of colored vertices of constant size as in the modular case.

Using the recent result that division can be performed in TC^0 [17, 18], and the fact that an NC^1 circuit can be encoded in an isomorphism problem [23], we can moreover prove that any logarithmic space counting function can be reduced to GI. In particular this implies that GI is many-one hard for $\mathrm{C}_=\mathrm{L}$ and for probabilistic logarithmic space, PL. The hardness results culminate in Theorem 4.9 where it is shown that GI is hard for DET, defined by Cook [13] as the class of problems NC^1 Turing reducible to the determinant.

Perfect matching problem is (as GI) another problem of the short list that has resisted classification in terms of completeness. It was shown in [5] that perfect matching is randomly (or non-uniformly) reducible to $\mathrm{Mod}_k\mathrm{L}$ for every k. From our results this implies a (random or non-uniform) reduction from matching to GI, which provides the first reduction between the two well studied problems. Moreover, as a consequence of derandomization results from [21, 3, 25], under the natural hypothesis that there is a set in DSPACE(n) with circuits of size $2^{\Omega(n)}$, our reduction implies a many-one AC^0 (deterministic) reduction from perfect matching to GI.

The graph automorphism problem GA, the decision of whether a given graph has a nontrivial automorphism, is known to be many-one reducible to GI and seems to be a slightly easier problem. We show in Section 5 that the hardness results for GI hold also for GA.

2. Preliminaries. We assume familiarity with basic notions of complexity theory such as can be found in the standard textbooks in the area. We will prove hardness results for several logarithmic space complexity classes: NL is the class of languages accepted by nondeterministic Turing machines using logarithmic space. The graph accessibility problem GAP (given a directed graph with two designated nodes s and t decide whether there is a path from s to t) is known to be complete for NL, even in the case of acyclic graphs with in-degree at most 2.

#L defined in [4] analogously to Valiant's class #P, is the class of functions $f : \Sigma^* \to \mathbb{N}$ that count the number of accepting paths of a nondeterministic Turing machine M on input x. The computation of a #L function on an input x can be reduced to the problem computing the number paths from node s to node t in a directed graph G_x. The complexity classes PL (probabilistic logarithmic space), $\mathrm{C}_=\mathrm{L}$ (exact threshold in logarithmic space), and $\mathrm{Mod}_k\mathrm{L}$ (modular counting in logarithmic space, $k \geq 2$) can be defined in terms of #L functions:

$$\mathrm{PL} = \{A : \exists p \in \text{ Poly}, f \in \#\mathrm{L},\ x \in A \Leftrightarrow f(x) \geq 2^{p(|x|)}\} \ [16,\ 35]$$

$$\mathrm{C_{=}L} = \{A : \exists p \in \text{ Poly}, f \in \#\mathrm{L},\ x \in A \Leftrightarrow f(x) = 2^{p(|x|)}\} \text{ [2]}$$

$$\mathrm{Mod}_k\mathrm{L} = \{A : \exists f \in \#\mathrm{L},\ x \in A \Leftrightarrow f(x) = 1 \bmod k\} \text{ [10]}$$

Mod_k circuits ($k \geq 2$), are circuits where the input variables (and the wires) can take values in $\mathbb{Z}_k$, and the gates compute addition in $\mathbb{Z}_k$. The evaluation problem for such circuits (given fixed values for the inputs, decide whether the output value is for example 1) is complete for $\mathrm{Mod}_k\mathrm{L}$ under AC^0 many-one reductions. This is because a directed acyclic graph with in-degree at most two, and two designated nodes, s, t, can be easily transformed into a mod_k circuit computing the residue of the number of paths from s to t in G, modulo k.

In some of the proofs we will make use of NC^1 circuits. These are families of logarithmic depth, polynomial size Boolean circuits of bounded fan-in over the basis $\{\wedge, \vee, \neg\}$. DET [13] is the class of problems NC^1 Turing reducible to the determinant, or in other words, the class of problems that can be solved by NC^1 circuits with additional oracle gates that can compute the determinant of integer matrices.

The known relationships among the considered classes are:

$$\mathrm{Mod}_k\mathrm{L} \subseteq \mathrm{DET},$$

$$\mathrm{NL} \subseteq \mathrm{C_{=}L} \subseteq \mathrm{PL} \subseteq \mathrm{DET}.$$

Looking at the known inclusions, the hardness of GI for DET implies hardness with respect to the other classes. We prove however the result for all the classes separately showing how the graphs produced by the reductions increase in complexity.

2.1. Reducibilities. We prove our hardness results for the DLOGTIME uniform AC^0 many-one reducibility (in short AC^0 reducibility). A set A is AC^0 reducible to another set B if there is family of circuits $\{C_n \mid n \in \mathbb{N}\}$ where each circuit C_n contains only AND, OR and NOT gates, has size $n^{O(1)}$ and depth $O(1)$ and for each x of length n, $x \in A \Leftrightarrow C_n(x) \in B$. Moreover the uniformity condition requires that there is a DLOGTIME Turing machine with direct access to its input defining the circuit in the sense that the machine can recognize the direct connection language of C_n [34, 8]. This language consists of the set of tuples $\langle t, a, b, y\rangle$ where a and b are numbers of nodes in C_n, t is the type of a, b is a child of a and y is a string of length n.

2.2. Graph isomorphism, automorphism and promise isomorphism. An automorphism in an undirected graph $G = (V, E)$ is a permutation φ of the nodes, that preserves adjacency. That is, for every $u, v \in V, (u, v) \in E \Leftrightarrow (\varphi(u), \varphi(v)) \in E$. An isomorphism between two graphs G, H is a bijection between their sets of vertices which preserves the edges. $G \simeq H$ denotes that G and H are isomorphic. GI is the problem

$$\mathrm{GI} \ = \{(G, H) \mid G \text{ and } H \text{ are isomorphic graphs}\}$$

and GA is defined as

$$\mathrm{GA} \ = \{G \mid\ G \text{ has an automorphism different from the identity}\}.$$

A graph G in $\overline{\mathrm{GA}}$ is called rigid. For technical reasons we will consider the set of graph pairs $((G,H),(I,J))$ with exactly one of the pairs consisting of isomorphic graphs:

$$\mathrm{PGI} = \{((G,H),(I,J))|\ G \simeq H \text{ if and only if } I \not\simeq J\}.$$

For a tuple in PGI we are given the promise of G being isomorphic to H or I being isomorphic to J and the question is to decide which one is the isomorphic pair.

Sometimes we will deal with graphs with colored vertices. A coloring with k colors is a function $f : V \to \{1,\dots k\}$. In an isomorphism between colored graphs, the colors have to be preserved. The isomorphism problem for colored graphs can be easily reduced (by AC^0 reductions) to graph isomorphism without colors (see e.g. [26]).

