# GRAPH ISOMORPHISM IS LOW FOR PP

JOHANNES KÖBLER, UWE SCHÖNING
AND JACOBO TORÁN

**Abstract.** We show that the graph isomorphism problem is low for PP and for $C_=P$, i.e., it does not provide a PP or $C_=P$ computation with any additional power when used as an oracle. Furthermore, we show that graph isomorphism belongs to the class LWPP (see Fenner, Fortnow, Kurtz [12]). A similar result holds for the (apparently more difficult) problem Group Factorization. The problem of determining whether a given graph has a nontrivial automorphism, Graph Automorphism, is shown to be in SPP, and is therefore low for PP, $C_=P$, and $\mathrm{Mod}_k\mathrm{P}$, $k \geq 2$.
**Key words.** graph isomorphism; complexity classes; lowness; counting properties.
**Subject classifications.** 68Q15, 68Q25, 05C60, 68R10.

## 1. Introduction

The problem of finding an efficient (i.e., polynomial-time) algorithm for testing whether two given graphs are isomorphic has withstood all attempts for a solution up to date. The worst case running time of all known algorithms is of exponential order, and just for certain special types of graphs, polynomial-time algorithms have been devised (for further reference see [1, 22, 25]). Although the possibility that Graph Isomorphism (GI) is NP-complete has been discussed [14], strong evidence against this possibility has been provided [29, 4, 8, 16, 18, 33]. In the first place it was proved by Mathon [29] that the decision and counting versions of the problem are polynomial-time equivalent (in the sense of a truth-table reduction) which does not seem to be the case for NP-complete problems. In the second place it has been shown that the assumption Graph Isomorphism is NP-complete implies that the polynomial-time hierarchy collapses to $\mathrm{AM} \subseteq \Pi_2^p$.

Still, a complete complexity classification (positive or negative) of Graph Isomorphism has not yet been achieved, and it has been conjectured that this problem might lie strictly between P and NP-complete.

Although GI is the best known example of a problem with this property, it is not an isolated case. There are many other graph and group theoretic problems related to Graph Isomorphism that lie between P and NP-complete and whose exact complexity is not known either (see [22, 23, 28]). These problems can be divided into different classes depending on whether they seem easier or harder than GI: problems which are reducible to GI (like Graph Automorphism), problems Turing equivalent to GI (so-called isomorphism-complete problems), problems to which GI is Turing reducible (like Group Factorization) and problems that seem to be incomparable with GI (like Group Intersection).

The present work makes a contribution towards a better (structural) complexity classification of Graph Isomporhism and other related problems. Using a group theoretic approach we study certain "counting properties" of these problems which show differences among them and which allow finer classifications than the existing ones. We show that for all the mentioned problems nondeterministic Turing machines can be constructed which have predictable numbers of accepting computation paths; i.e., on input $x$, the machines have either $f_1(x)$ accepting paths if $x$ belongs to the language or $f_2(x)$ accepting paths in the other case. Here, $f_1$ and $f_2$ are polynomial-time computable functions. This fact allows us to relate the Graph Isomorphism problem to the area of counting classes, a field which has received considerable interest and investigation lately. Central are the classes #P [41] of functions that count the number of accepting paths of a nondeterministic Turing machine, and the language class PP (for "probabilistic polynomial time"; see [15]) or, equivalently, CP (for "counting polynomial time"; see [34, 43]) where membership in the language is determined by a polynomial-time computable threshold for the number of accepting computations of a nondeterministic Turing machine. Other important counting classes are $\mathrm{C_{=}P}$ ("exact counting") [43] and $\oplus\mathrm{P}$ ("parity") [31]. Very recently, the class Gap-P of functions that compute the difference between the number of accepting and non-accepting paths of a nondeterministic polynomial-time Turing machine has been considered [12]. This class is basically a generalization of #P ("counting P" [41]) with certain advantages. For example, it allows functions to have negative values and is closed under subtraction.

Some of our results can be best understood in the context of Gap-P functions. We show that for certain problems like Graph Automorphism (GA) there is a nondeterministic, polynomial-time Turing machine $M$ with the following properties:

- For every input of $M$, the difference between the number of accepting and non-accepting computations is either 0 or 1,
- $G \in \mathrm{GA}$ if and only if the difference between the number of accepting and non-accepting computations of $M$ is 1.

From this fact follows immediately that the problem is in the classes $\oplus\mathrm{P}$ and $\mathrm{C_{=}P}$. Observe also that the accepting behaviour of machine $M$ is similar to one for a language in UP (unambiguos NP [40]), where a machine accepting the language has either 0 or 1 accepting paths. The difference is that in the case of UP we count the number of accepting paths (a #P function) and in our case we count the difference between the number of accepting and rejecting paths (a Gap-P function).

A machine with such an accepting mechanism appeared for the first time in [27]. Recently, the class of all the languages for which there is a nondeterministic machine with the above accepting criteria has been considered. This class is called XP ("exact P") in [30] and SPP ("stoic PP") in [12], and the authors motivate the definition of the class as a generalization of UP.

In the case of Graph Isomorphism and some other related problems we are able to construct a nondeterministic, polynomial-time machine $M$ with the following properties:

- For every input pair $(G_1, G_2)$ of graphs with $n$ nodes, the difference between the number of accepting and non-accepting computations of $M$ is either 0 or $f(n)$,
- $(G_1, G_2) \in \mathrm{GI}$ if and only if the difference between the number of accepting and non-accepting computations of $M$ is $f(n)$.

Here, $f$ is a polynomial-time computable function. The class of languages for which there is a machine with such an accepting behaviour was also considered in [12] and called LWPP ("length dependent wide PP"). In the mentioned paper the authors ask whether there are nontrivial examples of languages in SPP. GA seems to be the first such example, whereas GI is the first natural problem in LWPP which is not known to be in SPP.

The classes SPP and LWPP have the property that they are *low* for PP; i.e., they do not provide a PP machine with any additional help when used as oracle in a PP computation. This fact provides one of the main corollaries in this work: Graph Isomorphism and all the related problems are low for PP (in symbols: $\mathrm{PP}^{\mathrm{GI}} = \mathrm{PP}$). It was already known that the problem is low for the class $\Sigma_2^p$ [33].

In the light of Toda's result [37] that $\mathrm{P}^{\mathrm{PP}^{\mathrm{PH}}} = \mathrm{P}^{\mathrm{PP}}$, namely that the entire polynomial hierarchy is low for $\mathrm{P}^{\mathrm{PP}}$, it looks as if lowness for PP is not a surprise. But one should notice that there is a big difference between the classes PP and $\mathrm{P}^{\mathrm{PP}}$. For example, it is not known whether NP is low for PP, i.e., $\mathrm{PP}^{\mathrm{NP}} = \mathrm{PP}$. (This is not even known for $\mathrm{NP} \cap \text{co-NP}$.) Another example of this difference is that on the one hand, PH is included in $\mathrm{P}^{\mathrm{PP}}$, but the largest fragment of the polynomial hierarchy known to be included in PP is $\mathrm{P}^{\mathrm{NP}[\log n]}$ [7]. Intuitively, if a set in NP is low for PP, then this means that one has very strong control over the possible number of accepting computations of a corresponding NP machine for this set. We use the lowness result for GI to provide further evidence for the problem not being NP-complete.

The article is organized as follows. We give in Section 2 the basics of group theory and of complexity theory needed in the rest of the paper.

In Section 3, it is shown that Graph Automorphism is in SPP and therefore is low for any uniformly gap-definable class [12] like $\mathrm{Mod}_k\mathrm{P}$, $\mathrm{C}_=\mathrm{P}$, and PP. Furthermore, we point out a connection of Graph Automorphism to Unique Graph Isomorphism and to the promise problem for 1SAT [11].

The lowness of the Graph Isomorphism problem for PP and $C_{=}P$ is proved in Section 4. As a corollary we obtain that Graph Isomorphism is contained in the class LWPP.

Generalizations of the Graph Automorphism and the Graph Isomorphism problem to other well-known permutation group problems are handled in Section 5, and it is shown that they behave exactly like the versions they generalize. Finally, we mention in Section 6 some further consequences of our results.

## 2. Notation and Preliminaries

**2.1. Group Theory.** We assume some familiarity with groups (see [20] or any other introductory book on group theory). We will denote groups by upper case Roman letters and elements of the groups by lower case Greek letters. The order of a group $A$, i.e. the number of its elements, is represented by $|A|$. Let $A$ and $B$ be two groups. The expression $B < A$ denotes that $B$ is a subgroup of $A$. If $\varphi$ is an element of $A$ then the set

$$B\varphi = \{\pi\varphi : \pi \in B\}$$

is a subset of $A$ called a *right coset of $B$ in $A$*. Two right cosets $B\varphi_1$, $B\varphi_2$ are either disjoint or equal, thus $A$ can be partitioned into right cosets of $B$. This is written as

$$A = B\varphi_1 + B\varphi_2 + \ldots + B\varphi_k.$$

The cardinality of any right coset of $B$ is equal to the order of $B$. The set $\{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ is called a *complete right transversal for $B$ in $A$*.

