A nonadaptive NC Checker for Permutation Group Intersection

V. Arvind
Institute of Mathematical Sciences
C. I. T. Campus
Madras 600 113, India

Jacobo Torán
Theoretische Informatik
Universität Ulm
D-89069 Ulm, Germany

November 28, 1996

Abstract

In this paper we design a nonadaptive NC checker for permutation group intersection, sharpening a result from Blum and Kannan [3]. This is a consequence of two results. First we show that a nontrivial permutation in the intersection of two given permutation groups (described by lists of generators) can be computed by an NC algorithm with one round of parallel queries to the Group Intersection problem. Next we design a two-round interactive proof system for the complement of the Group Intersection problem, for which the honest prover can be simulated by an NC algorithm with one round of parallel queries to Group Intersection. As a consequence we also have nonadaptive NC checkers for some related group-theoretic problems.

On the technical side, we define a generalization of wreath products of permutation groups. This product plays a crucial role in the design of the nonadaptive checkers.

1. Introduction

Motivated by the problem of program correctness and reliability, Blum and Kannan introduced in [3] the concept of *program checking.* Rather than trying to *prove* a given program correct for all inputs (which is the approach taken in the area of Program Verification), the approach of program checking is to test and certify a given program for a given input instance. More precisely, a program checker for a given program P is another program that for any instance x of P decides whether the output of P on x is correct or whether P has errors (the formal definition is given later in Section 2). In the course of checking P on x, the program checker might also query the program P on instances different from x.

Blum and Kannan in [3] and subsequent researchers, e.g. [4] have shown that this is a fundamental concept. In [3, 4] and several other papers, efficient (in an appropriate sense depending on the problem) checkers have been designed for several nontrivial problems. Interesting connections between program checking and various other concepts in complexity theory like, for example, interactive proof systems and random-self-reducibility [3, 7] have also been established. In particular, the following basic theorem is in [3].

Theorem 1 *(Checker Characterization Theorem.)* [3] *If a decision problem A and its complement have both interactive proof systems, in each of which the honest prover can be simulated in polynomial time with queries to A, then A has a polynomial-time program checker.*

Together with known results on interactive proofs [16, 5] it follows that all the problems that are complete for the classes PSPACE, PP or $\mathrm{MOD}_k\mathrm{P}$ have polynomial-time checkers.

It is required of the checker that it be significantly more efficient than the program that is being checked. This is a crucial aspect stressed by Blum and Kannan in [3]. In this paper we concentrate on two parameters for measuring the efficiency of a program checker: first, its running time (not counting the time spent in calls to the program), and next, the number of adaptive calls made by the checker to the program. Ideally, we would like to design a checker minimizing both these parameters. In particular, we consider nonadaptive NC checkers. I.e., the program checker is a polynomial size circuit of polylogarithmic depth, with query gates at each of which a call is made to the program being checked. The additional 'nonadaptiveness' property is that on every path from an input to the output gate in the circuit, there is at most one query gate. Alternatively, and more suitable for the description of algorithms, such program checkers can be seen as computed by synchronous PRAMs with a polynomial number of processors in polylogarithmic time. Moreover, each processor is allowed to make just one query to the program being checked, at a specific computation step, *all at the same time.* The notion of NC checkers was also first studied in [3]. In fact, in [3] it is shown that every P-complete problem has nonadaptive NC checkers.

When do problems have nonadaptive NC checkers? We state below a sufficient condition which can be derived easily by adapting the above Checker Characterization Theorem.

Theorem 2 *If a decision problem A and its complement have both 2-round interactive proof systems with randomized NC verifiers, in each of which the honest prover can be simulated in NC with nonadaptive queries to A, then A has a nonadaptive NC program checker.*

The NC checkers in this paper are essentially designed by applying this theorem. As the main result in this paper, we design a nonadaptive NC program checker for the *Group Intersection* problem for permutation groups, and some other related permutation group problems. *Group Intersection* is the following decision problem: given generator sets for two permutation groups $A < S_n$ and $B < S_n$ for some n, decide if there is a nontrivial permutation in $A \cap B$.

Formally, the language associated with the problem is:

$$\mathcal{GINT} = \{(A, B, n) |\ A, B \subseteq S_n \text{ and } \langle A \rangle \cap \langle B \rangle \neq \{id\}\}.$$

As our main results, we present an NC algorithm which takes as input an instance (A, B, n) of $\mathcal{GINT}$ and with one round of queries to $\mathcal{GINT}$ it computes a nontrivial permutation in $\langle A \rangle \cap \langle B \rangle$ (in case one exists). We also design a 2-round interactive protocol for $\overline{\mathcal{GINT}}$ in which the honest prover can be simulated by an NC algorithm with one round of queries to $\mathcal{GINT}$. These results combined with Theorem 2 directly yields a nonadaptive NC checker for $\mathcal{GINT}$.

In Blum and Kannan [3] the authors design a *polynomial-time, adaptive* program checker for a different version of this problem. We will refer to the problem they consider as *Group Intersection Generators* defined as follows:
Given generator sets for two permutation groups $A < S_n$ and $B < S_n$ for some n, compute a *generator set* for $A \cap B$.

It is easy to see that *Group Intersection Generators* is computationally harder problem than *Group Intersection* (*Group Intersection* is easily reducible to *Group Intersection Generators*). However, as we see in this paper, designing a nonadaptive checker for *Group Intersection* is more involved than for *Group Intersection Generators.* There is a heuristic explanation for this phenomenon given in [3]. It is argued that often it is easier to design a checker for an *extension* of the problem than the problem itself. The extension problem, although usually harder than the problem itself, is easier to check because there is more information produced by the program output for the problem. Notice that this is precisely the case with *Group Intersection* and *Group Intersection Generators* (which can be seen as the extension problem in this case). In fact, in [3] the authors highlight this point with a different important example, namely, the GCD problem. They show in a few lines that the 'Extended GCD' problem has an NC checker, leaving open the question whether GCD has an NC checker. Indeed, only recently an NC checker has been designed for GCD based on nontrivial number theory in [1].[1]

Likewise, since the problem *Group Intersection Generators* is an extension of *Group Intersection*, these problems also fall into a similar pattern. Indeed, as we show in this paper, it turns out that we need some nontrivial permutation group theory in order to design a nonadaptive NC checker for *Group Intersection.* In particular, in Section 4. we introduce a novel notion of π-wreath product of permutation groups generalizing the well-known wreath product [12]. Using π-wreath products we are also able to design a nonadaptive NC checker also for *Group Intersection Generators*, improving the result in [3].

[1]In [1], actually an efficient 'constant-query' sequential checker is designed, which also turns out to be implementable in NC.

We now summarize the plan of the paper. In Section 2 we give necessary definitions, and in Section 3 we show that the search problem for $\mathcal{GINT}$ can be solved in parallel with non-adaptive queries. In Section 4 we describe the above-mentioned 2-round interactive protocol for $\overline{\mathcal{GINT}}$. Finally, in Section 5 we give nonadaptive NC checkers for *Group Intersection Generators* and other related group problems.