In some cases we will consider the following restricted automorphism problem: Given a graph $G = (V,E)$ and two lists of nodes $(x_1,\dots,x_k),(y_1,\dots,y_k)$, is there an automorphism in G mapping x_i to y_i for $1 \le i \le k$? This problem is also easily reducible to GI. In order to check whether there is an automorphism with the desired properties one can make two copies of G, G' and G''. In G' each of the nodes x_i has color i and in G'' node y_i receives this color. All the other nodes are colored with a new color 0, for example. G' and G'' are isomorphic if and only if G has an automorphism with the mentioned properties.

3. Hardness for the modular counting classes. We show now that GI is hard for for all the logarithmic space modular counting classes $\mathrm{Mod}_k\mathrm{L}$ ($k \ge 2$). The idea for this proof is to simulate a modular gate with a graph gadget and then combine the gadgets for the different gates into a graph, whose automorphisms simulate the behavior of the modular circuit.

The gadgets are defined by the following graphs (shown in Figure 3.1 for the case $k = 2$):

Definition 3.1. *Let $k \ge 2$ and denote by $\oplus$ the addition in $\mathbb{Z}_k$. We define the undirected graph $G^k = (V,E)$, given by the set of $k^2 + 3k$ nodes*

$$\begin{aligned} V = \{&x_a, y_a, z_a \mid a \in \{0,\dots,k-1\} \cup \\ &\{u_{a,b} \mid a,b \in \{0,\dots,k-1,\}\} \end{aligned}$$

and edges

$$\begin{aligned} E = \{&(x_a, u_{a,b}) \mid a,b \in \{0,\dots,k-1\}\} \cup \\ \{&(y_b, u_{a,b}) \mid a,b \in \{0,\dots,k-1\}\} \cup \\ \{&(u_{a,b}, z_{a\oplus b}) \mid a,b \in \{0,\dots,k-1\}\}. \end{aligned}$$

The graph gadget for a modular gate has nodes encoding the inputs and outputs of the gate. Any automorphism in the graph mapping the input nodes in a certain way, must map the output nodes according to the value of the modular gate being simulated.

Lemma 3.2. *Fix $k \ge 2$, for any $a,b \in \{0,\dots,k-1\}$,*

1) *there is a unique automorphism φ in G^k mapping x_i to $x_{a\oplus i}$ and y_i to $y_{b\oplus i}$ for $i = 0\dots k-1$, and*
2) *this automorphism maps z_i to $z_{a\oplus b\oplus i}$.*

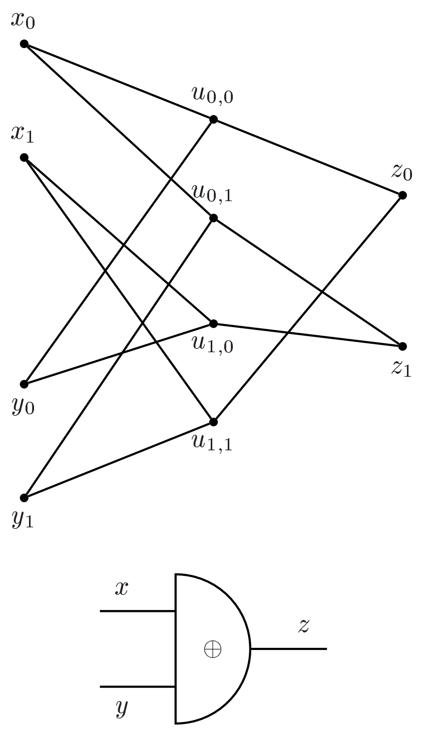

FIG. 3.1. *The graph G^2 simulating a parity gate.*

Proof. Let $a, b \in \{0, \ldots, k-1\}$, and denote by $\oplus$ the addition in $\mathbb{Z}_k$. We consider the following function $\varphi : V \to V$ defined as

$$\begin{aligned}
\varphi(x_i) &= x_{a\oplus i} \text{ for } i \in 0, \ldots, k-1,\\
\varphi(y_i) &= y_{b\oplus i} \text{ for } i \in 0, \ldots, k-1,\\
\varphi(u_{i,j}) &= u_{a\oplus i, b\oplus j} \text{ for } i, j \in 0, \ldots, k-1,\\
\varphi(z_i) &= z_{a\oplus b\oplus i} \text{ for } i \in 0, \ldots, k-1.
\end{aligned}$$

We prove first that φ is an automorphism. For this we have to show that for every pair of nodes v, w, $(v, w) \in E$ if and only if $(\varphi(v), \varphi(w)) \in E$. The nodes in graph G^k can be partitioned in three layers, the x and y nodes, (input layer) the u nodes and the z nodes (output layer). Edges exist only between nodes from first and second layers, or between nodes from second and third layers. We consider first an edge between the first two layers. Let $v = x_i$ and $w = u_{l,m}$ with $i, l, m \in \{0, \ldots, k-1\}$. Then $\varphi(v) = x_{a\oplus i}$ and $\varphi(w) = u_{a\oplus l, b\oplus m}$. By the definition of G^k,

$$\begin{aligned}
(x_i, u_{l,m}) \in E &\Leftrightarrow i = l\\
&\Leftrightarrow a \oplus i = a \oplus l\\
&\Leftrightarrow (x_{a\oplus i}, u_{a\oplus l, b\oplus m}) \in E\\
&\Leftrightarrow (\varphi(x_i), \varphi(u_{l,m})) \in E.
\end{aligned}$$

In the case $v = y_j$ the proof is analogous. For an edge (v, w) between the second and third layers, let $(v, w) = (u_{i,j}, z_l)$ with $i, j, l \in \{0, \ldots, k-1\}$. Then $\varphi(v) = u_{a\oplus i, b\oplus j}$ and $\varphi(w) = z_{a\oplus b\oplus l}$. By the definition of G^k,

$$\begin{aligned}(u_{i,j}, z_l) \in E &\Leftrightarrow i \oplus j = l\\ &\Leftrightarrow a \oplus b \oplus i \oplus j = a \oplus b \oplus l\\ &\Leftrightarrow a \oplus i \oplus b \oplus j = a \oplus b \oplus l\\ &\Leftrightarrow (u_{a\oplus i, b\oplus j}, z_{a\oplus b\oplus l}) \in E\\ &\Leftrightarrow (\varphi(u_{i,j}), \varphi(z_l)) \in E.\end{aligned}$$

In any automorphism ϕ with the restrictions $\phi(x_i) = x_{a\oplus i}$ and $\phi(y_i) = y_{b\oplus i}$, the node $\phi(u_{i,j})$ must have edges to $x_{a\oplus i}$ and $y_{b\oplus j}$ but the only node with such connections is $u_{a\oplus i, b\oplus j} = \varphi(u_{i,j})$.