We consider only permutation groups. The group of permutations on $\{1, \ldots, n\}$ is denoted by $S_n$. We represent the identity permutation by $id$. Let $A < S_n$ and $i$ be a point in $\{1, \ldots, n\}$, then the *orbit* of $i$ in $A$ is the set

$$\{j : \exists\varphi \in A,\ \varphi(i) = j\}.$$

If $X$ is a subset of $\{1, \ldots, n\}$ and $A < S_n$, then the *pointwise stabilizer of $X$ in $A$* (denoted by $A_{[X]}$) is the set of permutations in $A$ which map points in $X$ to themselves, i.e.,

$$A_{[X]} = \{\varphi \in A : \forall x \in X,\ \varphi(x) = x\}.$$

Clearly, $A_{[X]}$ is a subgroup of $A$. In case that $X = \{x\}$ is a singleton, $A_{[X]}$ is called the stabilizer of $x$ in $A$ and is denoted by $A_{[x]}$. Pointwise stabilizers play an important role in group theoretic algorithms. One can construct a "tower" of stabilizers in the following way: let $X_i = \{1, \ldots, i\}$ and denote $A$ by $A^{(0)}$ and $A_{[X_i]}$ by $A^{(i)}$, then

$$\{id\} = A^{(n)} < A^{(n-1)} < \ldots < A^{(1)} < A^{(0)} = A.$$

We will present nondeterministic algorithms for certain group problems such that the number of accepting computations must be one of two integers which are known beforehand. The following lemma is the key for such a behaviour and because of its importance, we include a proof of it.

LEMMA 2.1. [20, Theorem 5.2.2] *Let* $A = A_{[i]}\pi_1 + A_{[i]}\pi_2 + \ldots + A_{[i]}\pi_d$. *Then* $d$ *is the size of the orbit of* $i$ *in* $A$, *and for all* $\psi \in A_{[i]}\pi, \pi(i) = \psi(i)$.

PROOF. If $\psi \in A_{[i]}\pi$, then there is a permutation $\varphi$ in $A_{[i]}$ such that $\psi = \varphi\pi$. But then $\psi(i) = \pi(\varphi(i)) = \pi(i)$. This proves the second assertion of the lemma and shows that the orbit of $i$ in $A$ is $\{\pi_1(i), \ldots, \pi_d(i)\}$. It remains to show that the values $\pi_1(i), \ldots, \pi_d(i)$ are all different. Assuming $\pi_s(i) = \pi_t(i)$, we can write $\pi_s$ as $\pi_s\pi_t^{-1}\pi_t$. Since $\pi_t^{-1}(\pi_s(i)) = i$, $\pi_s\pi_t^{-1}$ is a permutation in $A_{[i]}$ and it follows that $\pi_s$ is in the same right coset as $\pi_t$ and thus $s = t$. $\square$

In other words, if $\{j_1, \ldots, j_d\}$ is the orbit of $i$ in the permutation group $A$, and if we denote the set of all permutations in $A$ which map $i$ to $j$ by $A_{[i \mapsto j]}$, then we can partition $A$ into $d$ sets of equal cardinality:

$$A = A_{[i \mapsto j_1]} + \cdots + A_{[i \mapsto j_d]}.$$

Thus, $|A| = d * |A_{[i]}|$, and for every $j \in \{1, \ldots, n\}$, the number of permutations in $A$ which map $i$ to $j$ is either 0 or $|A_{[i]}|$. Since $A^{(i)}$ is the stabilizer of $i$ in $A^{(i-1)}$, we can state the following two corollaries.

COROLLARY 2.2. *Let* $d_i$ *be the size of the orbit of* $i$ *in* $A^{(i-1)}, 1 \le i \le n$. *Then the order of* $A$ *is equal to* $\prod_{i=1}^{n} d_i$.

COROLLARY 2.3. *Let* $A < S_n$ *and consider the pointwise stabilizers* $A^{(i)}, 1 \le i \le n$. *Then for every* $j > i$, *the number of permutations* $\varphi \in A^{(i-1)}$ *such that* $\varphi(i) = j$ *is either 0 or* $|A^{(i)}|$.

The union of complete right transversals $T_i$ for the groups $A^{(i)}$ in $A^{(i-1)}, 1 \le i \le n$, forms a generating set for $A$. From this set it can be efficiently decided if a given permutation belongs to $A$, as stated in the next theorem.

THEOREM 2.4. [35, 13] *Let* $T_i$ *be a complete right transversal for* $A^{(i)}$ *in* $A^{(i-1)}, 1 \le i \le n$, *and let* $K_0 = \bigcup_{i=1}^{n} T_i$, *then*

1. *every element* $\pi \in A$ *can be expressed uniquely as a product* $\pi = \varphi_1\varphi_2 \ldots \varphi_n$ *with* $\varphi_i \in T_i$,

2. *from* $K_0$ *membership in* $A$ *can be tested in* $O(n^2)$ *steps,*

3. *the order of* $A$ *can be determined in* $O(n^2)$ *steps.*

Also, the set $K_0$ can be obtained in polynomial time from any given generating set for $A$, using the following result.

THEOREM 2.5. (see [35, 13, 24, 3, 2]) *Let $A < S_n$ given by a generating set $K$. A set $K_0$ satisfying the conditions of the above theorem can be found in $O(n^3 \cdot |K| \cdot (\log n)^c)$ steps for a constant $c$.*

The results about groups that we have mentioned will be the main tool to deal with certain graph problems. We will consider only simple undirected graphs $G = (V, E)$ without self-loops. We denote vertices in $V$ by natural numbers, i.e., if $G$ has $n$ nodes then $V = \{1, \ldots, n\}$. With this convention mappings from the set of nodes of $G$ onto the set of nodes of another graph $G'$ with the same number of vertices can be interpreted as a permutation in $S_n$. Let $G = (V, E)$ be a graph ($|V| = n$). An *automorphism* of $G$ is a permutation $\varphi \in S_n$ that preserves adjacency in $G$, i.e., for every pair of nodes $i, j$, $\{i, j\} \in E$ if and only if $\{\varphi(i), \varphi(j)\} \in E$. The set of automorphisms of $G$, $Aut(G)$, is a subgroup of $S_n$.

Two graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ are *isomorphic* if there is a bijection $\varphi$ between $V_1$ and $V_2$ such that for every pair of nodes $i, j \in V_1$, $\{i, j\} \in E_1$ if and only if $\{\varphi(i), \varphi(j)\} \in E_2$. Let $Iso(G_1, G_2)$ denote the set of all isomorphisms between $G_1$ and $G_2$. The following straightforward lemma relates the number of isomorphisms and the number of automorphisms of two given graphs.

LEMMA 2.6. *Let $G_1$ and $G_2$ be two graphs. If there is an isomorphism between $G_1$ and $G_2$, then, $Iso(G_1, G_2)$ is a right coset of $Aut(G_1)$, and thus $|Iso(G_1, G_2)| = |Aut(G_1)|$.*

The Graph Isomorphism problem is

**Graph Isomorphism (GI):** Given two graphs $G_1$ and $G_2$, decide whether they are isomorphic.

The problem is clearly in the class NP, and it is not known whether it is in P. It is also unknown whether the problem is NP-complete, but this seems to be unlikely since it would imply that the polynomial hierarchy collapses to its second level [8, 33]. In fact, evidence that GI is not NP-complete was given already by Mathon [29] who showed that the decision problem for GI and its counting version (i.e., the problem to compute the number of isomorphisms of two given graphs) are polynomial-time Turing equivalent. This is a remarkable situation since for the known NP-complete problems the corresponding counting version seems to be much harder than the decision version.

**2.2. Complexity Theory.** All languages considered here are over the alphabet $\Sigma = \{0,1\}$. For a string $x \in \Sigma^*$, $|x|$ denotes the length of $x$. We assume the existence of a pairing function $\langle\cdot,\cdot\rangle : \Sigma^* \times \Sigma^* \mapsto \Sigma^*$ which is computable in polynomial time and has inverses also computable in polynomial time. $\langle y_1, y_2, \ldots, y_n\rangle$ stands for $\langle n, \langle y_1, \langle y_2, \ldots, y_n\rangle\rangle\rangle$. For a set $A$, $|A|$ is the cardinality of $A$.

We assume that the reader is familiar with (nondeterministic, polynomial-time bounded, oracle) Turing machines and complexity classes (see [5, 32]). FP is the set of functions computable by a deterministic polynomial time bounded Turing machine.

For a nondeterministic machine $M$ and a string $x \in \Sigma^*$, let $acc_M(x)$ be the number of accepting computation paths of $M$ on input $x$. Analogously, for a nondeterministic oracle machine $M$, an oracle $A$, and a string $x \in \Sigma^*$, $acc_M^A(x)$ is the number of accepting paths of $M^A$ on input $x$.

Next, we give definitions for the complexity classes PP, $\mathrm{C_=P}$, $\oplus$P which are defined so as to consider the number of computation paths of a nondeterministic machine. These classes were first introduced in [15, 43, 31], respectively. The counting classes $\mathrm{Mod}_k\mathrm{P}$, $k \geq 2$, were independently defined in [6, 21] and [9].

