PERMUTATION GRAPHS, FAST FORWARD PERMUTATIONS, AND SAMPLING THE CYCLE STRUCTURE OF A PERMUTATION

BOAZ TSABAN

Abstract. $P \in S_N$ is a *fast forward permutation* if for each m the computational complexity of evaluating $P^m(x)$ is small independently of m and x. Naor and Reingold constructed fast forward pseudorandom cycluses and involutions. By studying the evolution of permutation graphs, we prove that the number of queries needed to distinguish a random cyclus from a random permutation in S_N is $\Theta(N)$ if one does not use queries of the form $P^m(x)$, but is only $\Theta(1)$ if one is allowed to make such queries.

We construct fast forward permutations which are indistinguishable from random permutations even when queries of the form $P^m(x)$ are allowed. This is done by introducing an efficient method to sample the cycle structure of a random permutation, which in turn solves an open problem of Naor and Reingold.

0. Introduction and Motivation

According to Naor and Reingold [1], a permutation $\sigma \in S_N$ is a *fast forward permutation* if for each integer m, and each $x = 0, \dots, N-1$, the computational complexity of evaluating $\sigma^m(x)$ is small and independent of m and x. An important example for such a permutation is the *successor* permutation s defined by

$$s(x) = x + 1 \bmod N,$$

as for each m and x, $s^m(x) = x + m \bmod N$. Observe that s is a *cyclus*, that is, its cycle structure consists of a single cycle of length N.

Throughout this paper, the term *random* is taken with respect to the uniform distribution. In [1], Naor and Reingold consider the following problem[1]: Assume that we have a fast forward permutation $\sigma \in S_N$.

Key words and phrases. permutation graphs, pseudorandom permutations, fast forward permutations, cycle structure.

[1]For the sake of clarity, we will concentrate in the beginning in the (purely) random case, and leave the *pseudo*random case for Part 3.

Assume further we have an oracle[2] $\mathcal{P}$ which fixes a random permutation $P \in S_N$, and for each x can compute $P(x)$ and $P^{-1}(x)$ in time which is polynomial in $\log N$. We wish to use this oracle in order to define a random permutation Q such that:

(1) Q is a random element of the space of all permutations which have the *same cycle structure* as σ.
(2) Q is a *fast forward permutation.*

The solution to this problem is as follows [1]: Define $Q = P\sigma P^{-1}$. Then for each integer m we have that

$$Q^m(x) = P(\sigma^m(P^{-1}(x))),$$

so Q is a fast forward permutation. Moreover, Q has the same cycle structure as σ, and it is not difficult to see that it distributes uniformly among the permutations which have the same cycle structure as σ.

Therefore Naor and Reingold's construction using $\sigma = s$ yields a fast forward random cyclus. The natural question which arises is whether this construction gives a pseudorandom *permutation*. Here by *pseudorandom permutation* we mean that the resulting permutation is difficult to distinguish from a truly random permutation using a limited number (under some reasonable definition of "limited") of calls to the oracle. In Section 4 of [1] it is conjectured that distinguishing a random cyclus in S_N from a random permutation should require roughly $\sqrt{N}$ evaluations. In the forthcoming Section 1 we prove that in the restricted model where only queries of the form $P(x)$ or $P^{-1}(x)$ are allowed (this is the usual model), the task of distinguishing a random cyclus from a random permutation requires roughly N (not $\sqrt{N}$) evaluations.

However, if one wants to allow the usage of the fast forward property in the mentioned construction then the resulting permutation is far from being pseudorandom: In Section 2 we show that a single evaluation is enough to distinguish a random cyclus from a random permutation in the fast forward model (where evaluations of the form $P^m(x)$ are allowed). Therefore, the question of construction of a fast forward pseudorandom permutation is far from having a satisfactory solution. It turns out that a solution of this problem can be obtained by solving another open problem.

After introducing their construction, Naor and Reingold ask whether it is possible to remove the restriction on the cycle structure of the fast

[2] An *oracle* is an algorithm initialized by a fixed unknown initial state, which works as a "black box" by accepting queries of some specific form, and making responses accordingly. (The initial state of the algorithm may change as it runs.) The user of such an algorithm can only know the queries and the responses to them.

forward permutation, that is, whether one can use the oracle $\mathcal{P}$ in order to define a random permutation Q such that:

(1) Q is a random element in the space S_N of *all* permutations.
(2) Q is a *fast forward permutation.*

We give an affirmative solution which is based on an efficient method to sample the cycle structure of a random permutation, together with an introduction of a fast forward permutation for any given cycle structure. This construction yields a fast forward random permutation which is indistinguishable from a random permutation even in the fast forward model.

Part 1. Indistinguishability and distinguishability

This part deals with the evolution of permutation graphs and its application to the indistinguishability of random cycluses from random permutations, and with the distinguishability of random cycles from random permutations when fast forward queries are allowed.

1. The indistinguishability of random cycluses from random permutations

In this section we prove that the number of evaluations of the form $P(x)$ or $P^{-1}(x)$ needed in order to distinguish a random cyclus in S_N from a random permutation in S_N is $\Theta(N)$.

Our proof is best stated in the language of graphs. We first set up the basic notation and facts. As these are fairly natural, the reader may wish to skip directly to Lemma 1.1, and return to the definitions only if an ambiguity occurs.

Throughout this section, $V = \{0, \ldots, N-1\}$ and G (with or without an index) will denote a finite directed graph with V as its set of vertices.

Fix a natural number N. The graph of a (partial) function f from (a subset of) N to N is the directed graph with set of vertices V and with an edge from x to y if, and only if, $f(x) = y$ (for all $x, y \in V$). For convenience we also require that for all $x, y \in V$ there exists at most one edge from x to y, and will write $x \to y$ when there exists an edge from x to y. The graph of a (partial) function will be called a *(partial) function graph*. Observe that there is a natural bijective correspondence between (partial) functions and their graph. A particular case of (partial) function graphs is the (partial) permutation graph, where we require that the (partial) function of the graph is injective.

Let Φ denote the "forgetful" functor assigning to each directed graph G the corresponding undirected graph $\Phi(G)$ (each edge from x to y is replaced by an undirected edge between x and y.) A set C of vertices in G is a *component* if it is a connected component in the undirected graph $\Phi(G)$ (isolated vertices are also components). A component C is *connected* if for each $x, y \in C$ there exists a path from x to y in G.

If G is a partial function graph then each connected component of G is a *cycle*. A permutation graph G of a cyclus will be called a *cyclus graph*. Thus a cyclus graph has a single connected component, and has the form

$$x_0 \to x_1 \to \cdots \to x_{N-1} \to x_0.$$

G is a *partial cyclus graph* if it can be extended to a cyclus graph. A partial cyclus graph is *proper* if it is not a cyclus graph.