Analogously $\phi(z_i)$ must be connected to $\phi(u_{0,i}) = u_{a, b\oplus i}$ and this implies $\phi(z_i) = z_{a\oplus b\oplus i} = \varphi(z_i)$. This means that φ is the unique automorphism in G^k mapping x_i to $x_{a\oplus i}$ and y_i to $y_{b\oplus i}$ □

We observe that the gadget in Lemma 3.2 for the case $k = 2$ has been already used for a different application in [12]. It is not hard to see that a gadget like the one defined in Lemma 3.2 for $(\mathbb{Z}_k, \oplus)$ can be constructed for any finite Abelian group $G = (A, \circ)$. We mean by this that for any such group a graph whose automorphism group simulates the group operation $\circ$ in the sense of the Lemma can be defined.

THEOREM 3.3. *For any $k \geq 2$, GI is hard for $Mod_k L$ under AC^0 many-one reductions.*

Proof. Let $k \geq 2$. We reduce the mod$_k$ circuit value problem to GI. We transform an instance C of the circuit value problem for mod$_k$ circuits into a graph G_C by constructing for every modular gate g_j of C a subgraph like the one described in Lemma 3.2. Moreover, we color the x, y, u and z nodes of the j-th gadget respectively with one of the colors $(x, j), (y, j), (u, j)$ and (z, j). Connections between gates are translated in the following way: If the output z of a gate in the circuit is connected to one of the inputs x of another gate, the reduction puts k additional edges connecting (for $i \in \{0, \ldots, k-1\}$) node z_i from the first gate to node x_i from the second gate. For an input variable v^j, k nodes $v_0^j, \ldots, v_{k-1}^j$ are considered in the reduction. The coloring implies that in any automorphism the nodes corresponding to a gate are mapped to nodes from the same gate. Suppose the input variables of the circuit, $v^1, \ldots, v^n$ take values $a_1, \ldots, a_n$. It follows from Lemma 3.2, by induction on the circuit depth, that the output gate z takes value $b \in \{0, \ldots, k-1\}$ if and only if there is an automorphism in G_C mapping v_i^j to $v_{i\oplus a_i}^j$ for all $i = 0, \ldots, k-1$ and $j = 1, \ldots, n$, and mapping z_i to $z_{i\oplus b}$.

All the steps in the reduction can be done locally by an AC^0 circuit. The question of whether the output of the circuit equals $b \in \{0, \ldots, k-1\}$ can be easily reduced to whether a pair of graphs G_b, G'_b are isomorphic, as explained in the preliminaries. In fact this question can be reduced to two graphs pairs $((G, H), (I, J)) \in$ PGI with G being isomorphic to H if the value of the circuit is b and I being isomorphic to J otherwise. For this it suffices to define G as G_b, H as G'_b, and I and J as the standard OR-function for GI of $\bigcup_{i\neq b}(G_i, G'_i)$. □

Observe that the graphs obtained in the reduction, have at most k^2 nodes with the same color (the nodes $u_{i,j}$ in any of the gate gadgets). The maximum degree can be reduced to 3. In the above description this does not necessarily hold because of the connection between gates. However, the reduction can be easily modified to achieve degree 3 by adding some extra nodes and arranging the fan-out connections of the gates in a tree-like fashion.

4. Hardness for other complexity classes. In this section we show the hardness of GI for nondeterministic logarithmic space, for $C_=L$, for probabilistic logarithmic space and for the class DET of problems NC^1 reducible to the determinant. The proofs follow by the modular results, using the Chinese Remainder Theorem (CRT).

A Chinese remainder representation base is a set $m_1, \ldots, m_n$ of pairwise coprime integers. Let $M = \prod_{i=1}^{n} m_n$. By the CRT, every integer $0 \le x < M$ is uniquely represented by its Chinese remainder representation $(x_1, \ldots, x_n)$, where $0 \le x_i < m_i$ and $x_i = x \bmod m_i$. We will consider the base B_n formed by the first n prime numbers.

THEOREM 4.1. *GI is hard for NL under AC^0 many-one reductions.*

Proof. The graph accessibility problem for directed acyclic graphs with fan-in at most 2 is complete for the class NL. We reduce the complement of this set (non-reachability) to GI. The result follows by the closure of NL under complementation [20, 37]. Let $G = (V, E)$ be such a graph, with $|V| = n$ and with two designated nodes s and t. Let P the number of paths from s to t in G. Clearly $P \le 2^n$ and $P = 0$ if and only if for for every i between 1 and n it holds $P \bmod i = 0$.

In the reduction, on input G, an AC^0 circuit, for each i between 1 and n, transforms G into a circuit C_i with addition modulo i gates. The circuits have the property that their outputs coincide with $P \bmod i$ (see the preliminaries). In a second step the reduction transforms the sequence of C_i circuits into a sequence of graphs G_{C_i} (as in the proof of Theorem 3.3) in which there is an automorphism mapping the input nodes according to the inputs of C_i and mapping z_0^i (the node corresponding to the output gate of G_{C_i}) to z_j^i if and only if $P = j \bmod i$. The number of paths from s to t in G is then 0 if and only if for all $i \le n$ there is an automorphism in G_{C_i} mapping the input nodes G_{C_i} according to the inputs of C_i and mapping z_0^i to itself. This can be easily reduced to GI as explained in the preliminaries. □

Observe that in the graphs obtained in this reduction, the size of the classes of the nodes with the same color are not bounded by a constant as before, but by n^2.

In fact, we can reduce any logarithmic space counting function to GI. We understand by this that for any function $f \in \#L$ the set

$$A_f = \{\langle x, 0^i \rangle \mid \text{ the } i\text{-th bit of } f(x) \text{ is } 1\}$$

is many-one reducible to GI.

For proving this reduction, we need two known results. On the one hand we need the surprising fact that division can be computed by uniform TC^0 circuits[1]. [17, 18]. More precisely we need the following part of the mentioned result:

THEOREM 4.2. *[17, 18] There is a DLOGTIME uniform family of TC^0 circuits that on input the Chinese remainder representation $(x_1, \ldots, x_n)$ in base B_n of a number x, outputs the binary representation of x.*

We also need the fact that the result of an NC^1 circuit with fixed values in the input nodes can be encoded as a graph isomorphism question. This follows from an adaptation of the proof of Theorem 3.1 in [22] stating that GI is hard for NC^1 under DLOGTIME uniform AC^0 many-one reductions. For completeness we give a sketch of the proof. The reader is referred to [22] for the details. For technical reasons needed

[1]In fact for our purposes suffices the weaker result stating that division is in alternating time $O(\log n)$ which proved in [14]

in the proof of Theorem 4.9, we encode the values of the circuit as tuples of graphs $((G, H), (I, J))$ in PGI, with $G \simeq H$ and $I \not\simeq J$ for the encoding of a 1 and with $G \not\simeq H$ and $I \simeq J$ for the encoding of a 0. Recall that PGI was the set of graph tuples $((G, H), (I, J))$ with exactly one of the graphs pairs being isomorphic.