2. Preliminaries and Notation

We denote the cardinality of a finite set X by $||X||$. Let $\mathbf{N}$ denote the set of natural numbers. We denote by $[n]$ the initial segment $\{1, 2, \cdots, n\}$ of $\mathbf{N}$.

This paper uses basic complexity-theoretic concepts like many-one reducibility, truth-table reducibility, and interactive proof systems. These can be found in standard textbooks like, for example, [2, 15]. A reducibility that is not standard, but will be useful for our proofs, is the NC truth-table reducibility:

Definition 3 For two sets, $A, B \subseteq \Sigma^*$, we say that A is NC truth-table reducible to B, ($A \leq_{tt}^{\mathrm{NC}} B$) if A can be computed by a uniform family of NC circuits with query gates for B, with the additional property that on every path from an input to the output gate, there is at most one query gate. If $A \leq_{tt}^{\mathrm{NC}} B$ and $B \leq_{tt}^{\mathrm{NC}} A$, we say that A and B are NC truth-table equivalent.

We now formally define program checkers.

Definition 4 [3] Let A be a decision problem, a program checker for A, C_A, is a (probabilistic) algorithm that for any program P (supposedly for A) that halts on all instances, for any instance x of A, and for any positive integer k (the security parameter) presented in unary:

i. If P is a correct program, that is, if $P(x) = A(x)$ for all instances x, then with probability $\geq 1 - 2^{-k}$, $C_A(x, P, k)$=Correct.

ii. If $P(x) \neq A(x)$ then with probability $\geq 1 - 2^{-k}$, $C_A(x, P, k)$=Incorrect.

The probability is computed over the sequences of coin flips that C_A could have tossed. Also C_A is allowed to make queries to the program P on some instances.

We recall some group-theoretic definitions and fix the notation. Details can be found, for example, in [10] or any other text on group theory.

We denote groups by upper case letters and elements of the groups by lower case letters. If X is a finite set, the symmetric group of X, S_X denotes the set of permutations of elements of X. If $||X|| = n$ then S_X can be identified with the group S_n of all permutations on $[n]$. The identity permutation is denoted by id (we use id to denote the identity of all groups). For $A \subseteq S_n$, the *permutation group generated by* A is the smallest subgroup of S_n that contains A and is denoted by $\langle A \rangle$. In algorithmic group theory an input group $G < S_n$ is usually presented by a set of generators $A \subseteq S_n$, where, in turn, each generator $\psi \in A$ is represented as a list of n pairs $\langle i, j \rangle \in [n] \times [n]$, describing it as a permutation on $[n]$.

Let G and H be two groups. The expression $H < G$ denotes that H is a subgroup of G. If φ is an element of G then the set $H\varphi = \{\pi\varphi : \pi \in H\}$ is a subset of G called a *right coset* *of H in G.* Two right cosets $H\varphi_1$, $H\varphi_2$ are either disjoint or equal, thus G can be partitioned into right cosets of H in G. This is written as $G = H\varphi_1 + H\varphi_2 + \ldots + H\varphi_k$.

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

If $X \subseteq [n]$ and $G < S_n$, then the *setwise stabilizer of X in G* (denoted by G_X) is the set of permutations in G mapping points in X to points in X, $G_X = \{\varphi \in G : \forall x \in X,\ \varphi(x) \in X\}$.

The *pointwise stabilizer of $[i]$ in G* (denoted by $G^{(i)}$) is the set of permutations in G which fix each element of $[i]$, i.e., $G^{(i)} = \{\varphi \in G : \forall x \in [i],\ \varphi(x) = x\}$.

Clearly, for any $X \subseteq [n]$ and for any $i \in [n]$ the sets G_X and $G^{(i)}$ are subgroups of G. In particular, pointwise stabilizers play an important role in the design algorithms for permutation group problems. A fundamental structure that is often exploited is the following chain of stabilizers subgroups in G.

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

A crucial property is that the union of complete right transversals T_i for the groups $G^{(i)}$ in $G^{(i-1)}, 1 \leq i \leq n$, forms a generator set for G. Such a generator set is called a *strong generator set* for G [9]. A major result which we will use is that given a permutation group G presented by a generator set, testing the membership of any permutation in G can be done in NC [6]. This theorem is stated below.

Theorem 5 [6] *Let $G < S_n$ given by a generating set K. There is an NC algorithm for computing a strong generator set $K_0 = \bigcup_{i=1}^{n} T_i$, where T_i is a complete right transversal for $G^{(i)}$ in $G^{(i-1)}, 1 \leq i \leq n$ such that the following hold.*

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

ii. Given K_0, membership in G of a given permutation can be tested in NC.

A generator set K_0 given by the above theorem is referred to as an *NC-efficient strong generator set.*

Another property of strong generator sets that we need is the following. Given a strong generator set for a group G there is a randomized NC algorithm for sampling elements of G uniformly at random. The algorithm works as follows: given a strong generator set for G, an element $\pi \in G$ can be generated uniformly at random by picking one element φ_i uniformly at random from each right transversal T_i and defining π as the product of all the φ_i's. This provides a uniform generation procedure because, as stated in Theorem 5, every element of G is uniquely expressible as such a product of elements from the strong generator set. Clearly, this can be easily implemented by a randomized NC algorithm in logarithmic time.

We next recall two notions of group products which will be used in the proofs of our results: the direct sum of permutation groups, and the wreath product of permutation groups.

Definition 6 [11] *Let $G_1 < S_{X_1}, G_2 < S_{X_2}, \cdots, G_k < S_{X_k}$ be k permutation groups. The* direct sum *of the groups $G_1, G_2, \cdots G_k$ denoted by $\oplus_{i=1}^{k} G_i$ is a permutation group that acts on the disjoint union $\bigcup_{i=1}^{k} X_i$, and whose elements are written as k-tuples $(g_1, g_2, \ldots, g_k)$, for*

$g_i \in G_i$, $i \in [k]$. An element x in $\bigcup_{i=1}^k X_i$ is permuted by $(g_1, g_2, \ldots, g_k)$ according to the following rule:

$$(g_1, g_2, \ldots, g_k)(x) = g_j(x) \text{ if } x \in X_j$$

It is easy to check that $\oplus_{i=1}^k G_i$ is indeed a permutation group. As a useful example, let $G < S_n$ be a permutation group, and for some $X \subseteq [n]$, let G_X denote the setwise stabilizer of X in G. Then G_X can be expressed as the intersection of G with the direct sum $S_X \oplus S_{([n]-X)}$.

