Computing the Cycles in the Perfect Shuffle Permutation *

John Ellis Tobias Krahn Hongbing Fan

Department of Computer Science

University of Victoria, P.O. Box 3055

Victoria, British Columbia, V8W 3P6, Canada

May 19, 2000

Keywords: algorithms, perfect shuffle, permutation, in-place

1 Introduction

We define the *perfect shuffle permutation* of order n to be the permutation:

$$\rho_n : i \rightarrow 2i \bmod (n+1),\ i \in \{1, 2, \cdots, n\}$$

where n is an even number. For example, the perfect shuffle on six elements, $\rho_6 = (1\ 2\ 4)(3\ 6\ 5)$, maps the list $abcdef$ to the list $daebfc$, The name is taken from that method of shuffling a deck of cards that cuts the deck into halves and then interleaves the two halves perfectly. In that context, our definition is called an *in* shuffle. There is also an *out* shuffle which leaves the first and last elements of the list unmoved.

The perfect shuffle has many interesting mathematical properties and applications in computer science. The group structure of the perfect shuffle and some applications to network design are given in [DGK83] and [MM87]. A family of parallel computer architectures and associated algorithms are based on the perfect shuffle. See for example [Sto71], [Bat91] and [Lei92]. In [EM00] it is shown that the classic problem of merging lists in-place, with stability, can be reduced to the problem of accomplishing the perfect shuffle in-place. Hence an efficient realisation of this permutation would permit efficient simulation of parallel algorithms on sequential machines, and may open up new merging methods.

We have in mind permuting, in-place, a list represented by a one dimensional array of elements indexed by the integers 1 through n. By in-place we mean without the use of substantial extra space over and above that which the list elements already occupy. To be

*This work was supported by the Natural Sciences and Engineering Research Council of Canada

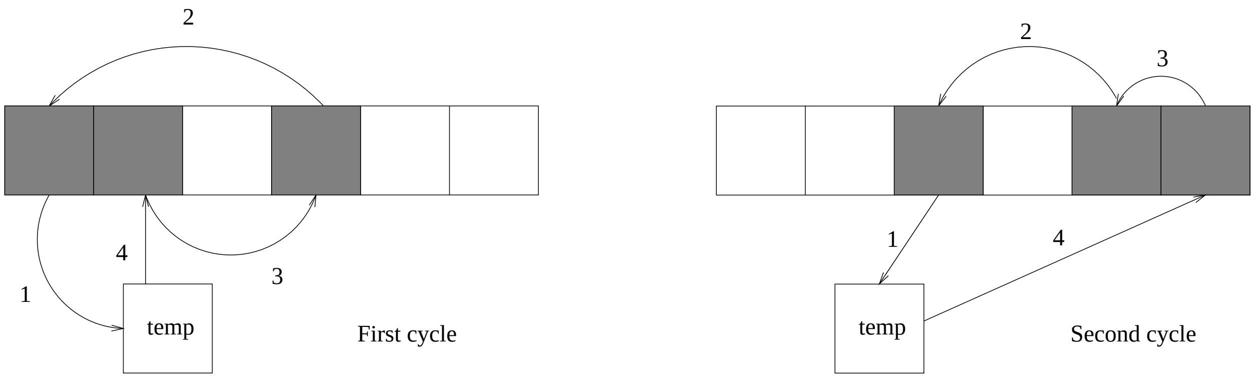

Figure 1: Realising the Perfect Shuffle

precise, we allow ourselves no more than $O(\log^2 n)$ extra bits for program variables and data structures. This definition was originally proposed by Knuth [Knu73], (Chapter 5, Section 5, Exercise 3). The intention was to permit some fixed number of program variables and also to allow recursion.

Permutations are made up of disjoint cycles and it is easy to move all the elements of one cycle, using just one extra location, by a so called "cycle leader" algorithm, [FMP95]. The method proceeds by repeatedly making a space in the list, computing the index of the element that belongs in that space and moving that element, so creating a new space. For example, to permute $\rho_6 = (1\ 2\ 4)(3\ 6\ 5)$, we can move the elements as indicated in Figure 1. In that figure, the numbers on the arrows define the order in which the moves take place.