The following sequence of observations will play a key role in our proof. We will give proofs only where it seems necessary.

Lemma 1.1. *Let G be a directed graph. The following are equivalent:*

1. *G is a proper partial cyclus graph.*
2. *G is a partial permutation graph with no cycles.*
3. *Each component of G is well-ordered by $\to$.*

Thus if G is a proper partial cyclus graph then each component C of G contains a unique minimal element $\min C$ and a unique maximal element $\max C$.

Lemma 1.2. *Assume that G is a partial cyclus graph with m components. Then there exist exactly $(m-1)!$ cyclus graphs extending G.*

Proof. Let $C_0, \dots, C_{m-1}$ be the components of G.

Fix any *cyclus* $\sigma \in S_m$. For each $i = 0, \dots, m-1$, add an edge from $\max C_{\sigma^i(0)}$ to $\min C_{\sigma^{i+1}(0)}$ to obtain a cyclus graph G^σ. We claim that for distinct cycluses $\sigma, \tau \in S_m$, the graphs G^σ and G^τ are distinct. Indeed, let $i \in \{0, \dots, m-1\}$ be the minimal such that $\sigma^{i+1}(0) \neq \tau^{i+1}(0)$ (observe that $\sigma^0(0) = 0 = \tau^0(0)$.) Then in G^σ there is an edge from $\max C_{\sigma^i(0)}$ to $\min C_{\sigma^{i+1}(0)}$, whereas in G^τ there is not. Thus each cyclus in S_m defines a unique cyclus graph extending G.

On the other hand, each cyclus graph extending G defines a unique well-ordering on G by removing the edge pointing to $\min C_0$, and this well-ordering defines, in turn, a unique cyclus $\sigma \in S_m$ by letting $\sigma^{i+1}(0)$ be the unique k such that there is an edge from $\max C_{\sigma^i(0)}$ to $\min C_k$.

It remains to recall that there exist exactly $(m-1)!$ cycluses in S_m. □

Let $\mathrm{comp}(G)$ and $\mathrm{cyc}(G)$ denote the collection of components and cycles in G, respectively. The following lemma describes the basic steps in the evolution of partial permutation graphs. We use $\uplus$ to denote disjoint union.

Lemma 1.3. *Assume that G is a partial permutation graph, and let $\tilde{G}$ be the new graph obtained by adding a new edge to G. Then $\tilde{G}$ is a partial permutation graph if, and only if, there exist (not necessarily distinct) connected components C_0 and C_1 in G such that the new edge is from $\max C_0$ to $\min C_1$. Moreover,*

1. *If C_0 and C_1 are the same component then $\operatorname{comp}(\tilde{G}) = \operatorname{comp}(G)$, and $\operatorname{cyc}(\tilde{G}) = \operatorname{cyc}(G) \uplus \{C_0\}$. (In particular, $|\operatorname{comp}(\tilde{G})| = |\operatorname{comp}(G)|$, and $|\operatorname{cyc}(\tilde{G})| = |\operatorname{cyc}(G)| + 1$.)*
2. *If C_0 and C_1 are distinct then $\operatorname{cyc}(\tilde{G}) = \operatorname{cyc}(G)$, and $\operatorname{comp}(\tilde{G}) = (\operatorname{comp}(G) \setminus \{C_0, C_1\}) \uplus \{C_0 \cup C_1\}$. (In particular, $|\operatorname{cyc}(\tilde{G})| = |\operatorname{cyc}(G)|$, and $|\operatorname{comp}(\tilde{G})| = |\operatorname{comp}(G)| - 1$.)*

For the following definition, recall our convention that throughout this paper, the term *random* is taken with respect to the uniform distribution.

Definition 1.4. Define the following oracles:

$\mathcal{C}$: Chooses a random cyclus $P \in S_N$, accepts queries of the form $(x, i) \in \{0, \dots, N-1\} \times \{1, -1\}$ and responds with $y = P^i(x)$ for each such query.

$\mathcal{O}_2$: Begins with the empty graph G_0 on $V = \{0, \dots, N-1\}$, accepts queries of the form $(x, i) \in V \times \{1, -1\}$, and constructs a partial cyclus graph on V as follows. In the kth query (x_k, i_k), the oracle responds as follows:

1. If the query was made earlier and answered with y, or a query of the form $(y, -i_k)$ was made earlier and answered with x_k, then the oracle responds with $y_k = y$.
2. Otherwise, the oracle responds as follows (let C_{x_k} denote the component of x_k):
   (a) If $i = 1$ then it chooses a random $C \in \operatorname{comp}(G_k) \setminus \{C_{x_k}\}$, sets $y_k = \min C$, adds the edge $x_k \to y_k$ to G_k to obtain a new graph G_{k+1}, and responds with y_k.
   (b) If $i = -1$ (this is the dual case) then it chooses a random $C \in \operatorname{comp}(G_k) \setminus \{C_{x_k}\}$, sets $y_k = \max C$, adds the edge $y_k \to x_k$ to G_k to obtain a new graph G_{k+1}, and responds with y_k.

A sequence $((x_0, i_0), y_0, \dots (x_k, i_k), y_k)$ is $\mathcal{C}$-*consistent* if the equations $P^{i_j}(x_j) = y_j$ have a solution $P \in S_N$ which is a cyclus. It is *nonrepeating* if there exists no $0 \le j < l \le k$ such that $(x_l, i_l) = (x_j, i_j)$, or $(x_l, i_l) = (y_j, -i_j)$. Thus a nonrepeating sequence is a sequence where Case 1 of $\mathcal{O}_2$ is never activated, that is, a sequence in which each query answer gives new information on the permutation (or its graph). Observe that any consistent sequence can be turned into a shorter nonrepeating sequence which induces the same partial cyclus graph.

Lemma 1.5. *For each nonrepeating $\mathcal{C}$-consistent sequence* $s = ((x_0, i_0), y_0, \dots (x_{k-1}, i_{k-1}), y_{k-1})$,

$$\Pr[s|\mathcal{C}] = (N-k-1)!/(N-1)! = \Pr[s|\mathcal{O}_2],$$

where $\Pr[s|\mathcal{A}]$ *is the probability that the oracle* $\mathcal{A}$ *responds with* y_0 *to* (x_0, i_0)*, then with* y_1 *to* (x_1, i_1)*, ..., and finally with* y_{k-1} *to* (x_{k-1}, i_{k-1}).