We next define the wreath product of any group $G < S_n$ with the permutation group S_2.[2]

Definition 7 [12] *Let $G < S_n$ be some permutation group. The* wreath product *of G and S_2, which we denote by $\tau(G)$ is a permutation group that acts on the set $[n] \times [2]$. The elements of $\tau(G)$ are written as $\tau(g_1, g_2, \psi)$ for $g_1, g_2 \in G$ and $\psi \in S_2$, where the permutation defined by $\tau(g_1, g_2, \psi)$ on the set $[n] \times [2]$ is as below:*

i. If $\psi = id$ then $\tau(g_1, g_2, \psi)\langle i, j\rangle = \langle g_j(i), j\rangle$, $\forall i \in [n], j \in [2]$.

ii. If $\psi = (1\ 2)$ then $\tau(g_1, g_2, \psi)\langle i, 1\rangle = \langle g_1(i), 2\rangle$ and $\tau(g_1, g_2, \psi)\langle i, 2\rangle = \langle g_2(i), 1\rangle$, $\forall i \in [n]$.

Remark Notice that given any group $G < S_n$ presented by a generator set A, the following set is a generator set for $\tau(G)$:

$$\{\tau(g_1, g_2, \psi) \mid g_1, g_2 \in A, \psi \in S_2\}.$$

It is easy to design a logarithmic space machine[3] that takes A as input and outputs the above generator set for $\tau(G)$.

3. Nonadaptive witness search

In this section we show that the $\mathcal{GINT}$ search problem can be solved by an NC algorithm doing parallel queries to $\mathcal{GINT}$.

We first need the following useful generalization of the *Group Intersection* problem, namely the *Multiple Group Intersection* problem: given the generator sets of $A_1, \ldots, A_k \subseteq S_n$ of some k subgroups of S_n decide whether $\bigcap_{i=1}^k A_i \neq \{id\}$.

$\mathcal{MULTINT} = \{(A_1, \ldots, A_k, n) \mid A_1, \ldots, A_k \subseteq S_n, \bigcap_{i=1}^k A_i \neq \{id\}\}$.

We first show that $\mathcal{MULTINT}$ is log-space many-one equivalent to $\mathcal{GINT}$. In order to prove this result we need a new definition.

Definition 8 *Let G be a subgroup of S_n. Then, for a positive integer k, the* diagonal subgroup *$Diag_k(G)$ of S_{kn} induced by G is the set of permutations*

$$Diag_k(G) = \{(g_1, g_2, \ldots, g_k) \in \oplus_{i=1}^k G \mid \exists g \in G : \ \forall i \in [k]\ g_i = g\}.$$

[2]We do not give the general definition of wreath product between any two groups because we do not require it and it is notationally elaborate. It can be found, for example, in [12].

[3]At various places in this paper, instead of showing NC computability we show the stronger logarithmic space computability, for reasons of convenience.

It is easy to check that $Diag_k(G)$ is indeed a subgroup of S_{kn}.

Lemma 9 *$\mathcal{MULTINT}$ is log-space many-one equivalent to $\mathcal{GINT}$.*

Proof

Let $(A_1, \ldots, A_k, n)$ be an instance of $\mathcal{MULTINT}$. Let G denote the subgroup $\oplus_{i=1}^k \langle A_i \rangle$ of S_{kn} and let H denote the subgroup $Diag_k(S_n)$ of S_{kn}.

Claim. $(A_1, \ldots, A_k, n) \in \mathcal{MULTINT}$ *iff* $G \cap H \neq \{id\}$.

Proof of Claim. If $\psi \in \bigcap_{i=1}^k \langle A_i \rangle$ is a nontrivial element, then consider the element $\pi = (\psi_1, \psi_2, \ldots, \psi_k) \in Diag_k(S_n)$ where $\forall i \in [k]\ \psi_i = \psi$. It is easy to verify from the definition of the direct sum $\oplus_{i=1}^k \langle A_i \rangle$ that $\pi \in G$.

Conversely, if there is an element $\pi \in G \cap H$, then by definition of $Diag_k(S_n)$ there is a permutation $\psi \in S_n$ such that $\pi = (\psi_1, \psi_2, \ldots, \psi_k)$, where $\forall i \in [k]\ \psi_i = \psi$, and by the definition of G, $\psi \in \bigcap_{i=1}^k \langle A_i \rangle$. This completes the proof of the claim. □

To establish the desired reduction it is enough to show that generator sets for the two groups G and H can be computed from the instance $(A_1, \ldots, A_k, n)$ in logarithmic space.

For each $A_i \subseteq S_n$ we define a set of permutations $A_i' \subseteq S_{kn}$ as follows.

$A_i' = \{(\psi_1, \psi_2, \ldots, \psi_k) \in \oplus_{i=1}^k S_n \mid \psi_i \in A_i \text{ and } \psi_j = id \text{ for } j \neq i\}$.

Let $A = \bigcup_{j=1}^k A_i'$. Clearly A is a generator set for G.

Next pick a standard generator set S for S_n, say, $S = \{\eta_1, \eta_2\}$ with $\eta_1 = (1\ 2)$ and $\eta_2 = (1\ 2 \cdots n)\}$. It is easy to see that $H = Diag_k(S_n)$ is generated by the two elements φ_1 and φ_2 of S_{kn}, where $\varphi_1 = (\eta_1, \eta_1, \ldots, \eta_1)$ and $\varphi_2 = (\eta_2, \eta_2, \ldots, \eta_2)$.

Notice that the generator sets A and $\{\varphi_1, \varphi_2\}$ can be constructed in logarithmic space with respect to the input size. Thus the reduction maps the instance $(A_1, \ldots, A_k, n)$ to $(A, \{\varphi_1, \varphi_2\}, kn)$. This completes the proof. ▮

We show now how to find a permutation in the intersection of two groups, $G \cap H$ by an NC algorithm which makes one round of parallel queries to $\mathcal{GINT}$. For this we use the wreath products $\tau(G)$ and $\tau(H)$ acting on the set of pairs $[n] \times [2]$.

Lemma 10 *Let $G, H < S_n$ and suppose $G^{(i)} \cap H^{(i)} = \{id\}$ and $G^{(i-1)} \cap H^{(i-1)} \neq \{id\}$. In this case there is a unique permutation in $G^{(i-1)} \cap H^{(i-1)}$ mapping i to some $j > i$ if and only if $\tau(G^{(i-1)}) \cap \tau(H^{(i-1)}) \cap \tau(S_n)_{\{\langle i,1\rangle, \langle j,2\rangle\}} \cap \tau(S_n)_{\{\langle i,2\rangle, \langle j,1\rangle\}} \neq \{id\}$.* [4]

Proof First, observe that since $G^{(i-1)} \cap H^{(i-1)} \neq \{id\}$ there is a permutation $\varphi \in G^{(i-1)} \cap H^{(i-1)}$ and some $j > i$ such that $\varphi(i) = j$. We claim that φ is the only permutation in $G^{(i-1)} \cap H^{(i-1)}$ such that $\varphi(i) = j$, for if π were another such permutation, then for every $k \leq i$, $\pi^{-1}\varphi(k) = k$, implying that $\pi^{-1}\varphi \in G^{(i)} \cap H^{(i)} = \{id\}$, and hence $\pi = \varphi$.