A language $L$ is in the class PP if there is a nondeterministic polynomial-time machine $M$ and a function $f \in \mathrm{FP}$ such that for every $x \in \Sigma^*$,

$$x \in L \iff acc_M(x) \geq f(x).$$

PP is called CP in the notation of Wagner [43]. A language $L$ is in the class $\mathrm{C_=P}$ if there is a nondeterministic polynomial-time machine $M$ and a function $f \in \mathrm{FP}$ such that for every $x \in \Sigma^*$,

$$x \in L \iff acc_M(x) = f(x).$$

The class of all languages whose complement is in $\mathrm{C_=P}$ is denoted by $\mathrm{C_{\neq}P}$. Note that $\mathrm{C_{\neq}P}$ is a generalization of NP. A language $L$ is in the class $\mathrm{Mod}_k\mathrm{P}$, $k \geq 2$, if there is a nondeterministic polynomial-time machine $M$ such that for every $x \in \Sigma^*$,

$$x \in L \iff acc_M(x) \not\equiv 0 \pmod{k}.$$

The class $\mathrm{Mod}_2\mathrm{P}$ is also called $\oplus$P ("Parity P") [31].

Closely related to the language class PP is the function class #P, defined by Valiant [41]. A function $f$ is in #P if there is a nondeterministic polynomial-time machine $M$ such that for every $x$ in $\Sigma^*$, $f(x) = acc_M(x)$. Fenner *et al.* [12] defined the *gap* produced by $M$ on input $x$ as

$$gap_M(x) = acc_M(x) - acc_{\overline{M}}(x),$$

where $\overline{M}$ is the same machine as $M$ but with accepting and non-accepting computations interchanged. A function $f$ is in Gap-P if there is a nondeterministic polynomial-time machine $M$ such that for every $x$ in $\Sigma^*$, $f(x) = gap_M(x)$. The arithmetic closure properties of Gap-P are summarized in the following lemma which is proved in [12].

LEMMA 2.7.

1. $\#\mathrm{P} \subseteq \text{Gap-P}$.

2. Gap-P *is closed under subtraction.*

3. *If* $f \in \text{Gap-P}$ *and* $q$ *is a polynomial, then the function*

$$g(x) = \sum_{|y| \leq q(|x|)} f(\langle x, y \rangle)$$

*is in* Gap-P.

4. *If* $f \in \text{Gap-P}$ *and* $q$ *is a polynomial, then the function*

$$g(x) = \prod_{0 \leq y \leq q(|x|)} f(\langle x, y \rangle)$$

*is in* Gap-P.

For a language $L$ and a complexity class $K$ (which has a sensible relativized version $K^{()}$), we will say that $L$ is *low* for $K$ ($L$ is $K$–low) if $K^L = K$. For a language class $C$, $C$ is low for $K$ if for every language $L$ in $C$, $K^L = K$. The following class SPP which is denoted by XP in [30] was also defined in [12] where it is shown that SPP consists exactly of those languages which are low for Gap-P.

DEFINITION 2.8. SPP *is the class of all languages $L$ such that there exists a nondeterministic polynomial-time machine $M$ such that, for all $x$,*

$$\begin{aligned} x \in L &\implies gap_M(x) = 1, \\ x \notin L &\implies gap_M(x) = 0. \end{aligned}$$

THEOREM 2.9. [12, Theorem 5.5]

$$\mathrm{SPP} = \{L \mid \text{Gap-P}^L = \text{Gap-P}\}.$$

A consequence of the above theorem is the lowness of SPP for any uniformly gap-definable class like PP, $\mathrm{C_=P}$, $\mathrm{Mod}_k\mathrm{P}$, and SPP (see [12]). In particular, it follows that SPP is closed under Turing reductions. Another interesting subclass of PP is WPP.

DEFINITION 2.10. [12] WPP *("wide" PP) is the class of all languages $L$ such that there exists a nondeterministic polynomial-time machine $M$ and a function $f > 0$ in* FP *such that for all $x$,*

$$\begin{aligned} x \in L &\implies gap_M(x) = f(x), \\ x \notin L &\implies gap_M(x) = 0. \end{aligned}$$

It is clear that $\mathrm{SPP} \subseteq \mathrm{WPP} \subseteq \mathrm{C_=P} \cap \mathrm{C_{\neq}P}$. Fenner *et al.* have also defined a restricted version LWPP of WPP, where the FP function $f$ depends only on the length of $x$, and they showed that LWPP is low for PP and $\mathrm{C_=P}$.

## 3. Graph Automorphism

We consider the following problem.

**Graph Automorphism (GA):** Given a graph $G$, decide whether the automorphism group of $G$ contains a non-trivial automorphism (i.e., an automorphism different from the identity).

Clearly the problem is in NP, and it is not known whether it belongs to P. Graph Automorphism is polynomial-time truth-table reducible to Graph Isomorphism [29] but it seems to be an easier problem (a reduction in the opposite direction is not known). The special "counting properties" of this problem have been used by Boppana, Hastad and Zachos [8] to show that Graph Automorphism is in the class co-AM. In the next theorem we show another property of this problem.

THEOREM 3.1. *There is a nondeterministic polynomial-time Turing machine $M$ such that*

1. *for every input, the difference between the number of accepting and non-accepting computations of $M$ is either 0 or 1,*
2. $G \in \mathrm{GA}$ *if and only if the difference between the number of accepting and non-accepting computations of $M$ on input $G$ is 0.*

PROOF. Consider the function

$$f(G) = \prod_{1 \le i < j \le n} (|Aut(G)_{[i]}| - |Aut(G)_{[i \mapsto j]}|).$$

First observe that this function is in the class Gap-P. The functions $f_1(G, i) = |Aut(G)_{[i]}|$ and $f_2(G, i, j) = |Aut(G)_{[i \mapsto j]}|$ are clearly in #P. Therefore it follows by Lemma 2.7 that $f \in$ Gap-P.

We show now that $f(G)$ is either 0 or 1 depending on whether $G \in \mathrm{GA}$. If $G \in \mathrm{GA}$ then there is a non-trivial automorphism $\pi \in Aut(G)$ such that $\pi(i) = j$ for some pair of vertices $i, j,\ i < j$. This implies that $\pi \in Aut(G)_{[i \mapsto j]}$ and by Lemma 2.1, $|Aut(G)_{[i]}| = |Aut(G)_{[i \mapsto j]}|$. Hence, at least one of the factors of $f(G)$ is 0 and this implies $f(G) = 0$.

On the other hand, if $G \notin \mathrm{GA}$, then for every pair $i, j,\ i < j$, $Aut(G)_{[i \mapsto j]}$ is empty, i.e., $|Aut(G)_{[i \mapsto j]}| = 0$. Moreover, for every $i$ the only automorphism in $Aut(G)_{[i]}$ is the identity. All the factors of $f(G)$ have value 1 and therefore $f(G) = 1$. □

This result shows that GA is in the class SPP. As a corollary we obtain the lowness properties of the problem with respect to certain counting classes, since the lowness of the class SPP for these classes has been shown in [12].

COROLLARY 3.2. *Graph Automorphism is low for* SPP, $\mathrm{Mod}_k\mathrm{P}$, $\mathrm{C}_=\mathrm{P}$, *and* PP.

It is also known that GA is low for $\Sigma_2^p$, the second level of the polynomial-time hierarchy. This follows from the results GA $\in$ co-AM [8], and NP $\cap$ co-AM being low for $\Sigma_2^p$ [33].

Remarkably, the following problem, which is not known to be in NP $\cup$ co-NP, is polynomial-time truth-table equivalent to Graph Automorphism.

**Unique Graph Isomorphism (UGI):** Given two graphs $G_1$ and $G_2$, decide whether there is a unique isomorphism between $G_1$ and $G_2$.

For the reduction from UGI to GA one has to make three independent queries asking whether $G_1$, $G_2$, and $G_1 \cup G_2$ belong to GA. It can be checked that $\langle G_1, G_2\rangle \in$ UGI if and only if the sequence of answers is "No,No,Yes". For the reduction in the other direction one query is enough since $G \in$ GA if and only if $\langle G, G\rangle \notin$ UGI.

Since the class SPP is closed under Turing reducibility, the reductions imply that Unique Graph Isomorphism is in SPP. As we have mentioned, GA seems to be an easier problem than GI, and it is not known whether GI is polynomial-time Turing reducible to GA. Therefore, a reduction from GI to UGI is not known either. This is another peculiarity of Graph Isomorphism that makes it different from problems that are known to be NP-complete. For example, it is well known that SAT is truth-table equivalent to USAT, the set of Boolean formulas with exactly one satisfying assignment.

We present now a connection between Graph Automorphism and the area of promise problems (see [11]). A promise problem is a formulation of a partial decision problem. It has the structure

**input** $x$
**promise** $Q(x)$
**property** $R(x)$

where $Q$ and $R$ are two predicates. An algorithm solving the promise problem has to decide $R(x)$ supposing that $x$ satisfies predicate $Q$. If $Q(x)$ does not hold then the algorithm can answer anything.

Definition 3.3. *A promise problem is a pair of sets $(Q, R)$. A set $L$ is called a solution to the promise problem $(Q, R)$ if*

$$\forall x(x \in Q \Rightarrow (x \in L \Leftrightarrow x \in R)).$$

Let 1SAT denote the set of formulas with at most one satisfying assignment. We will consider the promise problem (1SAT,SAT). Observe that a solution for this problem has to agree with SAT in the formulas with a unique satisfying assignment as well as in the unsatisfiable formulas. We show that Graph Automorphism is reducible to any solution of (1SAT,SAT).

THEOREM 3.4. *Graph Automorphism is polynomial-time disjunctively reducible to any solution $L$ of the promise problem* (1SAT,SAT).