If we can easily find a place to start a new cycle when the current cycle terminates then the entire task becomes easy. This is the case for some commonly used permutations such as reversal and cyclic shift. In both of those cases, if the current cycle was started at location i and elements remain to be moved at the end of the current cycle, then it is easy to show that the element at location $i+1$ has yet to be moved. The problem with the perfect shuffle is that it is by no means apparent how to compute the beginning of a new cycle when needed. A method already presented in [EM00] achieves the perfect shuffle in-place and in linear time at the expense of using about twice the number of element moves used by the method described in this paper. This is done by reducing the general case to the special case of permutations of length a power of 2. In that special case the problem reduces to computing representative elements of so called "necklaces", for which linear time solutions are already known.

In this paper we investigate an alternative approach, namely computing the position of representative elements of the cycles in the general case. We analyse the structure of the permutation in terms of the size, number and location of its cycles. Then we construct a procedure that computes a set of representative elements, in-place and in linear time. A cycle leader algorithm can use this set to realise the perfect shuffle in-place and in linear time.

We call the set of representative elements a set of *seeds*. The seed set is a set of array indices with the following properties:

1. No two seeds are in the same cycle.
2. Every cycle contains a seed.

We show how to compute a seed set using $O(\log^2 n)$ bits and in linear time.

2 The Cycle Structure and Algorithm

We use abstract algebra to analyse the cycle structure of the perfect shuffle. We follow the terminology and symbols used in, for example, [BS96] and we use the following number theoretic concepts:

- Let m be a positive integer. The *Euler Function* $\varphi(m)$ is defined to be the number of numbers $1, 2, \cdots, m$ that are relatively prime to m.
- Let a be a positive integer. If $\gcd(a, m) = 1$, the least natural number t such that $a^t \equiv 1 \pmod{m}$ is called the *order* of $a \pmod{m}$.
- a is called a *primitive root of* m if the order of a is $\varphi(m)$.
- Let g be a primitive root of m. The numbers g^w with $w = 0, 1, \cdots, \varphi(m) - 1$ form a reduced set of residues $\pmod{m}$; hence for every integer k with $\gcd(k, m) = 1$ there is a unique w such that $g^w \equiv k \pmod{m}$.
- The exponent w is called the *index* (or discrete logarithm) *of* k *with respect to* g (mod m) and is denoted by $\mathrm{ind}_g k \pmod{m}$, or just $\mathrm{ind}_g k$.

Let C be a cycle of ρ_n and a be an element of C. By the definition of ρ_n, $C = (a, 2a, 2^2 a, \ldots, 2^r a)$ where r is the least positive integer such that $2^{r+1} a \equiv a \pmod{n+1}$. Let $m = n + 1, g = \gcd(a, m)$ and $d = \frac{m}{g}$. Then $d \neq 1$ and $2^{r+1} \frac{a}{g} \equiv \frac{a}{g} \pmod{d}$ and $\gcd(2, d) = \gcd(\frac{a}{g}, d) = 1$. This implies that $2, \frac{a}{g} \in (Z/(d))^*$ and $\{1, 2, \ldots, 2^r\} = \langle 2 \rangle_d$ is a subgroup of $(Z/(d))^*$ generated by 2, where $(Z/(d))^*$ denotes the group formed from the units of the ring $Z/(d)$ under the operation of multiplication. Therefore, $\{\frac{a}{g}, 2\frac{a}{g}, \ldots, 2^r \frac{a}{g}\} = \{1, 2, \ldots, 2^r\} \frac{a}{g}$ is a coset of $\langle 2 \rangle_d$ in $(Z/(d))^*$, and C is formed from $\frac{m}{d}$ times a coset of $\langle 2 \rangle_d$ in $(Z/(d))^*$.