Next, suppose φ is in $G^{(i-1)} \cap H^{(i-1)}$ such that $\varphi(i) = j$, for some $j > i$. Then consider the permutation $\tau(\varphi, \varphi^{-1}, (1\ 2))$ in $\tau(S_n)$. Clearly, $\tau(\varphi, \varphi^{-1}, (1\ 2)) \in \tau(G^{(i-1)}) \cap \tau(H^{(i-1)})$

[4] Recall from the definitions that $\tau(S_n)_{\{\langle i,1\rangle, \langle j,2\rangle\}}$ is the setwise stabilizer of $\{\langle i,1\rangle, \langle j,2\rangle\}$ in $\tau(S_n)$ and similarly $\tau(S_n)_{\{\langle i,2\rangle, \langle j,1\rangle\}}$.

and $\tau(\varphi, \varphi^{-1}, (1\ 2))$ maps $\langle i, 1\rangle$ to $\langle j, 2\rangle$ and maps $\langle i, 2\rangle$ to $\langle j, 1\rangle$. Thus $\tau(\varphi, \varphi^{-1}, (1\ 2))$ is a permutation in $\tau(S_n)$ that stabilizes both the sets $\{\langle i, 1\rangle, \langle j, 2\rangle\}$ and $\{\langle i, 2\rangle, \langle j, 1\rangle\}$.

For the other direction, suppose there is a nontrivial permutation $\tau(\varphi_1, \varphi_2, \psi)$ in $\tau(G^{(i-1)}) \cap \tau(H^{(i-1)}) \cap \tau(S_n)_{\{\langle i,1\rangle,\langle j,2\rangle\}} \cap \tau(S_n)_{\{\langle i,2\rangle,\langle j,1\rangle\}}$. Notice that $\tau(\varphi_1, \varphi_2, \psi)$ must map $\langle i, 1\rangle$ to $\langle j, 2\rangle$ *and* $\langle i, 2\rangle$ to $\langle j, 1\rangle$. For otherwise i is fixed by both φ_1 and φ_2 forcing them both to belong to $G^{(i)} \cap H^{(i)} = \{id\}$. Furthermore, since ψ is also forced to be id we have that $\tau(\varphi_1, \varphi_2, \psi) = id$, which is a contradiction. Therefore, it follows that $\varphi_1(i) = j$, and hence φ_1 is a nontrivial permutation in $G^{(i-1)} \cap H^{(i-1)}$. ▮

By following a similar argument as in Lemma 10 we can easily prove the following lemma.

Lemma 11 *Let $G^{(i)} \cap H^{(i)} = \{id\}$ and $G^{(i-1)} \cap H^{(i-1)} \neq \{id\}$. Then for any three elements $j, k, l > i$, there is a (unique) permutation in $G^{(i-1)} \cap H^{(i-1)}$ that maps i to j and k to l, if and only if $\tau(G^{(i-1)}) \cap \tau(H^{(i-1)}) \cap \tau(S_n)_{\{\langle i,1\rangle,\langle j,2\rangle\}} \cap \tau(S_n)_{\{\langle i,2\rangle,\langle j,1\rangle\}} \cap \tau(S_n)_{\{\langle k,1\rangle,\langle l,2\rangle}$.*

Notice also that generator sets for the last two groups are easy to obtain, and by Theorem 5 we can obtain NC-efficient generator sets for G and H. From these generator sets it is easy to compute generator sets for $(G^{(i-1)})^2$ and for $(H^{(i-1)})^2$ in NC. The next theorem formalizes these ideas.

Theorem 12 *Given the groups G, $H < S_n$, described by generator sets, with $G \cap H \neq \{id\}$, a permutation φ in $G \cap H$ different from the identity can be found by an NC algorithm that makes one round of parallel queries to $\mathcal{GINT}$.*

Proof Suppose $G \cap H \neq \{id\}$, an algorithm to compute a permutation in the intersection of the two groups works as follows: for every 4-tuple $\langle i, j, k, l\rangle$ with $1 \leq i \leq n$, $j > i$, $k > i$, $l \geq i$ and $l \neq j$, a bunch of processors checks whether

$$\tau(G^{(i-1)}) \cap \tau(B^{(i-1)}) \cap \tau(S_n)_{\{\langle i,1\rangle,\langle j,2\rangle\}} \cap \tau((S_n)_{\{\langle i,2\rangle,\langle j,1\rangle\}} \cap \tau(S_n)_{\{\langle k,1\rangle,\langle l,2\rangle\}}$$

is nontrivial.

We first compute NC-efficient generator sets for groups G and H. Now it is easy to compute generator sets for each of the five groups in the above intersection in NC. Next, the algorithm can check whether the intersection of these five groups is nontrivial by making a suitable query to $\mathcal{MULTINT}$. By Lemma 9, this query can be converted in logarithmic space (and hence in NC) to a single query to $\mathcal{GINT}$. Let us fix the correct value of i satisfying $G^{(i)} \cap H^{(i)} = \{id\}$ and $G^{(i-1)} \cap H^{(i-1)} \neq \{id\}$. This value can easily be computed since it is the largest value of i such that a query $\langle i, j, k, l\rangle$ for some value of j, k and l is answered positively. The answer to the query $\langle i, j, k, l\rangle$ tells whether there is a permutation in $G^{(i-1)} \cap H^{(i-1)}$ mapping i to j and k to l. By Lemma 10, in case there is such a permutation it must be unique, and therefore from the answers to all the queries $\langle i, j, k, l\rangle$ (for the fixed i) a permutation in $G^{(i-1)} \cap H^{(i-1)}$ can be obtained. Notice that the algorithm actually needs to makes these queries to $\mathcal{MULTINT}$ for all possible values of i. Moreover, it can make all the above-mentioned queries to $\mathcal{MULTINT}$ nonadaptively. It is not hard to see that a nontrivial element in the intersection of $G \cap H$ can be recovered from the query answers. Notice that the algorithm makes in all $O(n^4)$ nonadaptive queries to $\mathcal{GINT}$ and the rest of its computation is in NC. ▮

4. Nonadaptive checking

For this section we introduce the following generalization of the wreath product of permutation groups.

Definition 13 *Let $G < S_n$ for $n \in \mathbf{N}$ be a permutation group and let $\pi \in S_n$ be some permutation. The* π-wreath product *$\tau_\pi(G)$ is a permutation group of degree $2n$ acting on the set $[n] \times [2]$. Each element $\tau_\pi(g_1, g_2, \varphi)$ in $\tau_\pi(G)$ is defined by elements $g_1, g_2 \in G$ and $\varphi \in S_2$. The action of the permutation $\tau_\pi(g_1, g_2, \varphi)$ on $[n] \times [2]$ is defined as follows:*

i. if $\varphi = id$ then $\tau_\pi(g_1, g_2, \varphi)\langle i, 1\rangle = \langle g_1(i), 1\rangle$.

ii. if $\varphi = id$ then $\tau_\pi(g_1, g_2, \varphi)\langle i, 2\rangle = \langle \pi g_2 \pi^{-1}(i), 2\rangle$.

iii. if $\varphi = (1\ 2)$ then $\tau_\pi(g_1, g_2, \varphi)\langle i, 1\rangle = \langle \pi g_1(i), 2\rangle$.

iv. if $\varphi = (1\ 2)$ then $\tau_\pi(g_1, g_2, \varphi)\langle i, 2\rangle = \langle g_2 \pi^{-1}(i), 1\rangle$.