Proof. The definition of $\mathcal{C}$-consistency ensures that the sequence s defines a partial cyclus graph. The requirement that s is nonrepeating implies by Lemma 1.3 that each answer to a query reduces the number of components in the induced partial cyclus graph by exactly 1. Thus, after k queries the induced graph has exactly $N-k$ components. By Lemma 1.2, there exist $(N-k-1)!$ cyclus graphs extending the given partial cyclus graph, and therefore the probability of getting s in $\mathcal{C}$ is $(N-k-1)!/(N-1)!$.

Now consider $\mathcal{O}_2$. Again, Lemma 1.3 implies that $|\operatorname{comp}(G_j)| = N-j$ for all j. Given G_j, the probability for a specific consistent answer y_j in the next query to $\mathcal{O}_2$ is $1/(N-j-1)$ (uniform choice of one out of the remaining $N-j-1$ components). Thus,

$$\Pr[s|\mathcal{O}_2] = \frac{1}{N-1} \cdot \frac{1}{N-2} \cdot \dots \cdot \frac{1}{N-k} = \frac{(N-k-1)!}{(N-1)!}.$$

$\square$

We say that two oracles are *equivalent* if there is no way to distinguish between them by making queries to the oracles and analyzing their responses.

Corollary 1.6. *The oracles $\mathcal{C}$ and $\mathcal{O}_2$ are equivalent.*

Definition 1.7. Define the following oracles.

$\mathcal{O}_3$: Initially sets a flag Bad to 0, and begins with the empty graph G_0 on $V = \{0, \dots, N-1\}$. This oracle accepts queries of the form $(x, i) \in V \times \{1, -1\}$, and constructs a partial *permutation* graph on V as follows. In the kth query (x_k, i_k), the oracle responds as follows:

(1) If the query was made earlier and answered with y, or a query of the form $(y, -i_k)$ was made earlier and answered with x_k, then the oracle responds with $y_k = y$.

(2) Otherwise, the oracle responds as follows:

(a) If $i = 1$ then it chooses a random $C \in \mathrm{comp}(G_k)$, sets $y_k = \min C$, adds the edge $x_k \to y_k$ to G_k to obtain a new graph G_{k+1}, and responds with y_k.
(b) If $i = -1$ (this is the dual case) then it chooses a random $C \in \mathrm{comp}(G_k)$, sets $y_k = \max C$, adds the edge $y_k \to x_k$ to G_k to obtain a new graph G_{k+1}, and responds with y_k.

If C is the component of x_k, this oracle sets $\mathsf{Bad} = 1$.

$\mathcal{P}$: Chooses a random permutation $P \in S_N$, accepts queries of the form $(x, i) \in \{0, \ldots, N-1\} \times \{1, -1\}$ and responds with $y = P^i(x)$ for each such query.

A sequence $((x_0, i_0), y_0, \ldots (x_k, i_k), y_k)$ is $\mathcal{P}$-*consistent* if the equations $P^{i_j}(x_j) = y_j$ have a solution $P \in S_N$. The proof of the following is similar to the proof of Lemma 1.5 (in fact, it is simpler) and we omit it.

Lemma 1.8. *For each nonrepeating $\mathcal{P}$-consistent sequence s which corresponds to k queries and replies,*

$$\Pr[s|\mathcal{O}_3] = (N-k)!/N! = \Pr[s|\mathcal{P}].$$

Corollary 1.9. *Oracles $\mathcal{O}_3$ and $\mathcal{P}$ are equivalent.*

For our purposes it seems convenient to use the following notion of a distinguisher. An (information theoretic) *distinguisher* D is a probabilistic algorithm[3] with an unlimited computational power and storage space, which accepts an oracle as input (where there are two possible oracles), makes m queries (where m is some fixed number) to that oracle (the distribution of each query depends only on the sequence of earlier queries and oracle responses), and outputs either 0 or 1 (again, the distribution of the answer depends only on the sequence of queries and oracle responses).

The intended meaning is that the distinguisher's output is its guess as to which of the two possible oracles made the responses. (Thus given two oracles $\mathcal{A}$ and $\mathcal{B}$, $D(\mathcal{A})$ and $D(\mathcal{B})$ are random variables taking values in $\{0, 1\}$.) The natural measure for the effectiveness of the distinguisher in distinguishing between two oracles $\mathcal{A}$ and $\mathcal{B}$ is its *advantage*, defined by

$$|\Pr[D(\mathcal{A}) = 1] - \Pr[D(\mathcal{B}) = 1]|.$$

[3] A *probabilistic algorithm* is an algorithm enhanced by an access to a random number generator, that is, at each stage the algorithm chooses which moves to make next according to some well-defined distribution. Mathematically, a probabilistic algorithm is a random variable, whereas a usual algorithm is a function.

The motivation for this measure is as follows. Assume without loss of generality that $\Pr[D(\mathcal{A}) = 1] \geq \Pr[D(\mathcal{B}) = 1]$. Then by the likelihood test we should decide $x = \mathcal{A}$ if the output of $D(x)$ is 1 and $x = \mathcal{B}$ otherwise. The effectiveness of this decision procedure clearly increases as the difference between $\Pr[D(\mathcal{A}) = 1]$ and $\Pr[D(\mathcal{B}) = 1]$ increases, and this (or any other) procedure is useless when the probabilities are equal. Moreover, it can be proved that the number of times needed to sample $D(x)$ in order to decide whether $x = \mathcal{A}$ or $x = \mathcal{B}$ with a significant level of certainty is $O(1/\epsilon^2)$, where $\epsilon = |\Pr[D(\mathcal{A}) = 1] - \Pr[D(\mathcal{B}) = 1]|$.

Theorem 1.10. *Assume that D is a distinguisher which makes $m < N$ queries to $\mathcal{C}$ or $\mathcal{P}$. Then*

$$|\Pr[D(\mathcal{C}) = 1] - \Pr[D(\mathcal{P}) = 1]| \leq \frac{m}{N}.$$

Proof. By Corollaries 1.6 and 1.9, it suffices to show that $|\Pr[D(\mathcal{O}_2) = 1] - \Pr[D(\mathcal{O}_3) = 1]| \leq \frac{m}{N}$.