By a *non-trivial* divisor d of m, we mean that $d | m$ and $d \neq 1$. For convenience, we will use "divisor" to mean "non-trivial divisor". For such a divisor d, let $\langle 2 \rangle_d$ be the subgroup of $(Z/(d))^*$ generated by 2 and $|\langle 2 \rangle_d| = r$ and $a \in (Z/(d))^*$. Then $\gcd(a, d) = 1$, and r is the least positive integer such that $2^{r+1} \equiv 1 \pmod{d}$ and $2^{r+1} a \equiv a \pmod{d}$. Hence $2^{r+1} \frac{am}{d} \equiv \frac{am}{d} \pmod{d\frac{m}{d}} \equiv \frac{am}{d} \pmod{m}$. Therefore, $(\frac{am}{d}, 2\frac{am}{d}, \ldots, 2^r \frac{am}{d})$ is a cycle of ρ_n.

Hence each cycle of the perfect shuffle permutation ρ_n corresponds to a divisor d of $n+1$ and an element of the group quotient $(Z/(d))^* / \langle 2 \rangle_d$. This cycle structure implies that the seed set can be decomposed into the union of sets of coset representatives times $(n+1)/d$ for all divisors d of $n+1$. This leads to the following method of computing a seed set.

First, compute the prime factorisation of $n+1$. Then all divisors of $n+1$ and their prime factorisations can be obtained easily. Let $q_1^{u_1}q_2^{u_2}\cdots q_s^{u_s}$ be the prime factorisation of a divisor d of $n+1$. Then each q_i must be an odd prime because $n+1$ is odd. Then $Z/(d) = Z/(q_1^{u_1}q_2^{u_2}\cdots q_s^{u_s}) \cong Z/(q_1^{u_1}) \oplus Z/(q_2^{u_2}) \oplus \cdots \oplus Z/(q_s^{u_s})$ with isomorphism $f=(f_1, f_2, \ldots, f_s)$, $f_i(x) = x \bmod p_i^{u_i}$ (Theorem 5.5.4 [BS96]) and $f(2) = (2, 2, \ldots, 2)$. Furthermore, the restriction of f on $(Z/(d))^*$ is an isomorphism from $(Z/(d))^*$ to $(Z/(q_1^{u_1}) \oplus Z/(q_2^{u_2}) \oplus \cdots \oplus Z/(q_s^{u_s}))^* = (Z/(q_1^{u_1}))^* \times (Z/(q_2^{u_2}))^* \times \cdots \times (Z/(q_s^{u_s}))^*$ (Lemma 5.6.1[BS96]).

Second, compute the primitive root g_i of $q_i^{u_i}$ and let $w_i = \mathrm{ind}_{g_i}2 \pmod{p_i^{u_i}}$. Then $\langle 2\rangle_{q_i^{u_i}} = \langle g_i^{w_i}\rangle$ where $\langle 2\rangle_{q_i^{u_i}}$ is the subgroup of $(Z/(q_i^{u_i}))^*$ generated by 2. Therefore, $(Z/(d))^*/\langle 2\rangle_d \cong (Z/(q_1^{u_1}))^* \times (Z/(q_2^{u_2}))^* \times \cdots \times (Z/(q_s^{u_s}))^*/\langle (g_1^{w_1}, g_2^{w_2}, \ldots, g_s^{w_s})\rangle$.

Third, compute the minimum (lexicographically) integer vector $(c_1, c_2, \ldots, c_s)$ such that $(g_1^{c_1}, g_2^{c_2}, \ldots, g_s^{c_s})$ is an element of $\langle (g_1^{w_1}, g_2^{w_2}, \ldots, g_s^{w_s})\rangle$. Then each $(k_1, k_2, \ldots, k_s), 0 \le k_i < c_i$ corresponds to a coset representative $(g_1^{k_1}, g_2^{k_2}, \ldots, g_s^{k_s})$ of $\langle (g_1^{w_1}, g_2^{w_2}, \ldots, g_s^{w_s})\rangle$ in $(Z/(q_1^{u_1}))^* \times (Z/(q_2^{u_2}))^* \times \cdots \times (Z/(q_s^{u_s}))^*$, and *vice versa*.

Fourth, compute $h = f^{-1}(g_1^{k_1}, g_2^{k_2}, \ldots, g_s^{k_s})$. Then h corresponds to a coset representative of $\langle 2\rangle_d$ in $(Z/(d))^*$, and hence $\frac{h(n+1)}{d}$ is a seed.

The Seed Generator

Step 1 Let $m = n+1$.