Notice that by setting $\pi = id$ the π-wreath product gives us the usual wreath product $\tau(G)$ of G with S_2.

Lemma 14 *For any permutation group $G < S_n$ and permutation $\pi \in S_n$*

i. the set $\tau_\pi(G)$ is indeed a subgroup of S_{2n}.

ii. The subgroup $\{\tau_\pi(g_1, g_2, id) \mid g_1, g_2 \in G\}$ of $\tau_\pi(G)$, when restricted to $[n] \times \{1\}$ is the same as the group G, and when restricted to the set $[n] \times \{2\}$ is the same as the conjugate group $\pi G \pi^{-1}$.

Proof Since $\tau_\pi(G)$ is clearly a subset of S_{2n}, we only need to show that $\tau_\pi(G)$ is closed under composition to prove that it is a group. Let $\tau_\pi(x_1, y_1, \varphi_1)$ and $\tau_\pi(x_2, y_2, \varphi_2)$ be two elements of $\tau_\pi(G)$. We have to consider the following cases:

i. Suppose $\varphi_1 = \varphi_2 = id$. Then $\tau_\pi(x_1, y_1, id)\tau_\pi(x_2, y_2, id)\langle i, 1\rangle = \tau_\pi(x_1, y_1, id)\langle x_2(i), 1\rangle = \langle x_1 x_2(i), 1\rangle = \tau_\pi(x_1x_2, y_1y_2, id)\langle i, 1\rangle$. Similarly, $\tau_\pi(x_1, y_1, id)\tau_\pi(x_2, y_2, id)\langle i, 2\rangle = \tau_\pi(x_1, y_1, id)\langle \pi y_2 \pi^{-1}(i), 2\rangle = \langle \pi y_1 y_2 \pi^{-1}(i), 2\rangle = \tau_\pi(x_1x_2, y_1y_2, id)\langle i, 2\rangle$.

   It follows that $\tau_\pi(x_1, y_1, id)\tau_\pi(x_2, y_2, id) = \tau_\pi(x_1x_2, y_1y_2, id)$.

ii. Suppose $\varphi_1 = \varphi_2 = (1\ 2)$. Then we have, $\tau_\pi(x_1, y_1, (1\ 2))\tau_\pi(x_2, y_2, (1\ 2))\langle i, 1\rangle = \tau_\pi(x_1, y_1, (1\ 2))\langle \pi x_2(i), 2\rangle = \langle y_1 x_2(i), 1\rangle = \tau_\pi(y_1x_2, x_1y_2, id)\langle i, 1\rangle$. Similarly, we have, $\tau_\pi(x_1, y_1, (1\ 2))\tau_\pi(x_2, y_2, (1\ 2))\langle i, 2\rangle = \tau_\pi(x_1, y_1, (1\ 2))\langle y_2\pi^{-1}(i), 1\rangle = \langle \pi x_1 y_2 \pi^{-1}, 2\rangle = \tau_\pi(y_1x_2, x_1y_2, id)\langle i, 2\rangle$.

   It follows that $\tau_\pi(x_1, y_1, (1\ 2))\tau_\pi(x_2, y_2, (1\ 2)) = \tau_\pi(y_1x_2, x_1y_2, id)$.

iii. We next consider the case $\varphi_1 = id$ and $\varphi_2 = (1\ 2)$.

$\tau_\pi(x_1, y_1, id)\tau_\pi(x_2, y_2, (1\ 2))\langle i, 1\rangle = \tau_\pi(x_1, y_1, id)\langle \pi x_2(i), 2\rangle = \langle \pi y_1 x_2(i), 2\rangle$. Similarly, $\tau_\pi(x_1, y_1, id)\tau_\pi(x_2, y_2, (1\ 2))\langle i, 2\rangle = \tau_\pi(x_1, y_1, id)\langle y_2\pi^{-1}(i), 1\rangle = \langle x_1 y_2 \pi^{-1}(i), 1\rangle$.

It follows that $\tau_\pi(x_1, y_1, id)\tau_\pi(x_2, y_2, (1\ 2)) = \tau_\pi(y_1 x_2, x_1 y_2, (1\ 2))$.

Similarly, the other case is symmetric and we have after working out that $\tau_\pi(x_1, y_1, (1\ 2))\tau_\pi(x_2, y_2, id) = \tau_\pi(x_1 x_2, y_1 y_2, (1\ 2))$.

Thus, by the above calculations we have established that $\tau_\pi(G)$ is a subgroup of S_{2n}. The second part of the lemma follows directly from the definition of $\tau_\pi(G)$. ▮

Let $A \subseteq S_n$ be a generator set for a group $G < S_n$ and $\pi \in S_n$ be some permutation. We claim that a logarithmic space machine can compute from A and π a generator set for $\tau_\pi(\langle A\rangle)$. To see this notice that a generator set for $\tau_\pi(\langle A\rangle)$ is

$$A' = \{\tau_\pi(g_1, g_2, \varphi) \mid g_1, g_2 \in A \cup \{id\}, \varphi \in S_2\}.$$

Furthermore, notice that a logarithmic space machine can, for each element $\tau_\pi(g_1, g_2, \varphi)$, which is a permutation on $[n] \times [2]$, list out the $2n$ pairs of elements of $[n] \times [2]$ describing the permutation $\tau_\pi(g_1, g_2, \varphi)$. (The logarithmic space machine simply writes out the pairs using Definition 13.) Thus a generator set for $\tau_\pi(\langle A\rangle)$ can be computed in logarithmic space.

Now, in order to give an interactive proof system for $\overline{\mathcal{GINT}}$ we will make use of the *Coset Intersection* problem.

$\mathcal{COSET} = \{(A, B, \pi, n) \mid \{\pi\}, A, B \subseteq S_n, \pi\langle A\rangle \cap \langle B\rangle \neq \emptyset\}$.

The corresponding *Coset Intersection Search* problem is: given an instance (A, B, π, n), if $(A, B, \pi, n) \in \mathcal{COSET}$, find $\varphi \in \langle A\rangle$ such that $\pi\varphi \in \langle B\rangle$.

In particular, we need the 'unique' version of $\mathcal{COSET}$, namely, the problem defined as:

$$\mathcal{UCOSET} = \{(A, B, \pi, n) \mid \{\pi\}, A, B \subseteq S_n, ||\pi\langle A\rangle \cap \langle B\rangle|| = 1\}.$$

In the interactive proof system for $\overline{\mathcal{GINT}}$ we give below, the prover has to actually solve instances of the $\mathcal{UCOSET}$ problem. It turns out, as we show in the next results, that solving the search problem for $\mathcal{UCOSET}$ can be reduced to $\mathcal{GINT}$. We now prove the crucial result of this section.