PROOF. Consider the set $B$ defined as

$$\langle G, i, j\rangle \in B \Leftrightarrow$$
$$i \neq j \text{ and there is an automorphism in } Aut(G)^{(i-1)} \text{ mapping } i \text{ to } j.$$

Clearly, $B$ is in NP and it can be many-one reduced to SAT by a parsimonious transformation [10, 34]; i.e., there is a polynomial-time function $f$ which transforms every input $\langle G, i, j\rangle$ for $B$ into a Boolean formula such that there are as many automorphisms mapping $i$ to $j$ in $Aut(G)^{(i-1)}$ as satisfying assignments for the formula $f(\langle G, i, j\rangle)$.

One can reduce Graph Automorphism to any solution $L$ of the promise problem (1SAT,SAT) using the following procedure.

```
input graph G with n nodes;
for i := n downto 1 do
   for j := i + 1 to n do
      if f(⟨G, i, j⟩) ∈ L then accept end;
   end;
end;
reject;
```

It is clear that this reduction is disjunctive. We show that it works properly. If $G$ is in GA, then there is at least one non-trivial automorphism in $Aut(G)$. Let $i$, $1 \leq i \leq n$, be the largest integer such that there is an automorphism in $Aut(G)^{(i-1)}$ mapping $i$ to a different node $j$. By Corollary 2.3, the number of such automorphisms is equal to $|Aut(G)^{(i)}|$ and this number is 1 since by the hypothesis, the identity is the only automorphism in $Aut(G)^{(i)}$. Therefore $f(\langle G, i, j\rangle)$ has exactly 1 satisfying assignment and thus $f(\langle G, i, j\rangle)$ is in $L$, and $G$ is accepted.

On the other hand, if $G$ is not in GA, then for every $i, j$, $\langle G, i, j\rangle \notin B$, i.e., $f(\langle G, i, j\rangle)$ is an unsatisfiable formula. This means that $L$ answers all its queries negatively, and $G$ is rejected. □

Valiant and Vazirani [42] have proved that NP is included in the probabilistic class R if the promise problem (1SAT,SAT) has a solution in P. Using the above result we can draw another conclusion from this hypothesis.

COROLLARY 3.5. *If the promise problem* (1SAT,SAT) *has a solution in* P*, then Graph Automorphism belongs to* P.

Note that the proof of Theorem 3.4 yields the slightly stronger result that GA can be recognized by a polynomial-time oracle machine $M$ under an NP oracle $A$ such that

- if a query is answered positively, then $M$ immediately accepts,
- $M^A$ asks only queries for which the number of witnesses is either 0 or 1.

As we will see in the next section, the second property is already a sufficient condition for GA to be in SPP (see Theorem 4.2 and Corollary 4.5).

## 4. Graph Isomorphism

We show in this section that the Graph Isomorphism problem is low for the counting classes PP and $\mathrm{C_=P}$. Our first step in this direction is done by the next theorem which shows that GI can be recognized by an oracle Turing machine $M$ asking an NP oracle only about strings $y$ for which the potential number of witnesses is known beforehand. This means that if $y$ is a member of the oracle then the number of witnesses is equal to some specific value. Moreover, this value depends not on the actual query but can be controlled by an additional parameter $m$ in the input to $M$. For this special purpose, we define the problem

$$\mathrm{GI}^* = \{\langle G, H, m\rangle \mid G \text{ and } H \text{ are isomorphic graphs and } m \geq n\},$$

where $n$ is the number of vertices of $G$. Clearly, $\mathrm{GI}^*$ is many-one equivalent to GI. The oracle machine $M$ of Theorem 4.1 recognizes $\mathrm{GI}^*$, and asks its NP oracle only about strings $y$ for which the number of witnesses is either 0 or $m!$. This machine will be used to construct, for a given nondeterministic oracle machine $M'$ which works under oracle GI, another machine $M''$ working under some NP oracle $A$ such that

- $gap_{M'}^{\mathrm{GI}} = gap_{M''}^{A}$,
- for all positively answered queries $y$ which are asked by $M''$ on input $x$ there are exactly $f(0^{|x|})$ witnesses for membership in $A$ ($f \in \mathrm{FP}$).

In a second step, we take the oracle queries out of the computation of $M''$, resulting in a nondeterministic machine $M'''$ such that $gap_{M'''}(x) = g(0^{|x|}) * gap_{M''}^{A}(x)$ for a polynomial-time computable function $g > 0$. From this, it can be easily seen that GI is low for PP and $\mathrm{C_=P}$.

THEOREM 4.1. *There is a deterministic polynomial-time oracle Turing machine $M$ and an oracle set $A \in \mathrm{NP}$ recognized by a nondeterministic polynomial-time machine $N$ such that*

1. *$L(M, A) = \mathrm{GI}^*$,*
2. *$M^A$ on input $\langle G, H, m\rangle$ asks only queries $y$ for which $acc_N(y) \in \{0, m!\}$.*

PROOF. Let $A$ be the oracle set $B \oplus C$ where $B$ defined as

$$\langle G, i, j, k\rangle \in B \Leftrightarrow$$
$$\text{there exists an automorphism in } Aut(G)^{(i-1)} \text{ which maps } i \text{ to } j$$

and

$$C = \{\langle G, H, k\rangle : G \text{ and } H \text{ are isomorphic}\}.$$

Note that the value of $k$ does not affect membership of $\langle G, i, j, k\rangle$ in $B$ and of $\langle G, H, k\rangle$ in $C$. $M$ uses $k$ only to pass to the oracle the information about how many accepting paths should be produced for every automorphism (isomorphism, respectively). More specifically, consider the following nondeterministic polynomial-time machine $N$ for $A$:

**input** $y$;<br>
**if** $y = 0\langle G, i, j, k\rangle$ **then**<br>
    guess a permutation $\pi$ on the vertex set of $G$;<br>
    **if** $\pi \in Aut(G)^{(i-1)}$ **and** $\pi(i) = j$ **then**<br>
        guess $z \in \{1, \ldots, k\}$;<br>
        **accept**;<br>
    **else reject end**;<br>
**else if** $y = 1\langle G, H, k\rangle$ **then**<br>
    guess a permutation $\varphi$ on the vertex set of $G$;<br>
    **if** $\varphi \in Iso(G, H)$ **then**<br>
        guess $z \in \{1, \ldots, k\}$;<br>
        **accept**;<br>
    **else reject end**;<br>
**else reject end**;

If $y = 0\langle G, i, j, k\rangle$ then the number of accepting paths of the algorithm is equal to $k$ times the number of automorphisms of $Aut(G)^{(i-1)}$ mapping $i$ to $j$. Similarly, if $y = 1\langle G, H, k\rangle$ then the number of accepting paths is equal to $k$ times the number of isomorphisms between $G$ and $H$.

Using $B$ as oracle, the following deterministic polynomial-time oracle Turing machine $M$ on input $\langle G, H, m\rangle$ first computes the size of $Aut(G)$ and afterwards asks oracle $C$ whether $G$ and $H$ are isomorphic. The preceding computation of $|Aut(G)|$ is necessary because in order to fulfill condition $ii)$ of the theorem, $M$ has to know the potential number of isomorphisms between $G$ and $H$.

**input** $\langle G, H, m\rangle$;<br>
$n :=$ the number of vertices in $G$;<br>
**if** $m < n$ **then reject end**;<br>
$d := 1$; /* $d$ counts the number of automorphisms of $G$ */<br>
**for** $i := n$ **downto** 1 **do**<br>
/* determine the size $d_i$ of the orbit of $i$ in $Aut(G)^{(i-1)}$ */

$d_i := 1;$
**for** $j := i+1$ **to** $n$ **do**
/* test whether $j$ lies in the orbit of $i$ in $Aut(G)^{(i-1)}$ */
**if** $\langle G, i, j, \lfloor \frac{m!}{d} \rfloor \rangle \in B$ **then** $d_i := d_i + 1$ **end**;
**end**;
$d := d * d_i$; /* now $d$ is the order of $Aut(G)^{(i-1)}$ */
**end**;
**if** $\langle G, H, \lfloor \frac{m!}{d} \rfloor \rangle \in C$ **then accept else reject end**;

For every $i = n, \ldots, 1$, $M$ computes in the inner for-loop the size $d_i$ of the orbit of $i$ in $Aut(G)^{i-1}$. It follows from Corollary 2.3 that whenever $M^A$ makes a query $y = \langle G, i, j, \lfloor \frac{m!}{d} \rfloor \rangle$ to oracle $B$, then there are either 0 or exactly $d = |Aut(G)^{(i)}|$ automorphisms in $Aut(G)^{(i-1)}$ mapping $i$ to $j$. Since for each such automorphism, $N$ branches into $\lfloor \frac{m!}{d} \rfloor$ accepting computations, and since $d$ as the order of a subgroup of $S_n$ divides $n!$,

$$acc_N(0y) = \begin{cases} 0, & y \notin B, \\ m!, & y \in B. \end{cases}$$

As it is stated in Corollary 2.2, the order of $Aut(G)$ is equal to the product $\prod_{i=1}^{n} d_i$, which is computed in $d$. Thus, when $M^A$ makes the last query $y = \langle G, H, \lfloor \frac{m!}{d} \rfloor \rangle$ to oracle $C$, then, according to Lemma 2.6, there are exactly $d = |Aut(G)|$ many isomorphisms between the two graphs $G$ and $H$ if they are isomorphic and 0 otherwise. Since for each isomorphism, $N$ branches into $\lfloor \frac{m!}{d} \rfloor$ accepting computations, condition $ii)$ of the theorem is also fulfilled for this last query. □

The computation of $|Aut(G)|$ in the above proof is based on Mathon's reduction of the Graph Isomorphism counting problem to GI [29]. But whereas his reduction is completely nonadaptive and is therefore a truth-table reduction, in our computation, the answers to previous queries have an influence on how later queries look like, and therefore it is highly adaptive. The adaptiveness seems to be necessary to obtain property $ii)$ above which will be exploited in the next theorem.