Oracles $\mathcal{O}_2$ and $\mathcal{O}_3$ behave identically as long as $\mathsf{Bad} = 0$ in $\mathcal{O}_3$, that is, as long as the component of x_k was not chosen. As long as this is the case, the number of components in the graph reduces by at most 1 with each new query answer (we do not assume that the queries are nonrepeating), and therefore the probability that the component of x_k was not chosen for all $k = 0, \dots, m-1$ is at least

$$\frac{N-1}{N} \cdot \frac{N-2}{N-1} \cdot \dots \cdot \frac{N-m}{N-m+1} = \frac{N-m}{N} = 1 - \frac{m}{N}.$$

Let $p = \Pr[D(\mathcal{O}_2) = 1]$. Then $p = \Pr[D(\mathcal{O}_3) = 1 | \mathsf{Bad} = 0]$, therefore

$$\begin{aligned}
&\Pr[D(\mathcal{O}_3) = 1] = \\
&\quad = \Pr[D(\mathcal{O}_3) = 1|\mathsf{Bad} = 0] \cdot \Pr[\mathsf{Bad} = 0] + \\
&\quad\quad + \Pr[D(\mathcal{O}_3) = 1|\mathsf{Bad} = 1] \cdot \Pr[\mathsf{Bad} = 1] \\
&\quad = p \cdot \Pr[\mathsf{Bad} = 0] + \Pr[D(\mathcal{O}_3) = 1|\mathsf{Bad} = 1] \cdot \Pr[\mathsf{Bad} = 1].
\end{aligned}$$

Thus,

$$\begin{aligned}
&|\Pr[D(\mathcal{O}_2) = 1] - \Pr[D(\mathcal{O}_3) = 1]| = \\
&\quad = |p(1 - \Pr[\mathsf{Bad} = 0]) - \Pr[D(\mathcal{O}_3) = 1|\mathsf{Bad} = 1] \cdot \Pr[\mathsf{Bad} = 1]| \\
&\quad = |p \cdot \Pr[\mathsf{Bad} = 1] - \Pr[D(\mathcal{O}_3) = 1|\mathsf{Bad} = 1] \cdot \Pr[\mathsf{Bad} = 1]| \\
&\quad = |(p - \Pr[D(\mathcal{O}_3) = 1|\mathsf{Bad} = 1]) \cdot \Pr[\mathsf{Bad} = 1]| \leq \\
&\quad = |p - \Pr[D(\mathcal{O}_3) = 1|\mathsf{Bad} = 1]| \cdot \frac{m}{N} \leq \frac{m}{N}.
\end{aligned}$$

□

Corollary 1.11. *For all $\epsilon > 0$, the number of evaluations required to distinguish a random cyclus in S_N from a random permutation in S_N with advantage greater or equal to ϵ is at least $\lfloor \epsilon N \rfloor$.*

Our bound on the distinguisher's advantage cannot be improved. The following theorem shows not only that there exists an optimal strategy (with advantage m/N) for the distinguisher, but that in some sense *all* strategies are optimal, including for example those which do not use queries of the form $(x, -1)$. By "all" we mean those which do not make queries where the responses are known in advance, that is, strategies for which the sequence of queries is nonrepeating. (As we remarked before, any strategy which makes repeating queries can be improved.)

Theorem 1.12 (Optimal strategies)**.** *Consider the following m-step strategy ($m < N$) for a distinguisher D to distinguish between $\mathcal{P}$ and $\mathcal{C}$:*

Queries: *For each $k = 0, \ldots, m-1$, choose any pair $(x_k, i_k) \in V \times \{1, -1\}$ such that the sequence $((x_0, i_0), y_0, \ldots, (x_k, i_k))$ is nonrepeating, and make the query (x_k, i_k).*

Output: *If one of the oracle responses introduced a cycle, the distinguisher outputs 1. Otherwise the distinguisher outputs 0.*

Then the advantage of this distinguisher is m/N. In other words, any strategy which generates only nonrepeating sequences is optimal.

Proof. As the query sequence is nonrepeating, the probability that a cycle is not introduced given that the oracle is $\mathcal{O}_3$ is exactly

$$\frac{N-1}{N} \cdot \frac{N-2}{N-1} \cdot \ldots \cdot \frac{N-m}{N-m+1} = \frac{N-m}{N} = 1 - \frac{m}{N}.$$

Thus $\Pr[D(\mathcal{P}) = 0] = \Pr[D(\mathcal{O}_3) = 0] = 1 - m/N$, and

$$\Pr[D(\mathcal{C}) = 0] - \Pr[D(\mathcal{P}) = 0] = 1 - \left(1 - \frac{m}{N}\right) = \frac{m}{N}.$$

□

2. Cryptanalysis of the Naor-Reingold fast forward cyclus

In this section we show that in the fast forward model (where the distinguisher is allowed to make queries of the form $P^m(x)$), random cycluses can be distinguished from random permutations with advantage $1 - o(1)$, using a single query to the given oracle.

For each N let $d(N)$ denote the number of divisors of N.

Theorem 2.1. *A fast forward random cyclus can be distinguished from a fast forward random permutation with advantage* $1 - d(N)/N$*, using a single query.*

Proof. We will use the following important fact.

Lemma 2.2 (folklore)**.** *Fix an* $x \in \{0, \dots, N-1\}$*. Then the length of the cycle of* x *in a random permutation in* S_N *distributes uniformly in* $\{1, \dots, N\}$*.*

Proof. For each $k = 1, \dots, N$ the probability that the cycle's length is k is

$$\frac{N-1}{N} \cdot \frac{N-2}{N-1} \cdot \dots \cdot \frac{N-(k-1)}{N-(k-2)} \cdot \frac{1}{N-(k-1)} = \frac{1}{N}.$$

□

Assume that P is a random permutation in S_N. By Lemma 2.2, the length a_0 of the cycle of 0 distributes uniformly in $\{1, \dots, N\}$. As there are $d(N)$ divisors of N, the probability that a_0 divides N is $d(N)/N$. Now, $P^N(0) = 0$ if, and only if, a_0 divides N. Thus, the probability that $P^N(0) = 0$ is $d(N)/N$ if P is random, but 1 if P is a cyclus. Therefore, the single query $(0, N)$ is enough to distinguish a random cyclus from a random permutation with advantage $1 - d(N)/N$. □

Example 2.3. If $N = 2^n$ (this is the standard case), then $d(N)/N = (n+1)/2^n$, which is negligible.

$d(N)/N$ converges to 0 quite rapidly as $N \to \infty$. However, for our purposes, the following easy observation is enough.

Proposition 2.4. $d(N)/N = o(1)$*.*

Proof. Observe that for each N, if the factorization of N is $p_1^{e_1} \cdot \dots \cdot p_k^{e_k}$, then $d(N) = (e_1 + 1) \cdot \dots \cdot (e_k + 1)$, thus

$$\frac{d(N)}{N} = \frac{e_1 + 1}{p_1^{e_1}} \cdot \dots \cdot \frac{e_k + 1}{p_k^{e_k}}.$$

For all $N > 1$, as the function $f(x) = (x+1)/N^x$ is decreasing for $x \geq 0$, we have that for all $k \geq 1$, $(k+1)/N^k \leq 2/N \leq 1$.