THEOREM 4.3. *Given a uniform family of circuits C_n with logarithmic depth and polynomial size and given n tuples of graphs $((G_i, H_i), (I_i, J_i)) \in PGI$, there is an AC^0 reduction constructing a tuple $((G, H), (I, J)) \in PGI$ with the property that $G \simeq H$ if and only if C_n outputs 1 and $I \simeq J$ if and only if C_n outputs 0, where the i-th input to C_n consists of the bit of the boolean value of the statement $G_i \simeq H_i$.*

Proof. (Sketch) An NC^1 circuit can be simulated by a balanced DLOGTIME uniform family of circuits with fan out 1, logarithmic depth, polynomial size and alternating layers of ANDs and ORs [8]. We show how to transform these expressions to graph tuples. The idea is to construct graph gadgets to simulate the AND and OR connectives in the circuit. Given two tuples $((G_1, H_1), (I_1, J_1))$ and $((G_2, H_2), (I_2, J_2))$ in PGI consider the graphs $G_\wedge, H_\wedge, I_\wedge$ and $J_\wedge$ in Figure 4.1, where an edge between two graphs represents that each node of the first graph is connected to each node of the second graph. We can also suppose that the nodes of G_1 and H_1 are colored with the same special color to handle the cases when $G_1 \simeq H_2$. This is represented by a double ring arround the graphs in the figure. The constructed graphs have the property that $G_\wedge \simeq H_\wedge$ if and only if $G_1 \simeq H_1$ and $G_2 \simeq H_2$. Also $I_\wedge \simeq J_\wedge$ if and only if $G_1 \not\simeq H_1$ or $G_2 \not\simeq H_2$ (in this case $I_1 \simeq J_1$ or $I_2 \simeq J_2$).

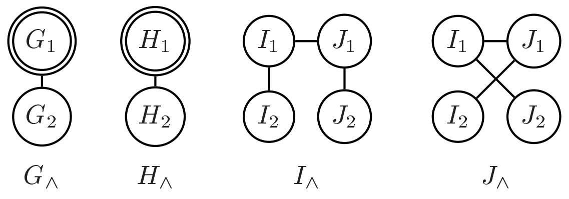

FIG. 4.1. *Tuple $(G_\wedge, H_\wedge, I_\wedge, J_\wedge)$ simulating AND.*

Similarly, the graphs $G_\vee, H_\vee, I_\vee$ and $J_\vee$ from Figure 4.2 have the property that $G_\vee \simeq H_\vee$ if and only if $G_1 \simeq H_1$ or $G_2 \simeq H_2$ and $I_\vee \simeq J_\vee$ if and only if $G_1 \not\simeq H_1$ and $G_2 \not\simeq H_2$. Observe that $((G_\wedge, H_\wedge), (I_\wedge, J_\wedge))$ and $((G_\vee, H_\vee), (I_\vee, J_\vee))$ belong to PGI.

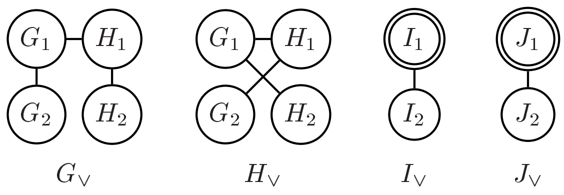

FIG. 4.2. *Tuple $((G_\vee, H_\vee), (I_\vee, J_\vee)$ simulating OR.*

The constructions doubles the number of nodes of the initial tuples. Notice also that it is easy to simulate a NOT by transforming $((G, H), (I, J))$ to $((I, J), (G, H))$.

A 1 in the circuit is represented by a tuple $((G, H), (I, J))$ with $G \simeq H$ and a 0 by a tuple with $I \simeq J$. Starting from the input nodes the reduction transforms the nodes of the circuit into graph tuples encoding the values of the circuit gates . Since the circuit has logarithmic depth the tuples corresponding to the output gate have a polynomial number of nodes. □

We can now show the hardness of GI with respect to #L.

THEOREM 4.4. *Every #L function*[2] *is AC^0 many-one reducible to GI.*

Proof. Let $f \in$ #L. For some polynomial q, it is possible to construct in AC^0 for $x \in \Sigma^*$ a graph G_x with at most $q(|x|)$ nodes so that $f(x)$ is the number of $s - t$ paths in G_x. Let i be the bit of $f(x)$ we want to reduce to GI, and let $m = q(|x|)$. By Theorem 4.2, in order to compute $f(x)$, it suffices to compute its Chinese remainder representation $(f(x)_1, \dots, f(x)_m)$ in B_m. Once this is done, $f(x)$ can be computed by an NC^1 circuit.

The Chinese remainder representation can be obtained by computing prime number p_i, for every $1 \le i \le m$, (this can be done by an NC^1 circuit) and reducing G_x to a circuit with addition gates in $\mathbb{Z}_{p_i}$, as in the proof of Theorem 4.1. The circuits are transformed into p_i graph tuples $((G_j, H_j), (I_j, J_j))$ with the property that in the j-th tuple the first two graphs are isomorphic if and only if $f(x) = j - 1 \bmod p_i$. These form a list of $\sum_{i=1}^{m} p_i$ graph tuples, and can be considered as an encoding of the CRR of $f(x)$ $(f(x)_1, \dots, f(x)_m)$ of the form $(w_1, \dots, w_m)$ where each $w_i \in \{0,1\}^{p_i}$ is formed by 0's with a 1 in position $f(x)_i + 1$. The 0's and 1's in the w_i's are encoded by tuples in PGI.

By Theorem 4.2 it is possible to construct in DLOGTIME a TC^0 (and therefore also an NC^1) circuit that having as inputs the CRR of $f(x)$, outputs the i-th bit of $f(x)$. We can consider the list of graph tuples as the inputs of this circuit.