In Theorem 4.2, it is shown that a $\mathrm{C}_{\neq}\mathrm{P}$ oracle $A$ can be "removed" from a Gap-$\mathrm{P}^A$ computation that on input $x$ only queries $A$ about strings $y$ whose gap is in the set $\{0, f(x)\}$, where $f$ is a given function in Gap-P. In fact, the cost of removing oracle $A$ is only a multiplicative factor $f(x)^{p(|x|)}$ for some polynomial $p$.

Theorem 4.2. *Let $M$ be a nondeterministic polynomial-time oracle machine and let $A = \{y : gap_N(y) \neq 0\}$ be a set in* $\mathrm{C}_{\neq}\mathrm{P}$. *If there is a function $f \in$* Gap-P *such that $M^A$ on input $x$ only asks queries $y$ for which $gap_N(y) \in \{0, f(x)\}$, then there is a polynomial $q$ such that the function $x \mapsto gap_M^A(x) * f(x)^{q(|x|)}$ is in* Gap-P.

Proof. Let $q$ be a polynomial delimiting the number of queries asked by $M$ on each computation path. We can assume that on input $x$ $M$ makes exactly $q(|x|)$ many queries. The following polynomial time machine $M'$ nondeterministically guesses a path $p$ of the oracle Turing machine $M$. The computation path $p$ is specified by the sequence of nondeterministic choices and the sequence of oracle query answers.

> **input** $x$;
> guess a path $p$ of $M$ on input $x$;
> let $y_1, \ldots, y_{q(|x|)}$ be the queries on $p$ and let $a_1, \ldots, a_{q(|x|)}$ be the corresponding answers;
> generate a gap of size
>
> $$G_p = \begin{cases} \prod_{i=1}^{q(|x|)} g_i, & \text{if } p \text{ is accepting,} \\ -\prod_{i=1}^{q(|x|)} g_i, & \text{if } p \text{ is rejecting,} \end{cases}$$
>
> where
>
> $$g_i = \begin{cases} gap_N(y_i), & \text{if } a_i = \text{“Yes”}, \\ f(x) - gap_N(y_i), & \text{if } a_i = \text{“No”}. \end{cases}$$

We now argue that $gap_{M'}(x) = g(x) * f(x)^{q(|x|)}$. First, consider the case that $M'$ guesses a computation path $p$ on which all queries $y_1, \ldots, y_{q(|x|)}$ are answered correctly, i.e., $gap_N(y_i) \neq 0 \Leftrightarrow a_i =$ “Yes” for $i = 1, \ldots, q(|x|)$. Then, since $gap_N(y) \in \{0, f(x)\}$, it follows that

$$gap_N(y_i) = \begin{cases} f(x), & \text{if } a_i = \text{“Yes”}, \\ 0, & \text{if } a_i = \text{“No”}, \end{cases}$$

and consequently $g_i = f(x)$ for every $i$. Therefore the generated gap is of size $G_p = f(x)^{q(|x|)}$ if $p$ is accepting, and of size $G_p = -f(x)^{q(|x|)}$ if $p$ is rejecting. If $M'$ guesses a path $p$ on which not all queries are answered according to oracle $A$, then let $y_j$ be the first query on $p$ for which the answer $a_j$ is wrong. Since $y_j$ is asked by $M$ on oracle $A$, we know that $gap_N(y_j)$ lies in the set $\{0, f(x)\}$. So

$$gap_N(y_j) = \begin{cases} 0, & \text{if } a_j = \text{“Yes”}, \\ f(x), & \text{if } a_j = \text{“No”}, \end{cases}$$

which implies $g_j = G_p = 0$. □

The proof of Theorem 4.2 is similar in flavor to the proof of Theorem 5.5 in [12] that SPP is low for Gap-P. The crucial observation here is that queries which are made subsequent to a wrong oracle answer don't need to possess the restricted counting properties imposed by PP-low classes like SPP and LWPP.

Now we are ready to prove the main result of this section, namely that for every function $g$ in Gap-P$^{\mathrm{GI}}$ there is a nondeterministic polynomial-time machine whose gap $h$ is a multiple of $g$, i.e., $h(x) = g(x) * f(0^{|x|})$ for a function $f$ in FP. This can be easily derived from the last two theorems and is the key for the lowness properties of Graph Isomorphism with respect to the counting classes PP and $\mathrm{C_=P}$.

THEOREM 4.3. *For every $g \in$ Gap-P$^{\mathrm{GI}}$ there is a function $f > 0$ in* FP *such that the function $x \mapsto g(x) * f(0^{|x|})$ is in* Gap-P.

PROOF. Let $g = gap_M^{\mathrm{GI}}$ for an NP oracle machine $M$ and let $p$ be a polynomial delimiting the running time of $M$. Clearly, there is an NP oracle machine $M'$ which on input $x$ also produces a gap of size $g(x)$, but uses $\mathrm{GI}^*$ as oracle and makes only queries of the form $\langle G, H, p(|x|)\rangle$. Now we can build another oracle machine $M''$ which simulates $M'$ and replaces every query $\langle G, H, p(|x|)\rangle$ to $\mathrm{GI}^*$ by a computation of the P oracle Turing machine described in Theorem 4.1; i.e., $g(x) = gap^A_{M''}$, where $A \in \mathrm{NP}$ is recognized by a nondeterministic polynomial-time machine $N$ such that for all queries $y$ of $M''$ on input $x$ the number of accepting paths of $N(y)$ is either 0 or $p(|x|)!$. Finally, we can apply Theorem 4.2 and we obtain that the function $g(x) * p(|x|)!^{q(|x|)}$ is in Gap-P for some polynomial $q$. □

COROLLARY 4.4. GI *is low for* PP *and* $\mathrm{C_=P}$.

PROOF. Let $L$ be in $\mathrm{PP}^{\mathrm{GI}}$. Then there is a function $g$ in Gap-$\mathrm{P}^{\mathrm{GI}}$ such that for all $x$,

$$x \in L \Leftrightarrow g(x) > 0.$$

According to Theorem 4.3 there is a function $f > 0$ in FP such that the function $x \mapsto g(x) * f(0^{|x|})$ is in Gap-P. But this implies $L \in \mathrm{PP}$ since $g(x) > 0$ if and only if $g(x) * f(0^{|x|}) > 0$. The lowness for $\mathrm{C_=P}$ follows analogously. □

Another corollary of Theorem 4.3 is that Graph Isomorphism is in the class LWPP.

COROLLARY 4.5. *There is a nondeterministic polynomial-time machine $N$ and a function $f > 0$ in* FP *such that*

$$gap_N(\langle G, H\rangle) = \begin{cases} 0, & \text{if } G \not\simeq H, \\ f(0^{|\langle G,H\rangle|}), & \text{if } G \simeq H. \end{cases}$$

PROOF. Clearly, the characteristic function

$$c(\langle G, H\rangle) = \begin{cases} 0, & \text{if } G \not\simeq H, \\ 1, & \text{if } G \simeq H \end{cases}$$

of Graph Isomorphism is in Gap-$\mathrm{P}^{\mathrm{GI}}$. Thus, using Theorem 4.3, there is a function $f > 0$ in FP such that the function $x \mapsto c(x) * f(0^{|x|})$ is in Gap-P. This proves the corollary. □

## 5. General Permutation Group Problems

Hoffmann [22] has shown that the automorphism group of any graph is the intersection of two permutation groups for which generating sets can easily be computed. From this follows that Graph Automorphism is many-one reducible to the following problem.

**Group Intersection:** Given generating sets for two permutation groups $A, B$, decide whether $A \cap B$ contains a non-trivial permutation.

In some sense, Group Intersection can be considered to be a generalization of the Graph Automorphism problem. Hoffmann [22] also shows that Graph Isomorphism is in fact a special case of the following more general problem:

**Double Coset Membership:** Given generating sets for two permutation groups $A, B < S_n$ and two permutations $\pi, \varphi \in S_n$, decide whether $\varphi \in A\pi B$.

Hoffmann [22], [23] and Luks [28] have found many other group problems which are Turing equivalent to the Double Coset Membership problem. The complexity of all these problems seems to be strictly greater than the one of Graph Isomorphism but still below the complexity of the NP-complete problems because the corresponding counting problems are also in the same Turing degree. The following are examples of problems Turing equivalent to Double Coset Membership:

**Coset Intersection:** Given generating sets for the groups $A, B < S_n$ and a permutation $\pi \in S_n$, decide whether $A\pi \cap B$ is empty.

**Group Factorization:** Given generating sets for the groups $A, B < S_n$ and a permutation $\pi \in S_n$, decide whether $\pi \in AB$.