Fix any $\epsilon > 0$. If N has a prime factor $p \geq 2/\epsilon$, then $d(N)/N \leq 2/p \leq \epsilon$. Otherwise, all prime factors of N are smaller than $c = 2/\epsilon$. Assume that $N = p_1^{e_1} \cdot \dots \cdot p_k^{e_k}$. Then $k \leq c$. Let $e_i = \max\{e_1, \dots, e_k\}$. $N \leq c^{e_1 + \dots + e_k}$, so $ce_i \geq e_1 + \dots + e_k \geq \log_c N$, therefore $e_i \geq h(N) = \log_c N/c$, thus $d(N)/N \leq (e_i + 1)/p_i^{e_i} \leq (h(N)+1)/p_i^{h(N)}$ which is smaller than ϵ for large enough N. □

Remark 2.5. One may suggest the following ad-hoc solution to the problem raised by Theorem 2.1: Simply bound the possible value of m in queries of the form $P^m(x)$ to be $\leq N/k$ for some fixed k. But then $P^N(x)$ can still be computed (using k queries instead of 1), so this solution is not good if we do not want to restrict the value of m too much.

Remark 2.6. Theorem 2.1 can be extended as follows: Fix a cycle structure. Let a_0 be the size of the largest cyclus in this structure, and assume that $P \in S_N$ is a random permutation with the given cycle structure. The probability that an element x appears in a cyclus of size a_0 is (at least) a_0/N. If k is $\Omega(N/a_0)$, then with large probability one of the elements $0, \ldots, k-1$ appears in the cyclus and therefore $P^{a_0}(i) = i$ for some $i \in \{0, \ldots, k-1\}$. But if P is random, then it is conceivable that with a non-negligible probability (it is not straightforward to quantify the term "non-negligible" here), for all $i \in \{0, \ldots, k-1\}$ the cycle lengths do not divide a_0 and therefore $P^{a_0}(i) \neq i$.

Of course, if $a_0 < N/a_0$, then one may simply verify in a_0 calls that the cycle of 0 has size $\leq a_0$. Thus our method works in complexity $O(\min\{a_0, N/a_0\})$.

Remark 2.7. Uzi Vishne has pointed out to me that one can distinguish a random permutation *which is not a cyclus* from a random cyclus in with advantage 1 at the price of increasing the number of queries to $\nu(N) + 1$ (where $\nu(N)$ is the number of prime divisors of N): One simply verifies that for each prime factor p of N, $P^{N/p}(0) \neq 0$, whereas $P^N(0) = 0$. This happens if, and only if, P is a cyclus. (Similar observations apply to Remarks 2.5 and 2.6.)

Observe that in probability $1/N$, a random permutation is a cyclus and therefore one cannot hope to obtain advantage greater than $1 - 1/N$, so this improves the advantage from $1 - d(N)/N$ to $1 - 1/N$ at the price of $\nu(N)$ additional queries. Clearly $\nu(N) \leq \log_2 N$. In fact, by the Hardy-Ramanujan Theorem, $\nu(N)$ is asymptotically close to $\log\log N$ "for almost all N" (we will not give the precise formulation here). Observe that when N is a power of 2 we get here $\nu(N) = 1$, so two queries are enough to distinguish with advantage $1 - 1/N$.

Part 2. **Fast forward random permutations**

This part introduces an efficient method to sample the cycle structure of a random permutation, and its application to the construction of fast forward random permutations.

3. Ordered cycle structures

Definition 3.1. Assume that Ω is a finite, well-ordered set, and $P \in S_\Omega$. Let $C_0, \dots, C_{k-1}$ be all (distinct) cycles of P, ordered by

$$C_i < C_j \Leftrightarrow \min C_i < \min C_j.$$

Then the *ordered cycle structure* of P, $\mathrm{OCS}(P)$, is the sequence $(|C_0|, \dots, |C_{k-1}|)$.

Example 3.2. If

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 1 & 3 & 1 & 0 \end{pmatrix},$$

then the cycles of P are $(05), (142), (3)$ in this order, as the minimum elements of the cycles are $0, 1, 3$, respectively. Thus, $\mathrm{OCS}(P) = (2, 3, 1)$.

Sampling the ordered cycle structure of a random permutation in $P \in S_\Omega$ (by choosing a random P, finding the size of the cycle of 0, then the size of the cycle of the first element not in this cycle, etc.) requires $O(|\Omega|)$ steps, which is infeasible when Ω is a large space. The following theorem allows us to sample this distribution efficiently.

Theorem 3.3. *Let Ω be a finite set of size N. Consider the following two random processes:*

Process I: *Choose a random permutation $P \in S_\Omega$, and give $\mathrm{OCS}(P)$ as output.*

Process II:
1. *Set $s_{-1} = 0$.*
2. *For $i = 0, \dots$ do the following:*
   - (a) *Choose a random number $s_i \in \{1 + s_{i-1}, \dots, N\}$.*
   - (b) *If $s_i = N$, then exit the loop.*
3. *Output the sequence $(s_0, s_1 - s_0, s_2 - s_1 \dots, s_i - s_{i-1})$.*

Then these processes define the same distribution on the space of all possible ordered cycle structures of permutations $P \in S_\Omega$.

Proof. We prove the theorem by induction on the size of Ω. The theorem is evident when $|\Omega| = 1$.

For $|\Omega| > 1$, assume that P is a random element of S_Ω, and let $\mathrm{OCS}(P) = (a_0, \dots)$. By Lemma 2.2, a_0 distributes uniformly in $\{1, \dots, N\}$. Using the notation of Definition 3.1, let C_0 be the cycle of 0. As P distributes uniformly over S_Ω, an easy counting argument shows that the restriction of P to the remaining elements, $P \restriction \Omega \setminus C_0$ distributes uniformly over $S_{\Omega \setminus C_0}$. By the induction hypothesis, the output $(b_0, b_1, \dots)$ of Process II for $n = |\Omega \setminus C_0|$ distributes exactly as the output of Process I on $P \restriction \Omega \setminus C_0$. Thus, the sequence $(a_0, b_0, \dots)$ given by Process II distributes the same as the sequence given by Process I. $\square$

Definition 3.4. For ease of reference, we will call Process II of Theorem 3.3 the *Choose Cycle Lengths (CCL)* process.