Theorem 15 *There is a log-space computable function f such that if $(A, B, \pi, n) \in \mathcal{UCOSET}$ then $f(A, B, \pi, n) = (C, D, m) \in \mathcal{GINT}$ such that $\langle C\rangle \cap \langle D\rangle$ has exactly one nontrivial element. Moreover, given that nontrivial element in $\langle C\rangle \cap \langle D\rangle$ the unique element in $\pi\langle A\rangle \cap \langle B\rangle$ can be computed in logarithmic space.*

Proof

Let (A, B, π, n) be an instance of $\mathcal{UCOSET}$. Let G denote $\langle A\rangle$ and H denote $\langle B\rangle$ respectively. Consider the subgroups $\tau_\pi(G)$ and $\tau(H)$ of S_{2n} Then in holds:

Claim. *$(A, B, \pi, n) \in \mathcal{UCOSET}$ if and only if $G \cap H = \{id\}$, $\pi G \pi^{-1} \cap H = \{id\}$ and $\tau_\pi(G) \cap \tau(H) \neq \{id\}$.*

Proof of Claim. We show first the direction from left to right. Suppose $\langle A, B, \pi, n\rangle \in \mathcal{UCOSET}$, this means that that there is a unique element $g \in G$ and such that $\pi g = h \in H$. For any element $\phi \in G \cap H$ we have $\pi g \phi = h\phi$. Since there is a unique element in $\pi G \cap H$, $g\phi = g$ and it follows $\phi = id$. Similarly, for an element $\varphi \in \pi G \pi^{-1} \cap H$, for some element $g' \in G$ it holds $\varphi = \pi g' \pi^{-1}$. Since $\pi g = h$ we can substitute π by hg^{-1} in the above equality and obtain $g^{-1}g'g = h^{-1}\varphi h$. Therefore $h^{-1}\varphi h$ belongs to $G \cap H = \{id\}$ and it follows $\varphi = id$.

We claim that the element $\tau_\pi(g, g^{-1}, (1\ 2)) \in \tau_\pi(G)$ and $\tau(h, h^{-1}, (1\ 2)) \in \tau(H)$ are the same element different from the identity in S_{2n} and therefore $\tau_\pi(G) \cap \tau(H) \neq \{id\}$. In order to see this first notice that clearly both $\tau_\pi(g, g^{-1}, (1\ 2))$ and $\tau(h, h^{-1}, (1\ 2))$ are different from id because they map points with second component 1 to points with second component 2 and vice-versa. Also, since $\pi g = h$ (and equivalently $g^{-1}\pi^{-1} = h^{-1}$), we have $\tau_\pi(g, g^{-1}, (1\ 2))\langle i, 1\rangle = \langle \pi g(i), 2\rangle = \langle h(i), 2\rangle = \tau(h, h^{-1}, (1\ 2))\langle i, 1\rangle$, for all $i \in [n]$. Similarly, $\tau_\pi(g, g^{-1}, (1\ 2))\langle i, 2\rangle = \langle g^{-1}\pi^{-1}(i), 1\rangle = \langle h^{-1}(i), 1\rangle = \tau(h, h^{-1}, (1\ 2))\langle i, 2\rangle$, for all $i \in [n]$. Thus $\tau_\pi(g, g^{-1}, (1\ 2)) = \tau(h, h^{-1}, (1\ 2))$.

For the direction from right to left, let us suppose $G \cap H = \{id\}$, $\pi G \pi^{-1} \cap H = \{id\}$ and $\tau_\pi(G) \cap \tau(H) \neq \{id\}$. Observe first that $\pi G \cap H$ has at most one element since in case there were two different elements $h_1, h_2 \in H$ for which $\pi g_1 = h_1$ and $\pi g_2 = h_2$ for some elements $g_1, g_2 \in G$, we would have that $h_2^{-1}h_1 = g_2^{-1}g_1$ is a nontrivial element of $G \cap H$, contradicting the hypothesis. Secondly, since we are supposing $\tau_\pi(G) \cap \tau(H) \neq \{id\}$ there must be a nontrivial element $\tau_\pi(g_1, g_2, \varphi) = \tau(h_1, h_2, \varphi)$ in the intersection. We claim that φ cannot be the identity in S_2 since if this were the case, then by the second part of Lemma 14 considering the action of the elements $\tau_\pi(g_1, g_2, \varphi)$ and $\tau(h_1, h_2, \varphi)$, restricted to $[n] \times \{1\}$ and $[n] \times \{2\}$ respectively, we get elements each in the group $G \cap H$ and $\pi G \pi^{-1} \cap H$ respectively. More precisely, we get $g_1 = h_1 \in G \cap H$ and $\pi g_2 \pi^{-1} = h_2 \in \pi G \pi^{-1} \cap H$, and by the hypothesis we obtain the contradiction $\tau_\pi(g_1, g_2, \varphi) = \tau(h_1, h_2, \varphi) = id$. Thus φ must be $(1\ 2)$. Now, the equality $\tau_\pi(g_1, g_2, \varphi) = \tau(h_1, h_2, \varphi)$ immediately yields $\tau_\pi(g_1, g_2, \varphi)\langle i, 1\rangle = \tau(h_1, h_2, \varphi)\langle i, 1\rangle$ which in turn implies $\pi g_1(i) = h_1(i)$ for all $i \in [n]$, which shows that $\pi G \cap H \neq \emptyset$. This proves the claim. □

Continuing with the proof of the theorem, we prove now that if $\langle A, B, \pi, n\rangle \in \mathcal{UCOSET}$ then $\tau_\pi(G) \cap \tau(H)$ has a unique nontrivial element from which the unique element in $\pi A \cap B$ can be obtained in logarithmic space. Observe that from this result, and the fact that generator sets for $\tau_\pi(G)$ and $\tau(H)$ can be computed from generator sets for G and H in logarithmic space, the theorem follows.

In the above Claim we have seen that if there is a unique pair $g \in G$, $h \in H$ with $\pi g = h$ then $G \cap H = \{id\}$, $\pi G \pi^{-1} \cap H = \{id\}$, and $\tau_\pi(g, g^{-1}, (1\ 2))_\pi = \tau(h, h^{-1}, (1\ 2))$. If we are given this element in $\tau_\pi(G) \cap \tau(H)$, we can easily read off (in logarithmic space) both g and h.