So far we have shown that there is a uniform AC^0 reduction that on input x computes an NC^1 circuit that outputs the i-th bit of $f(x)$ and has its input values encoded as graph tuples in PGI. As done in the proof of Theorem 4.3, an AC^0 reduction can also transform the whole circuit into a single tuple of graphs $((G, H), (I, J))$. G is isomorphic to H or I is isomorphic to J depending on the output of the NC^1 circuit, which coincides with the i-th bit of $f(x)$. □

Basically the same proof as the one for the hardness for NL holds for proving hardness for the class $\mathrm{C_{=}L}$. Here instead of checking that the number of paths from s to t is 0, we have to check that this number coincides with some exact threshold $f(G) \le 2^n$. For this the reduction machine has to compute for each small prime p_i the residue $r_i = f(G) \bmod p_i$ (this can be done in NC^1 [32] and in fact in TC^0 by the mentioned result on division in [18]), and then check whether there is an automorphism that for all i maps z_0^i to $z_{r_i}^i$.

COROLLARY 4.5. *GI is hard for $C_{=}L$ under AC^0 many-one reductions.*

As mentioned in the preliminaries, for a set $L \in$ PL, there is a function $f \in$ #L and a polynomial p such that for any input x, $x \in L$ if and only if $f(x) \ge 2^{p(|x|)}$. The next result follows then directly from Theorem 4.4 since an input x belongs to L if an only if at least one of the bits corresponding to positions $\ge p(|x|)$ (starting from the right) in the binary representation of $f(x)$ is a 1.

COROLLARY 4.6. *GI is hard for the class PL under AC^0 many-one reductions.*

The class DET of problems NC^1 Turing reducible to the determinant coincides with $\mathrm{NC}^1(\#\mathrm{L})$ (see e.g. [2]). Combining Theorems 4.3 and 4.4, we can prove the hardness of GI for DET, which is the strongest known hardness result for GI.

The proof of this result is based on a simulation of the NC^1 circuit as done in Theorem 4.3 replacing each of the oracle queries to f by a small circuit as in the proof

[2]In fact this result also holds for the more powerful class of GapL functions defined in [2].

of Theorem 4.4. The main problem here is that while in Theorem 4.4 the input for the #L function to be computed is a binary string x, in the simulation of the NC^1 circuit the input to the oracle calls are not given as a sequence of bits but as a sequence of graph tuples encoding these bits. To deal with this problem we need the following lemma stating that Theorem 4.4 is also true when the input is encoded as a sequence of tuples:

LEMMA 4.7. *For each function $f \in \#L$ there is a DLOGTIME uniform family $\{C_n\}$ of AC^0 circuits such that on input a sequence of graph tuples $((G_i, H_i), (I_i, J_i))$ in PGI, $1 \leq i \leq n$, of size polynomial in n encoding a binary string $x \in \Sigma^n$, C_n constructs a sequence of tuples $((G'_i, H'_i), (I'_i, J'_i))$ in PGI, $1 \leq i \leq q(n)$, encoding the bits of $f(x)$.*

Proof. Let $f \in \#\mathrm{L}$ and $n, k, m \in \mathbb{N}$. From the description of the nondeterministic logarithmic space machine M computing f the reduction constructs first in AC^0 a graph G^n_f of polynomial size in n related to the configuration graph of M. We can consider that M has a read-only tape for the input and a work tape of logarithmic size. The set of nodes of G^n_f consists of the set of tuples (s, c, p_1, p_2, b) where s is a state of M, c is a possible content of the work tape, p_1 and p_2 are the positions of the tape heads on the input and work tape respectively, and b is one bit that will be used to encode the content at position p_1 on the input tape. For a concrete input some of these descriptions are not consistent with the input information since b might not be the correct bit at position p_1. Nevertheless we consider the set of all such possible descriptions at this point. This set hat polynomial size in n. The set of edges in G^n_f is given by the transition function of M. If the machine can reach from a description $d = (s, c, p_1, p_2, b)$ the configuration (s', c', p'_1, p'_2) in one step, then there is a directed edge in G^n_f from d to $(s', c', p'_1, p'_2, 0)$ and another one from d to $(s', c', p'_1, p'_2, 1)$.

Let x be the input for f encoded by a sequence of graph triplets in PGI. In order to compute whether $f(x) \bmod k$ is congruent with m we will consider that the nodes of G^n_f are addition gates in $\mathbb{Z}_k$ in a polynomial size circuit C. If all the nodes of C would correspond to descriptions consistent with the input, then the output of this circuit would be $f(x) \bmod k$. However, half of the gates in C correspond to inconsistent descriptions and they corrupt the final sum. To avoid this problem we use a method that guarantees that the wires coming out of the inconsistent gates always have value 0 and therefore do not contribute to the final sum. This will be done with a new graph gadget. Using first the graph gadgets in Section 3, circuit C can be transformed into a graph G_C where each of the mod k gates corresponding to a machine description $d = (s, c, p, p', b)$ is transformed into a subgraph with input nodes $x_0, \ldots x_{k-1}$ and $y_0, \ldots y_{k-1}$ and output nodes $z_0, \ldots z_{k-1}$ in such a way that if there is an automorphism mapping x_l to $x_{l \oplus i}$ and y_l to $y_{l \oplus j}$ in this subgraph, then the automorphism maps z_l to $z_{l \oplus i \oplus j}$ for $i, j, l \in \{0, \ldots k-1\}$ (Lemma 3.2). The output nodes z_l are then connected with an edge to the input nodes of other gates, nodes $w_0 \ldots w_{k-1}$ (the nodes of z are connected to as many nodes as the fan out of the corresponding gate, for simplicity we consider it is just one). Let us suppose that the bit b in description d is 1 (the 0 case is completely analogous) and let $((G_p, H_p), (I_p, J_p))$ be the input tuple in PGI encoding the correct value for the position p in the input x. The gate corresponding to d is a consistent gate if and only if b equals the Boolean value of $G_p \simeq H_p$. To force that the inconsistent gates always propagate a 0 (an automorphism mapping z_0 to itself) the reduction includes between the z and w nodes the following gadget Gad_k that can be seen in Figure 4.3 for the case $k = 2$. Connections between a node v and a graph in the figure and in the following description of Gad_k mean that there is an edge between v and each of the nodes in the graph.