Observe that the running time of the CCL process in the worse case is N, which is too large (usually, a quantity which is polynomial in $\log N$ is considered small, and $\Omega(N^{\epsilon})$ where $\epsilon > 0$ is considered infeasible). We can however define an algorithm which is probabilistically close to the CCL process but runs in time $O(\log N)$.

Let R_N denote the random variable counting the number of cycles in a permutation in S_N. It is well known [3] that the expectation and variance R_N (and therefore the running time of the CCL process) are both $\log N + O(1)$. By Chebyshev's Inequality,

$$\begin{aligned}
\Pr[R_N \geq (c+1)\log N] &= \Pr[R_N - \log N \geq c\log N] = \\
&= \Pr[R_N - \log N \geq (c\sqrt{\log N})\sqrt{\log N}] \leq \\
&\leq \frac{1}{(c\sqrt{\log N})^2} = \frac{1}{c^2\log N}
\end{aligned}$$

for all constant $c > 0$, which is $\Theta(1/\log N)$. We say that a function $f(N)$ is *negligible* if it is $O(1/N^{\epsilon})$ for some positive ϵ. The bound given by Chebyshev's Inequality is not negligible. Fortunately we can improve it significantly in our case. To this end, we need to have a tight upper bound on the distributions of the random variables s_i defined by the CCL process.

Proposition 3.5. *Fix $l \in \{0,\ldots,N-1\}$. Then*

$$\Pr[s_l = k] < \frac{\left|\log(1-\frac{k}{N})\right|^l}{l!N}$$

if $k \in \{l+1,\ldots,N\}$ and is 0 otherwise.

Proof. Recall that for an increasing function $f:[0,k]\to\mathbb{R}$, $\sum_{i=0}^{k-1} f(i) < \int_0^k f(x)dx$.

We prove the proposition by induction on l. For $l=0$ we have that $\Pr[s_0 = k] = 1/N$ as required. Assume that our assertion is true for l, and prove it for $l+1$ as follows.

$$\begin{aligned}
&\Pr[s_{l+1} = k] = \\
&= \sum_{i=l+1}^{k-1} \Pr[s_l = i]\cdot\Pr[a_{l+1} = k-i | s_l = i] = \sum_{i=l+1}^{k-1}\Pr[s_l = i]\cdot\frac{1}{N-i} < \\
&< \int_0^k \frac{\left(-\log(1-\frac{x}{N})\right)^l}{l!N}\cdot\frac{1}{N-x}dx
\end{aligned}$$

Substituting $t = -\log(1 - x/N)$, we have that the last integral is equal to

$$\frac{1}{l!N}\int_0^{-\log\left(1-\frac{k}{N}\right)} t^l dt = \frac{\left(-\log\left(1-\frac{k}{N}\right)\right)^{l+1}}{(l+1)!N}.$$

□

Theorem 3.6. *Fix $l \in \{0, \dots, N-1\}$. Then for all m,*

$$\Pr[s_l < m] < \frac{m}{N} \cdot \frac{\left|\log\left(1-\frac{m}{N}\right)\right|^l}{l!}.$$

Proof. By Proposition 3.5,

$$\begin{aligned}
&\Pr[s_l < m] < \\
&< \frac{1}{l!N}\int_0^m \left(-\log\left(1-\frac{x}{N}\right)\right)^l dx < \frac{1}{l!N}\int_0^m \left(-\log\left(1-\frac{m}{N}\right)\right)^l dx = \\
&= m \cdot \frac{\left|\log\left(1-\frac{m}{N}\right)\right|^l}{l!N}.
\end{aligned}$$

□

Corollary 3.7. *Assume that $c > e$. The probability that the running time of the CCL process is larger than $c \log N$ is $O\left(\sqrt{\log N}/N^{c(\log c-1)}\right)$ and is therefore negligible. In particular, if $c > e^2$ then this probability is $o(1/N^c)$.*

Proof. Use Theorem 3.6 with $m = n-1$ and $l = c\log N$. Then $1 - m/N = 1/N$. Using Stirling's Formula,

$$\Pr[s_l < m] < \frac{\left|\log\frac{1}{N}\right|^l}{l!} \approx \frac{\log^l N}{\sqrt{\frac{2\pi}{l}}\left(\frac{l}{e}\right)^l}. \tag{1}$$

Now, as $l = c\log N$,

$$\frac{\log^l N}{\left(\frac{l}{e}\right)^l} = \left(\frac{e\log N}{l}\right)^l = \frac{e^l}{c^l} = \frac{N^c}{N^{c\log c}} = N^{c(1-\log c)},$$

therefore the right hand side of Equation 1 is equal to

$$\sqrt{\frac{c\log N}{2\pi}} \cdot \frac{1}{N^{c(\log c-1)}}.$$

This implies the assertions in the theorem. □

We can therefore define the following variant of the CCL process:

Definition 3.8 (l-truncated CCL)**.** Fix a positive integer l and run the CCL process $l-1$ steps. If the process terminated after $k < l$ steps, then output the sequence $(s_0, \dots, s_{k-1})$. Otherwise set $s_{l-1} = N$ and output $(s_0, \dots, s_{l-1})$.

Corollary 3.9. *Fix $l \geq 3.6 \log N$. Then the output of the l-truncated CCL cannot be distinguished from the output of the CCL process with advantage greater than $o(1/N)$.*

Proof. This follows from Theorem 3.7, once we observe (numerically) that the solution to the equation $c(\log c - 1) = 1$ is $c = 3.5911^+$. □

4. Fast forward permutations

Definition 4.1. Assume that $(a_0, a_1, \dots, a_{l-1})$ is a sequence of positive integers such that $\sum_{k<l} a_k = N$, and write $s_0 = a_0$ and $s_i = \sum_{k\leq i} a_k$ for each $i = 1, \dots, l$. The *fast forward permutation coded by* $(a_0, a_1, \dots, a_{l-1})$ is the permutation $\pi \in S_N$ such that for each $x \in \{0, \dots, N-1\}$,

$$\pi(x) = s_i + (x + 1 \bmod a_{i+1}) \quad \text{where } s_i \leq x < s_{i+1}.$$

Example 4.2. The fast forward permutation $\pi \in S_7$ coded by $(1, 2, 3, 1)$ is

$$\pi = (0)(12)(345)(6) = (12)(345).$$

Here $s_0 = 1$, $s_1 = 3$, $s_2 = 6$, and $s_3 = 7$. Thus, e.g., as $s_1 \leq 4 < s_2$, we have that

$$\pi^5(4) = s_1 + (4 + 5 \bmod a_2) = 3 + (9 \bmod 3) = 3,$$

as can be verified directly.