Therefore, we need to show that this is the only nontrivial element in $\tau_\pi(G) \cap \tau(H)$. Let us suppose that there is another nontrivial element $\tau_\pi(g_1, g_2, \varphi) = \tau(h_1, h_2, \varphi)$. By the proof of the above lemma we also know that φ must be the permutation $(1\ 2)$. $\tau_\pi(g_1, g_2, \varphi)\langle i, 1\rangle = \tau(h_1, h_2, \varphi)\langle i, 1\rangle$ yields $\pi g_1(i) = h_1(i)$ for all $i \in [n]$, and $\tau_\pi(g_1, g_2, \varphi)\langle i, 2\rangle = \tau(h_1, h_2, \varphi)\langle i, 2\rangle$ yields $g_2\pi^{-1}(i) = h_2(i)$ for all $i \in [n]$. Thus we have $\pi g_1 = h_1$ and $g_2\pi^{-1} = h_2$. From this it follows that $g_1 = g$ and $g_2 = g^{-1}$, thus showing that $\tau_\pi(G) \cap \tau(H)$ contains a unique nontrivial element. ▮

Notice that the Claim in Theorem 15 actually shows a logspace truth-table reduction from $\mathcal{UCOSET}$ to $\mathcal{GINT}$. Observe also that $(A, B, n) \in \mathcal{GINT}$ if and only if $(A, B, id, n) \notin \mathcal{UCOSET}$. We deduce the following corollary. The next corollary follows directly from the firstone, Theorems 12 and 15.

Corollary 16 $\mathcal{UCOSET}$ is NC truth-table equivalent to $\mathcal{GINT}$.

Corollary 17 *The search problem for $\mathcal{UCOSET}$ can be solved by an NC algorithm making nonadaptive queries to $\mathcal{GINT}$.*

We now describe the interactive protocol for $\overline{\mathcal{GINT}}$.

Theorem 18 *There is a two round interactive proof system for $\overline{\mathcal{GINT}}$ with an NC verifier and for which the honest prover can be simulated by an NC algorithm making one round of parallel queries to $\mathcal{GINT}$.*

Proof

We first describe a two-round interactive proof system for $\overline{\mathcal{GINT}}$. Then we show for this two-round interactive proof system, that it suffices to have a prover which is NC truth-table reducible to $\mathcal{GINT}$.

Protocol for $\overline{\mathcal{GINT}}$:

i. Input (A, B, n) (an instance of $\mathcal{GINT}$).

ii. **Verifier**

  (a) Compute NC-efficient strong generator sets A' for $\langle A \rangle$ and B' for $\langle B \rangle$ (using the NC algorithm of [6]).

  (b) Using private coins uniformly randomly pick $x \in \langle A' \rangle$ and $y \in \langle B' \rangle$ and send yx to the prover.

iii. **Prover**: Solve the *Coset Intersection Search* problem for the instance (A', B', yx, n), and send back a solution $\varphi \in \langle A' \rangle$.

iv. **Verifier**: Accept iff $\varphi = y$.

Observe first that all permutations of the form ηy with $\eta \in \langle A \rangle \cap \langle B \rangle$ are solutions for the $\mathcal{COSET}$ instance (A', B', yx, n). If $\langle A \rangle \cap \langle B \rangle \neq \{id\}$ then the prover has to choose between at least 2 equally likely possible solutions and the probability that $\varphi = y$ is at most 1/2. As in the Graph-Nonisomorphism protocol, this probability can be made exponentially small by parallel repetition. On the other hand, for input instances (A, B, n) in $\mathcal{GINT}$, since $\langle A \rangle \cap \langle B \rangle = \{id\}$, it holds for every $x \in \langle A \rangle$ and $y \in \langle B \rangle$ that $\langle A, B, yx, n \rangle$ is a 'yes' instance of $\mathcal{COSET}$ with the additional property that $||yx\langle A' \rangle \cap \langle B' \rangle|| = 1$. Thus, if $\langle A \rangle \cap \langle B \rangle = \{id\}$, it holds that $\langle A, B, yx, n \rangle$ is in $\mathcal{UCOSET}$. From Corollary 17 it is clear that the prover can be simulated by an NC algorithm with one round of parallel queries to $\mathcal{GINT}$. This completes the proof. ▮

Theorems 2 and 15, and the interactive protocol of Theorem 18 yield the following corollary.

Corollary 19 *$\mathcal{GINT}$ has nonadaptive NC checkers.*

5. Nonadaptive NC checkers for related problems

As mentioned in the introduction, in [3] an adaptive polynomial-time checker for *Group Intersection Generators* is given. Using π-wreath products we prove in the following theorem that the checker given in [3] can be modified to obtain a nonadaptive NC checker.

First notice that the polynomial-time adaptive checker described in [3] is developed in two steps: first the authors give a 2-round IP protocol for the *Group Intersection Generators* problem. In their protocol the prover is essentially the *Group Factorization Search* problem. To complete the design of the checker it is shown in [3], using a result of [12], that the prover in the above protocol can be simulated in polynomial-time with *adaptive queries* to *Group Intersection Generators*. Furthermore, it can be seen that the verifier in [3] is essentially an NC verifier.

Thus in order to get a nonadaptive NC checker from the above interactive protocol, it suffices to show that the honest prover can be simulated by an NC algorithm with one round of queries to *Group Intersection Generators*. We prove this below. More precisely, we show that the verifier can in fact ask one (functional) query to *Group Intersection Generators*.[5]

The *Group Factorization Search* problem is defined as follows:
Given as input (A, B, π, n), where $A, B \subseteq S_n$ and $\pi \in S_n$, if $\pi \in \langle A\rangle\langle B\rangle$ then output a factorization $\pi = ab$, where $a \in \langle A\rangle$ and $b \in \langle B\rangle$, else output that $\pi \notin \langle A\rangle\langle B\rangle$.

We obtain the nonadaptive NC checker for *Group Intersection Generators* as a direct consequence of the following result which is of independent interest.

Lemma 20 *There is a log-space computable function f that maps an instance (A, B, π, n) of* Group Factorization Search *to an instance $f(A, B, \pi, n) = (X, Y, m)$ of* Group Intersection Generators *such that, given a generator set S for $\langle X\rangle \cap \langle Y\rangle$, it can be decided in logarithmic space if $\pi \in \langle A\rangle\langle B\rangle$, and if so, a factorization of π in $\langle A\rangle\langle B\rangle$ can also computed in NC.*

Proof Let (A, B, π, n) be an instance of *Group Factorization Search*. Let $G = \langle A\rangle$ and $H = \langle B\rangle$. Consider the π-wreath product $\tau_\pi(H)$ and the wreath product $\tau(G)$ of H and G respectively with S_2 (as defined in Section 4.). Notice that both $\tau_\pi(H)$ and $\tau(G)$ are subgroups of S_{2n}.

Claim. *Let S be any generator set of $\tau_\pi(H) \cap \tau(G)$. Then, $\pi \in GH$ iff the generator set S has an element $\psi = \tau_\pi(h_1, h_2, \varphi) = \tau(g_1, g_2, \varphi)$, where $\varphi = (1\ 2)$. Moreover, for any such generator in S, π has the factorization $\pi = g_1 h_1^{-1}$.*

Proof of Claim. Clearly, if the generator set S has an element $\psi = \tau_\pi(h_1, h_2, \varphi) = \tau(g_1, g_2, \varphi)$, where $\varphi = (1\ 2)$ then it follows by the definition of these elements that $\pi = g_1 h_1^{-1}$. Conversely, suppose $\pi \in GH$. Let $\pi = gh$ be a factorization of π. Consider the element

[5] We are essentially exploiting the fact that *Group Intersection Generators* is a functional problem.