Step 2 Compute the prime factorisation of m, say $p_1^{e_1}p_2^{e_2}\cdots p_r^{e_r}$.

Step 3 For each prime factor p_i, compute a primitive root of p_i and call it $g_{i,1}$. If $e_i \ge 2$, compute a primitive root of p_i^2 and call it $g_{i,2}$.

Step 4 For each prime factor p_i, compute $\mathrm{ind}_{g_{i,1}}2 \pmod{p_i}$ and call it $w_{i,1}$. If $e_i \ge 2$ compute $\mathrm{ind}_{g_{i,2}}2 \pmod{p_i^2}$ and call it $w_{i,2}$.

Step 5 Using the prime factorisation of m, systematically compute each divisor d of m and its prime factorisation, $q_1^{u_1}q_2^{u_2}\cdots q_s^{u_s}$. As a divisor d is generated, carry out steps 5.1 to 5.3.

Step 5.1 For each prime factor q_i of d, let k be such that $q_i = p_k$.
If $u_i = 1$ then let $g_i = g_{k,1}$ otherwise let $g_i = g_{k,2}$. If $u_i = 1$ then let $w_i = w_{k,1}$ else if $u_i = 2$ then let $w_i = w_{k,2}$ otherwise let $w_i = \mathrm{ind}_{g_i}2 \pmod{q_i^{u_i}}$, which must be computed.

Step 5.2 Compute c_i for $i = 1, 2, \cdots, s$ by

$$\begin{cases} t_i = \varphi(q_i^{u_i}), \\ a_i = \gcd(w_i, t_i), \\ b_0 = 1, b_i = t_i/a_i, \\ c_i = a_i \gcd(\mathrm{lcm}(b_1, b_2, \ldots, b_{i-1}), b_i) \end{cases} \tag{1}$$

Step 5.3 For every integer vector $(k_1, k_2, \ldots, k_s)$ with $0 \leq k_i < c_i$, solve

$$\left\{ \begin{array}{rcl} h & \equiv & g_1^{k_1} \pmod{q_1^{u_1}} \\ h & \equiv & g_2^{k_2} \pmod{q_2^{u_2}} \\ & \vdots & \\ h & \equiv & g_s^{k_s} \pmod{q_s^{u_s}} \end{array} \right. \tag{2}$$

to obtain a solution h in $\{1, 2, \ldots, n\}$ and output $\frac{hm}{d}$, which is a seed.

3 Proof of Correctness

We will use some well known theorems and elementary techniques of number theory to prove the correctness of the algorithm as well as to justify the claimed bounds on time and space requirements in Section 4. Most are standard [Jon64, Agn72]. For those that are less common we cite a reference. For convenience and consistency we define lcm(), where () denotes the empty list, to be one.

Theorem 1 *Let a, b, c be positive integers then the following equations hold:*

(1) $\operatorname{lcm}(a, b) \gcd(a, b) = ab$.

(2) $\gcd(a, bc) = \gcd(a, \gcd(a, b) \gcd(a, c))$.

(3) $\operatorname{lcm}(\operatorname{lcm}(a, b), c) = \operatorname{lcm}(a, b, c)$ *and* $\gcd(\gcd(a, b), c) = \gcd(a, b, c)$.

Theorem 2 *The following statements about congruence hold:*

(1) $ax \equiv b \pmod{m}$ *has a solution for some integer x if and only if* $\gcd(a, m) | b$.

(2) (The Chinese Remainder Theorem) *The system of congruence equations*

$$x \equiv a_i \pmod{m_i}, i = 1, 2, \ldots, k$$

where $\gcd(m_i, m_j) = 1$ *for all $i \neq j$ has a unique solution modulo M given by*

$$x \equiv a_1 \left(\frac{M}{m_1}\right)^{\varphi(m_1)} + a_2 \left(\frac{M}{m_2}\right)^{\varphi(m_2)} + \cdots + a_k \left(\frac{M}{m_k}\right)^{\varphi(m_k)} \tag{3}$$

where $M = m_1 m_2 \cdots m_k$.