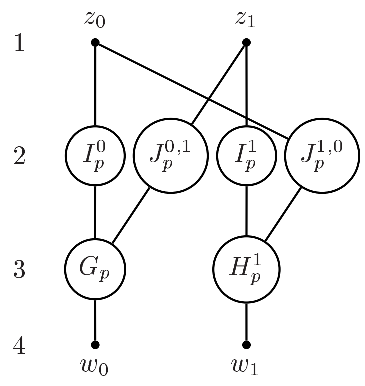

FIG. 4.3. *The graph Gad_2.*

Subgraph Gad_k can be represented in four levels. Levels 1 and 4 contain the nodes z_i, and w_i, respectively, for $i \in \{0, \dots k-1\}$. Level 2 contains for each i a copy I_p^i of I_p and $k-1$ copies of J_p, $J_p^{i,j}$, $j \in \{0, \dots k-1\}$, $j \neq i$. Level 3 contains a copy of G_p and for each i in $\{1, \dots, k-1\}$ a copy H_p^i of H_p. The edges are defined as follows:

– Each node z_i is connected to the graphs I_p^i and to the $k-1$ graphs $J_p^{l,i}$ for $l \neq i$ in the second level.
– Graph G_p in the third level is connected to I_p^0 and to each of the graphs $J_p^{0,j}$ $j \neq 0$, all of them in the second level.
– The graphs H_p^i, $i \neq 0$ in the third level are connected to I_p^i and to $J_p^{i,j}$, $j \neq i$.
– Finally in the fourth level, node w_0 is connected to G_p and each w_i for $i \neq 0$ is connected to H_p^i.

Gad_k has very nice properties, as can be seen in the next lemma.

LEMMA 4.8. *Subgraph Gad_k has the following properties:*

1. *If the gate is consistent with the input, that is, if $G_p \simeq H_p$ then for any $c \in \{0, \dots, k-1\}$ there is an automorphism in Gad_k mapping z_i to $z_{i \oplus c}$ for each i. Such automorphism maps also w_i to $w_{i \oplus c}$.*
2. *If the gate is inconsistent with the input, that is, if $I_p \simeq J_p$ then for any $c \in \{0, \dots, k-1\}$ there is an automorphism in Gad_k mapping z_i to $z_{i \oplus c}$ for each i. Such automorphism maps also w_i to w_i.*

Proof. In order to see 1), observe that if the automorphism maps z_i to $z_{i \oplus c}$, then the graph I_p^i connected to z_i has to be mapped to one of the graphs connected to $z_{i \oplus c}$, $J_p^{j, i \oplus c}$ or $I_p^{i \oplus c}$. But I_p cannot be mapped to J_p since these graphs are not isomorphic. This implies that in any automorphism all the graphs I_p in the second level have to be mapped to graphs of type I_p. In particular I_p^i has to be mapped to $I_p^{i \oplus c}$. This means that the graph G_p at the third level has to be mapped to the H_p graph over w_c (this can happen since $G_p \simeq H_p$) and this implies that for all i, w_i has to be mapped to $w_{i \oplus c}$. An automorphism satisfying all these conditions can be defined by mapping for all i, j with $i \neq j$ $J_p^{i,j}$ to $J_p^{i \oplus c, j \oplus c}$ at the second level. Observe that in case G_p, H_p, I_p and J_p are rigid graphs, then the described automorphism is the only one mapping z_i to $z_{i \oplus c}$ for each i.

For the proof of 2) observe that in case the gate is inconsistent with the input, then the graph G_p at the third level has to be mapped to itself and therefore the

w nodes also have to be mapped to themselves. We have to prove that there is an automorphism with these properties mapping z_i to $z_{i\oplus c}$ for each i. This is clear for $c = 0$. For $c \neq 0$ this automorphism maps in the second level graph I_p^i to $J_p^{i,i\oplus c}$ (this is possible since $I_p \simeq J_p$) and maps $G_p^{i,j}$ to $G_p^{i,j\oplus c}$, if $i \neq j \oplus c$ or to I_p^i in case $i = j \oplus c$. The automorphism fixes the third and fourth levels, and again, in case G, H, I and J are rigid graphs, it is the only one mapping z_i to $z_{i\oplus c}$ for each i. □

We continue with the proof of Lemma 4.7. Let G'_C be the graph corresponding to circuit C with the new gadgets on the edges coming out of the gates in C. The above lemma guarantees that inconsistent gates always produce value 0 and therefore the circuit produces the correct value for $f(x)$ mod k. Let $z_0, \dots z_{k-1}$ be the output nodes in G'_C corresponding to the output gate of the circuit. By the results in Section 3, there is an automorphism in G'_C mapping for each i z_i to $z_{i\oplus m}$ if and only if $f(x) \equiv m$ mod k. This property can be encoded by the reduction using standard methods into a graph tuple $((G, H), (I, J))$ in PGI satisfying that $G \simeq H$ if $f(x) \equiv m$ mod k and $I \simeq J$ otherwise. Observe that if the graphs in the tuples have size at most s then the size of the output graphs is at most $p(n)s$ for a polynomial p depending on the machine M. The rest of the proof is exactly as in Theorem 4.4. □

We can now prove the hardness of GI for DET. This result answers positively a question posed by Allender in [1]. Recall that DET can be characterized as $\mathrm{NC}^1(\#\mathrm{L})$ the class of problems computed by an AC^0 uniform family of polynomial size and logarithmic depth circuits with oracle gates to a function f in #L. By convention, an oracle gate querying a string x contributes $\log(|x| + |f(x)|)$ to the total circuit depth.

THEOREM 4.9. *GI is hard for the class DET under AC^0 many-one reductions.*

Proof. Let L be a set in $\mathrm{NC}^1(\#\mathrm{L})$ and let $\{C_n\}$ be the family of NC^1 circuits computing L with functional oracle queries to a function f in #L.

We want to compute $C_n(x)$ for a string x of length n. The reduction can first transform each oracle gate g into a circuit D_g as done in Theorem 4.4. Observe that the structure of the circuit computing gate g does not depend on the input bits of g, but just on the number of such bits. D_g computes the query using modular gates as well as AND and OR gates. D_g has polynomial size (in the size of its input) and its depth is not necessarily logarithmic, but the number of levels with AND or OR gates in this circuit is logarithmic in the input size of g. If we only count the depth of the AND and OR gates (the maximum number of such gates in a path from an input to the output gate), C_n with the expanded oracles gates still has logarithmic depth in n since we are dealing with an NC^1 reduction.

Each gate in the circuit C_n with expanded oracle queries can be transformed by the AC^0 reduction into a tuple of four graphs $((G, H), (I, J))$ encoding the value of the gate as explained before. Using Theorems 4.3 and 4.9 the reduction can construct these tuples for all the levels of the circuit. The graph tuple corresponding to the output gate encodes the result of the circuit computation.