**Number of Factorizations:** Given generating sets for the groups $A, B < S_n$ and a permutation $\pi \in S_n$, determine the number of different factorizations $\pi = \alpha\beta$ with $\alpha \in A$ and $\beta \in B$.

We will show that the results of the previous sections are also true for these new problems and that with respect to the counting properties considered in this paper these problems behave exactly like the versions they generalize. More precisely, we show that there is a gap function with value 0 or 1 depending on whether its input is an instance of Group Intersection, and that there is a gap function with value 0 if its input belongs to Double Coset Membership or value $f(n)$ otherwise ($f \in \mathrm{FP}$). This last result is also true for all the decision problems Turing equivalent to Double Coset Membership.

We start with the Group Intersection problem. To show that this problem has a gap function with 0-1 behaviour we need the following lemmas.

LEMMA 5.1. [22] *Let $A$ and $B$ be two subgroups of $S_n$ and let $\pi \in S_n$ be a permutation. If $\pi \in AB$, then the number of factorizations $\pi = \alpha_1\beta_1 = \alpha_2\beta_2 = \ldots = \alpha_k\beta_k$ where $\alpha_i \in A$ and $\beta_i \in B$, is equal to $|A \cap B|$.*

The following lemma is also proved in [23, Theorem 3.2] in a slightly different form to the presentation below.

LEMMA 5.2. *Let $A$ and $B$ be two subgroups of $S_n$ and let $i, j$ be two integers, $1 \le i < j \le n$. For every pair of permutations $\alpha \in A$, $\beta \in B$ such that $\alpha(i) = \beta(i) = j$, the product $\psi = \alpha\beta^{-1}$ is in $A_{[i]}B_{[i]}$ if and only if there is a permutation $\varphi \in A \cap B$ such that $\varphi(i) = j$.*

We can now prove that Group Intersection behaves exactly like Graph Automorphism.

THEOREM 5.3. *There is a non-deterministic polynomial-time Turing machine $M$ that, receiving as input generating sets $gen(A)$ and $gen(B)$ for the groups $A, B < S_n$, produces the following gap:*

$$gap_M(gen(A), gen(B)) = \begin{cases} 0, & \text{if } A \cap B \neq \{id\}, \\ 1, & \text{otherwise.} \end{cases}$$

PROOF. Given generating sets for $A$ and $B$, for every pair $i, j$ $(1 \le i < j \le n)$, it can be decided in polynomial time whether there are permutations $\alpha_{i,j} \in A$ and $\beta_{i,j} \in B$ mapping $i$ to $j$ (i.e., $\alpha_{i,j}(i) = \beta_{i,j}(i) = j$). If such permutations exist then they can be easily obtained. For every pair $i, j$ $(1 \le i < j \le n)$ let $\psi_{i,j} = \alpha_{i,j}\beta_{i,j}^{-1}$ where $\alpha_{i,j} \in A_{[i \mapsto j]}$ and $\beta_{i,j} \in B_{[i \mapsto j]}$ are permutations as above. If for a pair $i, j$ such permutations do not exist, then $\psi_{i,j}$ is left undefined. Consider the following function

$$f(gen(A), gen(B)) = \prod_{\substack{1 \le i < j \le n, \\ A_{[i \mapsto j]} \neq \emptyset, \\ B_{[i \mapsto j]} \neq \emptyset}} (|A_{[i]} \cap B_{[i]}| - |\{\alpha \in A_{[i]} : \exists \beta \in B_{[i]}, \psi_{i,j} = \alpha\beta\}|).$$

Clearly, the functions

$$f_1(gen(A), gen(B), i) = |A_{[i]} \cap B_{[i]}|$$

and

$$f_2(gen(A), gen(B), i, j) = |\{\alpha \in A_{[i]} : \exists \beta \in B_{[i]}, \psi_{i,j} = \alpha\beta\}|$$

are in the class #P. Thus, by the above considerations, and by Lemma 2.7, it follows that $f$ is in Gap-P.

We show now that this function always has value 0 or 1. If there is a non-trivial permutation $\varphi \in A \cap B$ then for some pair $i, j$, $i < j$, $\varphi(i) = j$. By Lemma 5.2, this implies that $\psi_{i,j} \in A_{[i]}B_{[i]}$. By Lemma 5.1, it follows that $|A_{[i]} \cap B_{[i]}| = |\{\alpha \in A_{[i]} : \exists \beta \in B_{[i]}, \psi_{i,j} = \alpha\beta\}|$. In this case, there is at least one 0-factor in $f$ and $f(gen(A), gen(B)) = 0$.

If $A \cap B = \{id\}$, then for every $i$, $A_{[i]} \cap B_{[i]} = \{id\}$. Moreover, by Lemma 5.2, for every pair $i, j$, we have $\psi_{i,j} \notin A_{[i]}B_{[i]}$ and therefore $|\{\alpha \in A_{[i]} : \exists \beta \in B_{[i]}, \psi_{i,j} = \alpha\beta\}| = 0$. It follows that $f(gen(A), gen(B)) = 1$. $\square$

We show next that the generalizations of the Graph Isomorphism problem are also in the class LWPP and therefore low for PP and $C_=P$. For this we prove first that the size of the intersection of two permutation groups (given by generating sets) can be computed by a deterministic machine querying an oracle for which the potential number of accepting paths can be computed beforehand.

THEOREM 5.4. *Let $A, B < S_n$ be two groups given by generating sets $gen(A)$, $gen(B)$. The function $f(\langle gen(A), gen(B)\rangle) = |A \cap B|$ can be computed in polynomial time by a deterministic Turing machine $M$ querying an oracle set $L \in \mathrm{NP}$, where $L$ is recognized by a nondeterministic machine $N$ such that $M^L$ asks on input $\langle gen(A), gen(B)\rangle$ only queries $y$ for which $acc_N(y) \in \{0, n!\}$.*

PROOF. The proof proceeds along the lines of the proof of Theorem 4.1. Consider the family of pointwise stabilizers $A^{(i)}, B^{(i)}$, $1 \le i \le n$, and denote $C = A \cap B$ and $C^{(i)} = A^{(i)} \cap B^{(i)}$. Let $\Delta_i$ be the orbit of $i$ in $C^{(i-1)}$. We want to obtain the size of $C$, and by Corollary 2.2 $|C| = \prod_{i=1}^{n} |\Delta_i|$. Inductively we can compute $\Delta_i$ using the following subroutine to decide whether there is a permutation in $C^{(i)}$ mapping $i$ to $j$.

**input** $\langle i, j\rangle$;
guess a permutation $\varphi$ in $S_n$;
**if** $\varphi(i) = j$, $\varphi \in A^{(i)}$ and $\varphi \in B^{(i)}$ **then accept**
**end**

By Theorems 2.4 and 2.5, the condition of the subroutine can be checked in polynomial time. Also, the number of permutations $\varphi$ satisfying the condition is either 0 or $|C^{(i-1)}|$. Based on this fact, one can construct an algorithm which iteratively computes the size of $C^{(i)}$. The algorithm basically works like machine $M$ in Theorem 4.1, querying the above subroutine (with an additional parameter to ensure that the number of accepting paths of the machine computing the oracle set is either 0 or $n!$). □

As a corollary we obtain that the Group Factorization problem can also be computed by a deterministic machine making oracle queries for which the number of accepting paths can be computed beforehand.

COROLLARY 5.5. *There is a deterministic polynomial-time oracle Turing machine $M'$ and an oracle set $L' \in \mathrm{NP}$ recognized by a nondeterministic polynomial-time machine $N'$ such that*

1. *$L(M', L')$ = Group Factorization,*
2. *On input $\langle gen(A), gen(B), \pi\rangle$ (where $gen(A), gen(B)$ are generating sets for groups $A, B < S_n$ and $\pi \in S_n$), $M'^A$ asks only queries $y$ for which $acc'_N(y) \in \{0, n!\}$.*

PROOF. By Lemma 5.1, the number of factorizations of $\pi$ in $AB$ is either 0 or $|A \cap B|$. By the above theorem, the size of $A \cap B$ can be computed by a deterministic polynomial time machine $M$ with an oracle $L$ in NP such that for every query $y$ made by $M$ the number of accepting paths produced by the nondeterministic machine computing $L$ is either 0 or $n!$. Machine $M'$ computes Group Factorization in the following way: on input $\langle gen(A), gen(B), \pi\rangle$, it first simulates $M$ with oracle $L$ to compute $k = |A \cap B|$. Then it makes one more query to a set $L''$ to decide if there are 0 or $k$ factorizations. This set can be computed by a machine that on every path guesses a possible factorization and then, if it is correct, the machine branches each of these paths into $n!/k$ new ones. The number of accepting paths for this last query is therefore again either 0 or $n!$. □

Using Theorem 4.2 and Corollary 5.5 we obtain that Group Factorization is in LWPP.

COROLLARY 5.6. *There is a nondeterministic polynomial-time machine $N$ and a function $f \in$ FP such that*

$$gap_N(\langle gen(A), gen(B), \pi\rangle) = \begin{cases} 0, & \text{if } \pi \notin AB, \\ f(0^{|\langle gen(A), gen(B), \pi\rangle|}), & \text{if } \pi \in AB. \end{cases}$$

Group Factorization is therefore low for PP and for $C_=P$. Of course this also applies to all the problems Turing reducible to Group Factorization, like the Coset Intersection problem. A summary of our results concerning membership in the PP-low classes SPP and LWPP can be seen in Figure 5.1. The arrows indicate known polynomial-time Turing reductions among the problems.