A fast forward permutation coded by a sequence $(a_0, \dots, a_{l-1})$ is indeed fast forward, if we can either preprocess the corresponding sequence $(s_0, \dots, s_{l-1})$ (this is done in time $O(l)$) or have access to an oracle which can tell s_i for each i in time $O(1)$.

Proposition 4.3. *Assume that π is the fast forward permutation coded by $(a_0, \dots, a_{l-1})$. Assume further that we have an $O(1)$ time access to the corresponding values s_i, $i \in \{0, \dots, l-1\}$. Then for all $x \in \{0, \dots, N-1\}$ and all m, the complexity of the computation of $\pi^m(x)$ is $O(\log l)$ (and in particular $O(\log N)$).*

Proof. As the values s_i are increasing with i, we can use binary search to find the i such that $s_i \leq x < s_{i+1}$ (this requires $O(\log l)$ accesses to the values s_i). Then

$$\pi^m(x) = s_i + (x + m \bmod (s_{i+1} - s_i)).$$

□

The proof of Proposition 4.3 is written such that we can see that the sequence $(a_0, \dots, a_{l-1})$ plays no role in the evaluations of $\pi^m(x)$. This means that all needed information is given in the sequence $(s_0, \dots, s_{l-1})$. We chose the sequence $(a_0, \dots, a_{l-1})$ rather than $(s_0, \dots, s_{l-1})$ as a "code" for the permutation only because this way it seems more clear how the permutation π is computed.

Consider the following oracles.

$\mathcal{P}_{\mathrm{FF}}$: Chooses a random permutation $P \in S_N$, accepts queries of the form $(x, m) \in \{0, \dots, N-1\} \times \mathbb{Z}$, and responds with $y = P^m(x)$ for each such query.

$\mathcal{F}$: Runs the l-truncated CCL process with $l = 4\log N$ to obtain a sequence $(a_0, \dots, a_{l-1})$. (Let π denote the fast forward permutation coded by $(a_0, \dots, a_{l-1})$.) This oracle accepts queries of the form $(x, m) \in \{0, \dots, N-1\} \times \mathbb{Z}$, and uses the oracle $\mathcal{P}$ (which fixes a random permutation P) to respond with $y = P(\pi^m(P^{-1}(x)))$ for each such query.

Theorem 4.4. (1) *The space used by the oracle $\mathcal{F}$ is $O(\log N)$ words of size $O(\log N)$ each.*

(2) *The preprocess of $\mathcal{F}$ requires $O(\log N)$ steps.*
(3) *For each query (x, m), the running time of $\mathcal{F}$ is $O(\log\log N)$ plus twice the running time of $\mathcal{P}$.*
(4) *Assume that D is a distinguisher which makes any number of calls to the oracles $\mathcal{P}_{\mathrm{FF}}$ or $\mathcal{F}$. Then the advantage of D is $o(1/N)$.*

Proof. (1) is evident. (2) follows from Proposition 4.3, and (3) follows from Corollary 3.9. $\square$

This completes our solution to the Naor-Reingold Problem in the (purely) random case.

Part 3. Pseudorandomness

Intuitively speaking, pseudorandom objects are ones which are easy to sample but difficult to distinguish from (truly) random objects. The assumption that we made on the oracle $\mathcal{P}$—namely, that it chooses a random permutation in S_N—is not realistic when N is large. A more realistic assumption is that the oracle chooses a *pseudorandom* element of S_N. More concretely, the oracle $\mathcal{P}$ accepts a *key* k as input, and uses it to define a permutation P_k in the sense that each time the oracle is asked to compute $P_k(x)$ (or $P_k^{-1}(x)$), the oracle computes it without the need to explicitly build the complete permutation P_k. ($\mathcal{P}$ can be thought of as a key dependent block cipher.) The reader is referred to [1] for the formal definitions. Naor and Reingold [1] actually stated their problem in the pseudorandom case. We will translate our main results into the pseudorandom case.

5. Translation of results from Part 1

Let $\mathcal{C}'$ be a pseudorandom cyclus oracle. This means that for any distinguisher D which makes a small number m of queries, the advantage $a = |\Pr[D(\mathcal{C}') = 1] - \Pr[D(\mathcal{C}) = 1]|$ is small.

Theorem 5.1. *For any distinguisher D which makes $m < N$ queries to $\mathcal{C}'$ or $\mathcal{P}$,*

$$|\Pr[D(\mathcal{C}') = 1] - \Pr[D(\mathcal{P}) = 1]| \leq a + \frac{m}{N},$$

where $a = |\Pr[D(\mathcal{C}') = 1] - \Pr[D(\mathcal{C}) = 1]|$.

Proof. By the Triangle Inequality and Theorem 1.10,

$$\begin{aligned} &|\Pr[D(\mathcal{C}') = 1] - \Pr[D(\mathcal{P}) = 1]| \leq \\ &\leq \ |\Pr[D(\mathcal{C}') = 1] - \Pr[D(\mathcal{C}) = 1]| + |\Pr[D(\mathcal{C}) = 1] - \Pr[D(\mathcal{P}) = 1]| \leq \\ &\leq \ a + \frac{m}{n}. \end{aligned}$$

□

Theorem 5.2. *Consider the m-step strategy ($m < N$) for a distinguisher D which was defined in Theorem 1.12 (an arbitrary strategy which generates nonrepeating sequences.) Then*

$$|\Pr[D(\mathcal{C}') = 1] - \Pr[D(\mathcal{P}) = 1]| = \frac{m}{N}.$$

Consequently, for all $\epsilon > 0$ there exists a strategy D to distinguish $\mathcal{C}'$ from $\mathcal{P}$ with advantage $\max\{a - \epsilon, m/N\}$, where a is the supremum of all possible advantages of an m-step distinguisher to distinguish $\mathcal{C}'$ from $\mathcal{C}$.

Proof. The proof of Theorem 1.12 only uses the fact that $\mathcal{P}$ chooses a random permutation and $\mathcal{C}$ chooses a cyclus. The fact that the cyclus $\mathcal{C}$ is random is not used. This implies the first claim in our theorem.

To prove the second part of the theorem, fix any $\epsilon > 0$. If $a - \epsilon \leq m/N$, we choose the strategy D and we are done. Otherwise $m/N < a - \epsilon$. As $a - \epsilon < a$, there exists an m-step strategy D' to distinguish $\mathcal{C}'$ from $\mathcal{C}$ with advantage at least $a - \epsilon$, so we can choose the strategy D'. □

We now translate the main result in the fast forward model to the pseudorandom case.