It is only left to show that the size of the graph tuples corresponding to the circuit gates remain of polynomial size in n. The gadgets corresponding to the AND and OR gates increase the size of the graph tuples at most by a factor of 2 in each level, and the number of circuit levels with AND or OR gates is logarithmic in n. The gadgets attached to the modular computations in the query gates increase the size of the tuples by a factor of $p(m)$ where m is the size of the query and p is a polynomial. Because C_n computes an NC^1 reduction, in a circuit path with oracle queries with sizes $m_1, \dots, m_l$, it must hold that the sum of the logarithms of all the query sizes is at most $c\log(n)$ for some constant c. From this follows that the product of the increasing factors $p(m_i)$ corresponding to all the oracle queries in the path is bounded by a polynomial in n. These facts imply that the the size of the graph tuples corresponding to every gate in C_n is polynomial in n. ☐

4.1. Matching is reducible to GI. We mention an interesting connection between the perfect matching problem and GI. The perfect matching problem consists in deciding whether a given undirected graph has a perfect matching, that is, a set of edges that contain all the vertices, and such that no two of these edges share a vertex. This problem has been intensively studied, but like GI, it has resisted all classification attempts in terms of completeness in a class. The problem has polynomial time algorithms, and it is known to be in random NC [24, 33]. In [5] it has been proved that for any $k \geq 2$, the perfect matching problem is randomly reducible to a set in Mod_kL. Together with Theorem 3.3 this implies:

COROLLARY 4.10. *Matching is reducible to GI under randomized reductions.*

Since the reduction works correctly with probability exponentially close to 1, for each input size n there is a sequence of random choices that can be taken as correct advice in the reduction of all instances of size n. This implies a non-uniform reduction from Matching to GI. Moreover as noted in [3], under a natural hardness hypothesis, the reduction from Matching to Mod_kL can be derandomized using techniques from [21, 25]. This yields:

COROLLARY 4.11. *If there is a set A in DSPACE(n) and $\delta > 0$ with the property that, for all large n, no circuits of size less than $2^{\delta n}$ accepts exactly the strings of length n in A, then perfect matching is included in $Mod_k L$ for any $k \geq 2$ and thus the problem is reducible to GI under AC^0 many-one reductions.*

5. Hardness results for graph automorphism. The graph automorphism problem (GA) deciding whether a given graph has a nontrivial automorphism is many-one reducible to GI and it seems to be a slightly easier problem. In this section we show that the proven hardness results for GI hold also for GA. We show first that the hardness for the modular classes can be easily translated to GA.

THEOREM 5.1. *For any $k \geq 2$, GA is hard for $Mod_k L$ under AC^0 many-one reductions.*

Proof. In Theorem 3.3 we transformed a circuit with addition gates in $\mathbb{Z}_k$ and values for the input gates, into a graph G having a unique automorphism with certain restrictions (some nodes encoding the input and output values of the circuits had to be mapped in a certain way) if and only if the output value of the circuit is 1. The question of whether G has an automorphism with the desired properties can in turn be transformed into a GI problem by making two copies of G, G_1 and G_2. These graphs have to include some coloring in the nodes representing the input and output values of the circuit in order to encode the restrictions in the automorphism. Observe that there is at most one isomorphism between G_1 and G_2. From this follows that there is a nontrivial automorphism in $G_1 \cup G_2$ if and only if the output of the original circuit is 1. ☐

Based on this Theorem the proof of the result 4.1 can be modified to show hardness of GA for NL.

COROLLARY 5.2. *GA is hard for NL under AC^0 many-one reductions.*

The additional ingredient that is needed to prove the stronger hardness results, is the fact that an NC^1 computation can be encoded as a GA question, that is, a version of Theorem4.3 for GA. A direct translation of this result does not work since GA is not known to have AND-functions. An AND-function for GA is a function that is easy to compute and transforms pairs of graphs into single graphs in such a way that both of the original graphs have nontrivial automorphisms if and only if the final graph has such an automorphism. Dieter van Melkebeek has found a way to avoid this problem.

THEOREM 5.3. *(van Melkebeek) Given a uniform family of circuits C_n with logarithmic depth and polynomial size and given n tuples of rigid graphs $((G_i, H_i), (I_i, J_i)) \in$ PGI, there is an AC^0 reduction constructing a tuple of rigid graphs $((G, H), (I, J)) \in$ PGI with the property that $G \simeq H$ if and only if C_n outputs 1 and $I \simeq J$ if and only if C_n outputs 0, where the i-th input to C_n consists of the bit of the boolean value of the statement $G_i \simeq H_i$.*

Proof. The proof is like the one for Theorem 4.3 simulating the alternating layers of ANDs and ORs of an NC^1 circuit by graph gadgets for the tuples. The main difficulty is to preserve the rigidity of the tuple components.

In order to simulate the AND, given two tuples of rigid graphs $((G_1, H_1), (I_1, J_1))$ and $((G_2, H_2), (I_2, J_2))$ in PGI consider the graphs $G_\wedge, H_\wedge, I_\wedge$ and $J_\wedge$ in Figure 5.1. $G_\wedge$ and $H_\wedge$ are defined as the standard AND function for GI of the G and H graphs, while $I_\wedge$ and $J_\wedge$ are constructed as the OR of (I_1, J_1) and the AND of (G_1, H_1) and (I_2, J_2). Again by a double ring arround some of the graphs we represent the fact that these graphs are marked in some special way and can only be mapped to other graphs with the same marking.

These graphs have the property that $G_\wedge \simeq H_\wedge$ if and only if $G_1 \simeq H_1$ and $G_2 \simeq H_2$. Also $I_\wedge \simeq J_\wedge$ if and only if $I_1 \simeq J_1$ (and therefore $G_1 \not\simeq H_1$) or $I_2 \simeq J_2$ (in this case $G_2 \not\simeq H_2$ and either $G_1 \simeq H_1$ or $I_1 \simeq J_1$). Although the standard OR does not preserve rigidity in case both inputs are isomorphic, observe that in this construction if all the graphs in the input tuples are rigid then $G_\wedge, H_\wedge, I_\wedge$ and $J_\wedge$ are also rigid. We avoid the ambiguous situation by rewriting "p or q" as "p or (not p and q)" and expressing "not p" positively by switching to the complementary pair of the tuple.

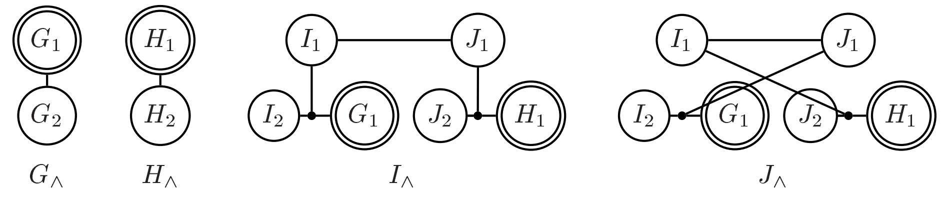

FIG. 5.1. *Tuple $(G_\wedge, H_\wedge, I_\wedge, J_\wedge)$ simulating AND.*

Similarly, the graphs $G_\vee, H_\vee, I_\vee$ and $J_\vee$ from Figure 5.2 have the property that $G_\vee \simeq H_\vee$ if and only if $G_1 \simeq H_1$ or $G_2 \simeq H_2$ and $I_\vee \simeq J_\vee$ if and only if $G_1 \not\simeq H_1$ and $G_2 \not\simeq H_2$. These gadgets simulate therefore an OR gate. Moreover, if the all the graphs G_i, H_i, I_i and J_i are rigid, for $i \in \{1, 2\}$, then the constructed graphs $G_\vee, H_\vee, I_\vee$ and $J_\vee$ are also rigid.