## 6. Some Consequences of the results

The fact that GI is in the class co-AM and therefore low for $\Sigma_2^p$ [4, 8, 16, 18, 33] provided strong evidence that the problem cannot be NP-complete, since this would imply that the polynomial-time hierarchy collapses to $\Sigma_2^p$. In fact, the collapse can be pushed down a little further to the class AM [8]. From the results of the previous sections we can obtain more evidence that GI is not NP-complete: we show that if this were the case then the polynomial-time hierarchy would be low for the classes PP and $C_=P$. Since it is not known whether lowness for one of the classes implies lowness for the other one, both results are independent. The last one is especially unlikely since $C_=P$ seems to be a very weak class (for example, in [19, 36] relativizations are shown under which $\Sigma_2^p$ or BPP are not even Turing reducible to $C_=P$).

THEOREM 6.1. *If the Graph Isomorphism problem is NP-complete, then the polynomial-time hierarchy is low for PP.*

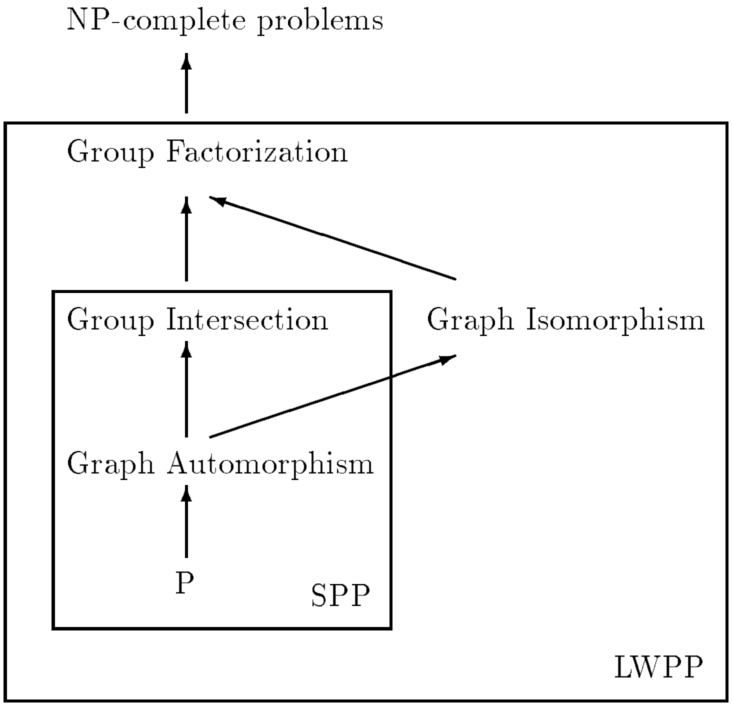


Figure 5.1: Results concerning membership to PP-low classes.

PROOF. As we have mentioned, if GI is NP-complete, then PH collapses to AM, and this class is included in $\mathrm{BPP}^{\mathrm{GI}}$. Therefore, for any set $L \in \mathrm{PH}$, $\mathrm{PP}^L \subseteq \mathrm{PP}^{\mathrm{BPP}^{\mathrm{GI}}}$. The class BPP is low for PP. This fact was shown in [27] using a proof that relativizes. As a consequence

$$\mathrm{PP}^{\mathrm{BPP}^{\mathrm{GI}}} \subseteq \mathrm{PP}^{\mathrm{GI}} \subseteq \mathrm{PP}.$$

The last containment follows from Corollary 4.4. $\square$

The following lemma is just a generalization of Theorem 4.4 from [33] to the class $\mathrm{C_=P}$. Its proof follows exactly the lines of the original one. We just give a sketch of it and refer the interested reader to [33].

LEMMA 6.2. $\mathrm{NP} \cap \mathrm{co\text{-}AM}$ *is low for* $\mathrm{C_=P}^{\mathrm{NP}}$.

PROOF. Let $A \in \mathrm{NP} \cap \mathrm{co\text{-}AM}$ and let $L \in \mathrm{C_=P}^{\mathrm{NP}^A}$. By the quantifier characterization of relativized counting classes [39], there is a deterministic polynomial time bounded oracle machine $M$ such that

$$L = \{x : (\mathrm{C}_{=}y)(\forall z)\langle x, y, z\rangle \in L(M, A)\},$$

where the quantifiers range over all strings of size $p(|x|)$ for some polynomial $p$. (The $\mathsf{C}_=$ quantifier is defined as $(\mathsf{C}_= y)R(y) \Leftrightarrow |\{y \in \Sigma^{p(|x|)} \mid R(y)\}| = 2^{p(|x|)-1}$.) We consider the complementary set

$$\overline{L} = \{x : (\mathsf{C}_{\neq} y)(\forall z)\langle x, y, z\rangle \in L(M, A)\},$$

and amplifying the probability of acceptance in the set $A$, and using quantifier simulation, and swapping quantifiers (see [33] for details) we obtain

$$\overline{L} = \{x : (\exists u)(\mathsf{C}_{\neq} y)(\forall v)(\forall z)\langle x, y, z, u, v\rangle \notin K\},$$

where $K$ is a set in NP.

It was observed by Toda [38] and Green [19] that an existential quantifier can be eliminated when it stands in front of a $\mathsf{C}_{\neq}$ quantifier. Using this fact and contracting the universal quantifiers (including the one implicit in $K$) we obtain that for a certain set $B$ in P,

$$\overline{L} = \{x : (\mathsf{C}_{\neq} y)(\forall z)\langle x, y, z\rangle \in B\}.$$

It follows that the complementary set $L$ belongs to the class $\mathrm{C_=P}^{\mathrm{NP}}$. □

Theorem 6.3. *If the Graph Isomorphism problem is NP-complete, then the polynomial-time hierarchy is low for* $\mathrm{C_=P}$.

Proof. If GI is NP-complete then PH collapses to $\Sigma_2^p$. Therefore, for any set $L \in \mathrm{PH}$,

$$\mathrm{C_=P}^L \subseteq \mathrm{C_=P}^{\Sigma_2^p} \subseteq \mathrm{C_=P}^{\mathrm{NP}^{\mathrm{GI}}}.$$

By the above lemma, $\mathrm{C_=P}^{\mathrm{NP}^{\mathrm{GI}}} = \mathrm{C_=P}^{\mathrm{NP}}$. Using again the hypothesis of the NP completeness of GI, $\mathrm{C_=P}^{\mathrm{NP}} = \mathrm{C_=P}^{\mathrm{GI}}$, but this last class is equal to $\mathrm{C_=P}$ since GI is low for $\mathrm{C_=P}$ (Corollary 4.4). □

At the end of this section we consider some problems related to the well known Graph Reconstruction Conjecture. Recently, these problems were defined by Kratsch and Hemachandra [26] in order to study the complexity-theoretic aspects of graph reconstruction.

Let $G = (V, E)$ be a graph, $V = \{1, \ldots, n\}$. A sequence $\langle G_1, \ldots, G_n\rangle$ of graphs is a *deck* of $G$ if there is a permutation $\pi \in S_n$ such that for every $i = 1, \ldots, n$, $G_{\pi(i)}$ is isomorphic to the one-vertex-deleted-subgraph $(V - \{i\}, E - \{\{i, j\} : j \in V\})$ of $G$. In this case, $G$ is called a *preimage* of the deck $\langle G_1, \ldots, G_n\rangle$. The Reconstruction Conjecture says that for any legitimate deck, there is just one preimage of it, up to isomorphism.

Among other problems, Kratsch and Hemachandra investigated the following decision and counting problems.

**Deck Checking:** Given a graph $G$ and a sequence of graphs $G_i$, $i = 1, \ldots, n$, decide whether $G$ is a preimage of the deck $\langle G_1, \ldots, G_n\rangle$.

**Legitimate Deck:** Given a sequence of graphs $G_i$, $i = 1, \ldots, n$, decide whether there is a preimage $G$ for the deck $\langle G_1, \ldots, G_n \rangle$, i.e., the deck is legitimate.

**Preimage Counting:** Given a sequence of graphs $G_i$, $i = 1, \ldots, n$, compute the number $\mathrm{PCount}(\langle G_1, \ldots, G_n \rangle)$ of all nonisomorphic preimages for the deck $\langle G_1, \ldots, G_n \rangle$.

Kratsch and Hemachandra showed that Deck Checking is many-one reducible to GI which in turn is many-one reducible to Legitimate Deck. They left it as an open question whether there is also a reduction from Legitimate Deck to GI.

Clearly, if the Reconstruction Conjecture holds, then Preimage Counting is not harder then Legitimate Deck. We will show that under this hypothesis Legitimate Deck lies in LWPP, and therefore Legitimate Deck and Preimage Counting are low for PP and $\mathrm{C_=P}$. This follows immediately from the next theorem which shows without any assumption that the function PCount can be computed in Gap-P, modulo a polynomial-time computable factor.

THEOREM 6.4. *There is a function $f$ in* FP *such that the function which maps every deck $\langle G_1, \ldots, G_n \rangle$ to $f(n)$ times the number of nonisomorphic preimages of the deck, i.e.,*

$$\langle G_1, \ldots, G_n \rangle \mapsto f(n) * \mathrm{PCount}(\langle G_1, \ldots, G_n \rangle)$$

*is in* Gap-P.