Theorem 5.3. *$\mathcal{C}'$ can be distinguished from $\mathcal{P}$ with advantage $1 - d(N)/N$, using a single query.*

Proof. Again, the only property of $\mathcal{C}$ we used in the proof of Theorem 2.1 is its choosing a cyclus, which is also true for $\mathcal{C}'$. □

6. Translation of results from Part 2

In order to shift to the pseudorandom case in our construction of a fast forward permutation, we need to have some pseudorandom number generator to generate the random choices of the s_i's in the CCL process. If we have no such generator available, we can use the oracle $\mathcal{P}$ itself: In addition to the key k used to generate P_k, we need another key $\tilde{k}$. The pseudorandom numbers s_i in the CCL process can then be derived from the values $P_{\tilde{k}}(0)$, $P_{\tilde{k}}(1)$, $P_{\tilde{k}}(2), \dots$ (This is the standard *counter mode* [2]). We now give an example how this can be done.

Consider the following oracles.

RND: Accepts positive integers $x, k < N$ and returns a sequence $(r_0, \dots, r_{k-1})$ of random numbers in the range $\{0, \dots, x-1\}$.

RND_1: Accepts positive integers $x, k < N$, calls RND with N and $2k$ to get a sequence $(x_0, \dots, x_{2k-1})$, and returns $(r_0, \dots, r_{k-1})$ where $r_i = (x_{2i} + N \cdot x_{2i+1}) \bmod x$ for all $i = 0, \dots, k-1$.

RND_2: Accepts positive integers $x, k, p_0 < N$, calls $\mathcal{P}$ $2k$ times to obtain the sequence $(x_0 = P(p_0), \dots, x_{2k-1} = P(p_0 + 2k - 1 \bmod N))$, and returns $(r_0, \dots, r_{k-1})$ where $r_i = (x_{2i} + N \cdot x_{2i+1}) \bmod x$ for all $i = 0, \dots, k-1$.

Theorem 6.1. *Fix positive integers $x, k < N$. Then:*

1. *If $k = c \log N$, then* RND *and* RND_1 *called with x and k cannot be distinguished with advantage greater than $c \log N/N$.*
2. RND_1 *and* RND_2 *called with x and k cannot be distinguished with advantage greater than $2k^2/N$.*

Proof. (1) Assume that a and b are random numbers in the range $\{0, \dots, N-1\}$. Then $c = a+bN$ is random in the range $\{0, \dots, N^2-1\}$. Let $x \in \{0, \dots, N-1\}$. With probability at least $1/N$, $c < \lfloor N^2/x \rfloor \cdot x$ and therefore $c \bmod x$ is random in the range $\{0, \dots, x-1\}$. The probability that this happens $c \log N$ times is therefore at least $(1 - 1/N)^{c \log N} \approx e^{-c \log N/N} > 1 - c \log N/N$.

(2) This follows from the well known result that a random permutation is a pseudorandom function. Briefly (see [4] for more details), consider any sequence of $2k$ random numbers in the range $\{0, \dots, N-1\}$. The probability that all these numbers are distinct is greater than

$1 - (2k)^2/2N = 1 - 2k^2/N$, and in this case this sequence forms a random partial permutation. □

Consider now the modification $\mathcal{F}'$ of the oracle $\mathcal{F}$ which calls $\mathcal{P}$ with two independent keys k and $\tilde{k}$, one for the evaluations $P_k(\pi^m(P_k^{-1}(x)))$ and the other for the values $P_{\tilde{k}}(0), P_{\tilde{k}}(1), \dots$ to be used by RND_2 in order to generate the sequence of pseudorandom numbers required by the l-truncated CCL process (the input argument p_0 to RND_2 is used to avoid sampling the same entry of $P_{\tilde{k}}$ twice).

Theorem 6.2. *$\mathcal{F}'$ and $\mathcal{F}$ cannot be distinguished with advantage greater than $O(\log^2 N/N)$.*

Proof. This follows from the Triangle Inequality and the earlier results 4.4, 6.1(1), and 6.1(1) with $k = 4\log N$. □

Here too, using a pseudorandom permutation oracle $\mathcal{P}'$ instead of a random one in the definition of $\mathcal{F}'$ cannot increase the advantage by more than a where a is the maximal advantage obtainable in distinguishing $\mathcal{P}$ from $\mathcal{P}'$.

7. Final remarks and open problems

Another problem is mentioned in the original paper of Naor and Reingold [1] and remains open, namely, whether one can construct a family of fast forward pseudorandom *functions* with graph structure distribution similar to that of pseudorandom functions.

The natural analogue of our construction for the case of pseudorandom permutations would not work for pseudorandom functions, simply because the "graph structure" of a pseudorandom function carries too much information. For example, there are $O(N)$ points with no preimage. This was not the case with permutations, where the structure is determined by the logarithmic number of its cycles and their length. Another approach will be needed in order to solve this problem.

Our study raises some other interesting open problems, the most interesting of which seems to be the following. Consider the l-truncated CCL process with $l = \log N$, which uses an oracle RND_3 similar to RND_2 as its random number generator with the difference that it makes only k calls to $\mathcal{P}$ to generate $(x_0 = P(p_0), \dots, x_{k-1} = P(p_0 + k - 1 \bmod N))$, and uses $r_i = x_i \bmod x$ instead of the original definition. (So we use $\log N$ values of P instead of $8\log N$ in the current construction.) The problem is to prove or disprove the following.

Conjecture 1. *$\mathcal{F}'$ with the parameters just described cannot be distinguished from $\mathcal{P}_{\mathrm{FF}}$ with a non-negligible advantage.*

8. Acknowledgments

I thank Kent E. Morrison for reference [3], and Uzi Vishne for reading the paper and suggesting (the first paragraph of) Remark 2.7. A special thanks is owed to Moni Naor for encouraging me to publish these results, and to the referee for suggesting important improvements in the presentation of the paper.

References

[1] Moni Naor and Omer Reingold, *Constructing Pseudo-Random Permutations with a Prescribed Structure*, Journal of Cryptology **15** (2002), 97–102.
[2] Bruce Schneier, *Applied Cryptography*, John Wiley and Sons, 1996.
[3] L. A. Shepp and S. P. Lloyd, *Ordered cycle lengths in a random permutation*, Trans. Amer. Math. Soc. **121** (1966), 340–357.
[4] Boaz Tsaban, *Bernoulli numbers and the probability of a birthday surprise*, Discrete Applied Mathematics, to appear. `http://arxiv.org/abs/math.NA/0304028`

Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
E-mail address: `tsaban@macs.biu.ac.il`, `http://www.cs.biu.ac.il/~tsaban`