Observe that the size of the constructed gadgets is at most $3n$, n being the sum of all the nodes in the input tuples. Because of this fact, for a logarithmic depth circuit C with alternating layers of AND and OR fan-out 1 gates, a tuple of polynomial size rigid graphs $((G, H)(I, J))$ can be constructed such that C has value 1 if and only if $G \simeq H$. Since G and H are rigid, this is equivalent to $G \cup H \in$ GA. □

An immediate consequence of this result is that GA is hard for NC^1. Using this fact and Theorem 5.1, it is now possible to prove the hardness of GA for the

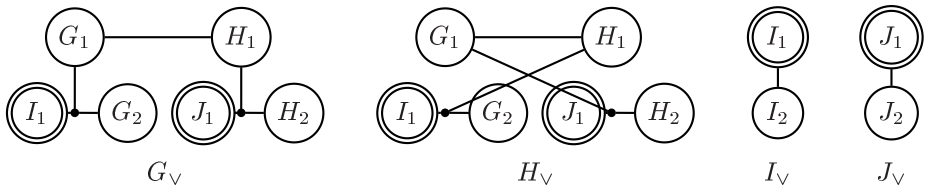

FIG. 5.2. *Tuple* $(G_\vee, H_\vee, I_\vee, J_\vee)$ *simulating OR.*

class DET. The proof of this result follows exactly the same lines as the one for Theorem 4.9 taking into consideration that the graph pairs produced in the reduction from Theorem 3.3 are rigid, and that the gadgets in the proof of Theorem 4.9 also preserve rigidity.

COROLLARY 5.4. *GA is hard for the class DET under AC^0 many-one reductions.*

One final observation is that from Theorem 5.1 it follows also that the perfect matching problem is randomly reducible to GA.

Acknowledgments: I would like to thank D. van Melkebeek for providing the proof for Theorem 5.3. He and an anonymous referee suggested also many improvements to the original version of the results. I had several interesting discussions with V. Arvind and J. Köbler that clarified the results of the paper. E. Allender pointed to me the connection to the determinant. I also would like to thank D. Therien for organizing the McGill Invitational Workshop where this research started.

REFERENCES

[1] E. Allender. The division breakthroughs. *Bulletin of the EATCS* Computational Complexity column June 2001.

[2] E. Allender and M. Ogihara, Relationships among PL, #L, and the determinant, *RAIRO Theoretical Information and Applications* 30:1–21, 1996.

[3] E. Allender, K. Rheinhardt and S. Zhou. Isolation, matching and counting: uniform and nonuniform upper bounds. In *Journal of Computer and System Sciences*, 59:164–181 1999.

[4] C. Àlvarez and B. Jenner. A very hard logspace counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[5] L. Babai, A. Gál and A. Widgerson. Superpolynomial lower bounds for monotone span programs. *Combinatorica* 19, 1999.

[6] L. Babai, D. Grigoryev, and D. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proc. 14th ACM Symp on Theory of Computing* pp. 310–324, 1982.

[7] L. Babai and E. Luks. Canonical labeling of graphs. In *Proc. 15th ACM Symp on Theory of Computing* pp. 171–183, 1983.

[8] 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.

[9] R. Boppana J. Hastad and S. Zachos. Does co-NP have short interactive proofs? In *Information Processing Letters* 25, pp. 27–32, 1987.

[10] G. Buntrock, C. Damm, U. Hertrampf and C. Meinel. Structure and importance of logspace-MOD-classes. In *Math. System Theory* 25: 223–237, 1992.

[11] 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, pp. 18-33, 1997.

[12] J. Cai, M. Fürer and N. Immerman, An optimal lower bound on the number of variables for graph identifications. *Combinatorica* 12(4): 389-410, 1992

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

[14] A. Chiu. Complexity of parallel arithmetic using the Chinese remainder representation. Master Thesis U. Wisconsin 1995.

[15] A. Chiu, G. Davida and B. Litow. Division in logspace uniform NC^1. *Theoretical Informatics and Applications* 35 (3) 259–275, 2001.

[16] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal con Computing* 6:675–695, 1977.

[17] W. Hesse. Division is in uniform TC^0. In *Proceedings ICALP 2001.* Lecture Notes in Computer Science 2076, Springer Verlag 104–114, 2001.

[18] W. Hesse, E. Allender, and D. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication, *Journal of Computer and Systems Sciences* 65:695–716, 2002.

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

[20] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[21] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR lemma. *Proc. 29th ACM Symp. on Theory of Computing* 220–229, 1997.

[22] B. Jenner, J. Köbler, P.McKenzie and J. Torán. *Completeness results for graph isomorphism. Journal of Computer and System Sciences*, 66: 549–566, 2003.

[23] B. Jenner, P.McKenzie and J. Torán. *A note on the hardness of tree isomorphism.* In Proc. 13th IEEE Computational Complexity Conference 101–106, 1998.

[24] R. Karp, E. Upfal and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica* 6:35–48, 1986.

[25] A. Klivans and D. van Melkebeek. Graph isomorphism has subexponential size proofs unless the polynomial time hierarchy collapses. *SIAM Journal on Computing* 31: 1501–1526, 2002.

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

[27] J. Köbler and J. Torán. The complexity of Graph Isomorphism for colored graphs with color classes of size 2 and 3. In *Proc. 19th STACS Conference*, Springer Verlag LNCS 2285 121–132, 2002.

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

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

[30] E. Luks. Parallel algorithms for permutation groups and graph isomorphism. *Proc. 27th IEEE Symp. on Foundations of Computer Science*, 292–302, 1986.

[31] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8:131–132, 1979.

[32] P. McKenzie and S. Cook. The parallel complexity of Abelian permutation group problems. *SIAM Journal on Computing* 19:880–909, 1987.

[33] K. Mulmuley, U. Vazirani and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica* 7:105–113, 1987.

[34] W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[35] W. Ruzzo, J. Simon and M. Tompa. Space bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28:216–230, 1984.

[36] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, 1988.

[37] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica* 26:279–284, 1988.