$\psi = \tau_\pi(h^{-1}, h, (1\ 2))$ in $\tau_\pi(H)$. It is easy to check that $\psi = \tau(g, g^{-1}, (1\ 2)) \in \tau(G)$. Hence it follows that $\psi \in \tau_\pi(H) \cap \tau(G)$. Now, since we have exhibited an element $\psi = \tau_\pi(h^{-1}, h, (1\ 2))$ in $\tau_\pi(H) \cap \tau(G)$, it is not possible that for all generators $\tau_\pi(h_1, h_2, \varphi)$ in S $\varphi = id$. Thus there is some generator $\tau_\pi(h_1, h_2, \varphi) = \tau(g_1, g_2, \varphi)$ in S with $\varphi = (1\ 2)$. □

The required function f is now defined as follows: it maps the instance (A, B, π, n) to $(X, Y, 2n)$ where X and Y are generator sets for $\tau_\pi(H)$ and $\tau(G)$ respectively. It is easy to see that f is logspace computable. It is also easy to see that, given a generator set S for $\tau_\pi(H) \cap \tau(G)$, we can pick an appropriate generator from it and compute a factorization of π in NC : each generator in S can be examined in parallel. And as explained in the above claim, one of the generators in S will yield a factorization of π. The factorization itself can be also easily computed in NC. This proves Lemma 20. ▮

Theorem 21 Group Intersection Generators *has a nonadaptive NC checker.*

Proof It clearly suffices to see that we can transform the interactive protocol for *Group Intersection Generators* given in [3] as follows: instead of the checker adaptively querying the program in order to solve the search problem for *Group Factorization*, the checker (by using the logspace computable function of Lemma 20) can solve the same search problem in NC by making just one query to the purported program for *Group Intersection Generators*. Furthermore, as observed in Lemma 20 the checker can also extract the solution to the search problem in NC. Combining these components, we obtain the desired nonadaptive NC checker for *Group Intersection Generators*. ▮

Remark. Notice that the above NC checker for *Group Intersection Generators* has the property that it makes only a *constant* number queries (indeed just one query) to the program being checked. Constant query checkers are highlighted in [1] as a notion of program checking that are practically significant. As already mentioned, it is shown in [1] that GCD has a constant query checker. The NC checker for *Group Intersection Generators* is another nontrivial example of a constant query checker.

The following theorem, which is a technical adaptation of Corollary 4.1.2 in [3], immediately yields nonadaptive NC checkers for some problems that are NC truth-table equivalent to $\mathcal{GINT}$.

Theorem 22 *Let A and B be two decision problems. If A and B are equivalent under NC truth-table reductions and A has nonadaptive NC program checkers, then so does B.*

In particular, from Corollary 16 we know that $\mathcal{UCOSET}$ is NC truth-table equivalent to $\mathcal{GINT}$. Another related problem is *Unique Group Factorization*: $\mathcal{UFACT} = \{(A, B, \pi, n) \mid \{\pi\}, A, B \subseteq S_n \exists$ unique $a \in A, b \in B : \pi = ab\}$. It is easy to see that $(A, B, \pi, n) \in \mathcal{UFACT}$ if and only if $(B, A, \pi, n) \in \mathcal{UCOSET}$. It follows that both problems are logspace many-one equivalent.

Corollary 23 *$\mathcal{UCOSET}$ and $\mathcal{UFACT}$ have nonadaptive NC program checkers.*

6. Discussion

We have obtained nonadaptive NC program checkers for *Group Intersection*, *Group Intersection Generators* and some other related problems.

It is interesting to observe that for the *Coset Intersection* and *Group Factorization* problems, that are probably harder than $\mathcal{GINT}$, but easier than *Group Intersection Generators*, we have nonadaptive NC checkers for their 'unique' solution versions. For the general versions, only adaptive program checkers are known [3]. The situation is summarized in the next table. *Group Intersection* is nonadaptively reducible to *Group Factorization* and this problem is in turn reducible to *Group Intersection Generators*. Nonadaptive reductions in the other directions are not known.

Problem	**Program Checkers**
Group Intersection	nonadaptive, NC
Group Factorization	adaptive, P [3]
Group Intersection Generators	(constant query) nonadaptive, NC

Finally, we note the striking similarity between the results of this paper and the checkability of *Graph Automorphism*, *Graph Isomorphism* and *Graph Automorphism Generators*: from the results of [3] it follows that program checkers for all three problems exist. However, a nonadaptive checker is only known for *Graph Automorphism* [14] (also see [8]). It is also easy to show that *Graph Automorphism Generators* is nonadaptively NC checkable. Designing a nonadaptive checker for *Group Factorization* (or *Graph Isomorphism*) appears to be a challenging open question.

References

[1] L. A. Adleman, H. Huang, K. Kompella, Efficient checkers for number-theoretic computations, *Information and Computation,* **121**, 93–102, 1995.

[2] J. L. Balcázar, J. Díaz, J. Gabarró, *Structural Complexity I & II*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1989.

[3] M. Blum, S. Kannan, Designing programs that check their work, *Journal of the ACM,* **43** 269–291, 1995.

[4] M. Blum, M. M. Luby and R. Rubinfeld, Self-testing/correcting with applications to numerical problems, *J. Comput. Syst. Sci.* **47**, 73–83, 1993.

[5] L. Babai and L. Fortnow, Arithmetization: a new method in structural complexity theory, *Comput. Complexity* **1** 41–66, 1991.

[6] L. Babai, E. Luks and Á. Seress, Permutation Groups in NC, in *Proc 19th ACM Symposium of Theory of Computing*, 409–420, 1987.

[7] J. Feigenbaum, Locally random reductions in interactive complexity, in *Advances of Complexity Theory, DIMACS Series in Discrete Math. and Theoretical Computer Science,* vol. 13, 73–98, AMS, Providence, 1993.

[8] L. Fortnow S. Kannan and S. Mahaney, Personal communication, 1993.

[9] 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.

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

[11] F. Harary, *Graph Theory*, Addison Wesley, Reading, 1969.

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

[13] J. Köbler, U. Schöning, and J. Torán, *Graph Isomorphism: its Structural Complexity,* Birkhäuser, Boston, 1992.

[14] A. Lozano and J. Torán, On the nonuniform complexity of the graph isomorphism problem, in *Proceedings of the Structure in Complexity Theory Conference*, 118–129, 1992.

[15] C. Papadimitriou, *Computational Complexity*, Addison Wesley, 1994.

[16] A. Shamir, IP=PSPACE, *Journal of ACM,* 39(4), 869-877, 1992.