PROOF. We show that the function which maps $\langle G_1, \ldots, G_n \rangle$ to $n! * \mathrm{PCount}(\langle G_1, \ldots, G_n \rangle)$ can be computed in #P relative to GI. From this, the theorem can be easily obtained using Theorem 4.3. Consider the following nondeterministic algorithm.

> **input** $\langle G_1, \ldots, G_n \rangle$;
> guess a graph $G$ with $n$ vertices;
> **if** $G$ is a preimage of $\langle G_1, \ldots, G_n \rangle$ **then**
>   $d := |Aut(G)|$;
>   guess $k \in \{1, \ldots, d\}$;
>   **accept**;
> **else reject end**;

Since Deck Checking is many-one reducible to GI [26], and since the order of the automorphism group of a given graph can be computed deterministically in polynomial time relative to GI [29], it is possible to implement the above algorithm using a nondeterministic oracle Turing machine $M$ with the oracle GI. $M$ first guesses a graph and tests using oracle GI whether $G$ is a preimage for the given deck $D = \langle G_1, \ldots, G_n \rangle$. Then, for each preimage of $D$, $M$ computes with the help of oracle GI the order of the automorphism group of $G$ and branches into $|Aut(G)|$ different accepting paths.

If $D$ is not a legitimate deck, i.e., there is no preimage for $D$, then all paths of $M$ are rejecting. In the other case, the set $P$ of all preimages of $D$ can be partitioned into sets $P_j$ of isomorphic preimages, $1 \leq j \leq \mathrm{PCount}(D)$. Let $d_j$ be the order of the automorphism group of any preimage in $P_j$, then $|P_j| = n!/d_j$ and therefore the number of accepting paths of $M$ is

$$acc_M^{\mathrm{GI}}(D) = \sum_{j=1}^{\mathrm{PCount}(D)} |P_j| * d_j = n! * \mathrm{PCount}(D).$$

□

COROLLARY 6.5. *If the Reconstruction Conjecture holds, then* Legitimate Deck *is in* LWPP *and therefore low for* $\mathrm{C_{=}P}$ *and* PP.

## Acknowledgements

This work was partially done during a visit of the third author at Universität Ulm supported by the DAAD (Acciones Integradas 1991), and by ESPRIT-II Basic Research Actions Program of the EC under Contract No. 3075 (project ALCOM).

## References

[1] L. BABAI, Moderately exponential bound for graph isomorphism, in *Proc. Fundamentals of Computation Theory Conference*, Lecture Notes in Computer Science #117 (1981), 34–50.

[2] L. BABAI, G. COOPERMAN, L. FINKELSTEIN, E. LUKS, Á. SERESS, Fast Monte Carlo algorithms for permutation groups, in *Proc. 23rd ACM Symposium on Theory of Computing*, 1991, 90–100.

[3] L. BABAI, E. LUKS, Á. SERESS, Fast management of permutation groups, in *Proc. 28th IEEE Symposium on Foundations of Computer Science*, 1988, 272–282.

[4] L. BABAI, S. MORAN, Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes, *Journal of Computer and System Sciences* **36** (1988), 254–276.

[5] J.L. BALCÁZAR, J. DÍAZ, J. GABARRÓ, *Structural Complexity I*, Springer, 1987.

[6] R. BEIGEL AND J. GILL. Counting classes: thresholds, parity, mods, and fewness, *Theoretical Computer Science* **103**, (1992), 3–23.

[7] R. Beigel, L. Hemachandra, G. Wechsung, Probabilistic polynomial time is closed under parity reductions, *Information Processing Letters* **37** (1991), 91–94.

[8] R. Boppana, J. Hastad, and S. Zachos, Does co-NP have short interactive proofs? *Information Processing Letters* **25** (1987), 127–132.

[9] J. Cai, L.A. Hemachandra, On the power of parity, in *Proc. 6th Symposium on Theoretical Aspects of Computer Science,* Lecture Notes in Computer Science # 349 (1989), 229–240.

[10] S.A. Cook, The complexity of theorem-proving procedures, in *Proc. 3rd ACM Symposium on Theory of Computing*, 1971, 151–158.

[11] S. Even, A. Selman Y. Yacobi, The complexity of promise problems with applications to public-key cryptography, *Information and Control* **61** (1984), 114–133.

[12] S. Fenner, L. Fortnow, S. Kurtz, Gap-definable counting classes, in *Proc. 6th Structure in Complexity Theory Conference*, 1991, 30–42.

[13] M. Furst, J. Hopcroft, E. Luks, Polynomial time algorithms for permutation groups, in *Proc. 21st IEEE Symposium on Foundations of Computer Science*, 1980, 36–41.

[14] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*, Freeman, San Francisco, 1979.

[15] J. Gill, Computational complexity of probabilistic Turing machines, *SIAM Journal on Computing* **6** (1977), 675–695.

[16] O. Goldreich, S. Micali, and A. Wigderson, Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, in *Proc. 27th Symposium on Foundations of Computer Science*, 1986, 174–187.

[17] S. Goldwasser, S. Micali, and C. Rackoff, The knowledge complexity of interactive proofs, *SIAM Journal on Computing*, **18**(1) (1989), 186–208.

[18] S. Goldwasser, M. Sipser, Private coins versus public coins in interactive proof systems, in *Randomness and Computation*, S. Micali, editor, volume 5 of *Advances in Computing Research*, JAI Press, 1989, 73–90.

[19] F. Green, On the Power of Deterministic Reductions to $C_=P$, to appear in *Mathematical System Theory*.

[20] M. Hall, *The Theory of Groups*, Macmillan, New York, 1959.

[21] U. Hertrampf, Relations among MOD-classes, *Theoretical Computer Science* **74** (1990), 325–328.

[22] C. Hoffmann, *Group-Theoretic Algorithms and Graph Isomorphism*, Lecture Notes in Computer Science #136, Springer, 1982.

[23] C. Hoffmann, Subcomplete generalizations of graph isomorphism, *Journal of Computer and System Sciences* **25** (1982), 332–359.

[24] M. Jerrum, A compact representation for permutation groups, *Journal of Algorithms* **7** (1986), 60–78.

[25] D.S. Johnson, The NP-completeness column: An ongoing guide, *Journal of Algorithms* **6** (1985), 434–451.

[26] D. Kratsch, L.A. Hemachandra, On the complexity of graph reconstruction, in *Proc. 8th Fundamentals of Computation Theory Conference*, Lecture Notes in Computer Science #529 (1991), 318–328. To appear in *Mathematical Systems Theory*.

[27] J. Köbler, U. Schöning, J. Torán and S. Toda, Turing Machines with few accepting computations and low sets for PP, *Journal of Computer and System Sciences* **44** (1992), 272–286.

[28] E. Luks, Isomorphism of Graphs of Bounded Valence can be tested in Polynomial Time, *Journal of Computer and System Sciences* **25** (1982), 42–65.

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

[30] M. Ogiwara, L. Hemachandra, A complexity theory for closure properties, in *Proc. 6th Structure in Complexity Theory Conference*, 1991, 16–29.

[31] C. Papadimitriou, S. Zachos, Two remarks on the power of counting, in *6th GI Conference on Theoretical Computer Science, Lecture Notes in Computer Science #145* (1983), 269–276.

[32] U. Schöning, *Complexity and Structure,* Lecture Notes in Computer Science #211, Springer, 1986.

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

[34] J. Simon, *On some central problems in computational complexity,* Ph.D. Thesis, Cornell University, 1975.

[35] C. Sims, Computation with permutation groups, in *Proc. 2nd ACM Symposium on Symbolic and Algebraic Manipulations*, 1971, 23–28.

[36] J. Tarui, Degree complexity of boolean functions and its applications to relativized separations, in *Proc. 6th Structure in Complexity Theory Conference*, 1991, 382–390.

[37] S. Toda, PP is as hard as the polynomial-time hierarchy, *SIAM Journal on Computing* **20**(5) (1991), 865–877.

[38] S. Toda, Private communication.

[39] J. Torán, Complexity classes defined by counting quantifiers, *Journal of the ACM* **38** (1991), 753–774.

[40] L.G. Valiant, The relative complexity of checking and evaluating, *Information Processing Letters* **5** (1976), 20–23.

[41] L.G. Valiant, The complexity of computing the permanent, *Theoretical Computer Science* **8** (1979), 189–201.

[42] L.G. Valiant, V.V Vazirani, NP is as easy as detecting unique solutions, *Theoretical Computer Science* **47** (1986), 85–93.

[43] K.W. Wagner, The complexity of combinatorial problems with succinct input representation, in *Acta Informatica* **23** (1986), 325–356.

Manuscript received 29 July 1991

Johannes Köbler
Theoretische Informatik
Universität Ulm
Oberer Eselsberg
89069 Ulm, Germany
koebler@informatik.uni-ulm.de

Uwe Schöning
Theoretische Informatik
Universität Ulm
Oberer Eselsberg
89069 Ulm, Germany
schoenin@informatik.uni-ulm.de

Jacobo Torán
Departamento L.S.I.
Universitat Politècnica de Catalunya
Pau Gargallo 5
E-08028 Barcelona, Spain
jacobo@lsi.upc.es