(3) *Let the prime factorisation of m be $p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$. The solution set of the congruence $f(x) \equiv 0 \pmod{m}$ is equal to the solution set of the system of simultaneous congruence equations $f(x) \equiv 0 \pmod{p_i^{e_i}}, i = 1, 2, \ldots, r$*

Theorem 3 *Let a, m be positive integers. The following statements hold:*

(1) *If the prime factorisation of* m *is* $p_1{}^{e_1} p_2{}^{e_2} \cdots p_r{}^{e_r}$ *then* $\varphi(m) = (p_1 - 1)p_1{}^{e_1-1}(p_2 - 1)p_2{}^{e_2-1} \ldots (p_r - 1)p_r{}^{e_r-1}$.

(2) (The Euler-Fermat Theorem) *If* $\gcd(a, m) = 1$ *then* $a^{\varphi(m)} \equiv 1 \pmod{m}$.

(3) $\sum_{d|m} \varphi(d) = m$, *where the sum is over all divisors of* m *including* 1.

(4) *If* $\gcd(a, m) = 1$ *then* $\varphi(am) = \varphi(a)\varphi(m)$.

(5) *If the order of* $a \pmod{m}$ *is* t *and* $a^k \equiv 1 \pmod{m}$, *then* $t|k$.

(6) *If* g *is a primitive root of* m *and* $g^t \equiv a \pmod{m}$ *then* a *has order* $\frac{\varphi(m)}{\gcd(t,\varphi(m))}$.

Theorem 4 *The following statements about primitive roots and indices hold:*

(1) *There exists a primitive root of* m *if and only if* $m = 2, 4, p^a$, *or* $2p^a$ *where* p *is an odd prime and* a *is a positive integer.*

(2) g *is a primitive root of* m *iff* $g^{(m-1)/q} \not\equiv 1 \pmod{m}$ *for all primes* q *that divide* $m - 1$ [Knu81, page 348].

(3) *Each odd prime number* m *has* $\varphi(\varphi(m))$ *primitive roots.*

(4) *Let* p *be an odd prime. If* g *is a primitive root of* p^2 *then* g *is a primitive root of* p^r *for all* $r \geq 2$ [Jon64, page 114] .

(5) *Let* g *be a primitive root of an odd prime* p. *If* $g^{p-1} \equiv 1 \pmod{p^2}$ *then* $g + p$ *is a primitive root of* p^2, *else* g *is a primitive root of* p^2 [Agn72, page 124].

(6) *Let* g *be a primitive root of* m. *Then* $\{g, g^2, \ldots, g^{\varphi(m)}\}$ *is a reduced residue system* $\pmod{m}$ *and each member of which is relatively prime to* m *and no two of which are congruent* $\pmod{m}$.

(7) *If* g *is a primitive root of* m *and* $a \equiv b^n \pmod{m}$ *then* $\mathrm{ind}_g a \equiv n\ \mathrm{ind}_g b \pmod{\varphi(m)}$.

(8) *If* g *is a primitive root of both* p^n *and* p^{n+1} *and if* $i_n = \mathrm{ind}_g 2 \pmod{p^n}$ *and* $i_{n+1} = \mathrm{ind}_g 2 \pmod{p^{n+1}}$, *then* $i_{n+1} = i_n + k\varphi(p^n)$ *where* $0 \leq k \leq p$.

We now demonstrate the correctness of the seed generating algorithm. We first note that, since m is an odd number, all prime factors of m must be odd primes and they and their powers must have primitive roots by Theorem 4-(1).

By Theorem 4-(4) when $u_i \geq 3$ the primitive root $g_{i,2}$ of q_i^2 is also a primitive root of $q_i^{u_i}$, hence the $g_i = g_{i,2}$ is a primitive root of $q_i^{u_i}$ and $w_i = \mathrm{ind}_{g_i} 2 \pmod{q_i^{u_i}}$ is well defined. This implies that g_i and w_i and a_i, b_i and c_i $(i = 1, 2, \ldots, s)$ in Step 5.1 and 5.2 are all well defined. Clearly $c_i \geq 1$, so that $\{(k_1, k_2, \ldots, k_s) \mid 0 \leq k_i < c_i\}$ is not empty. By Theorem 2-(2) the system of equations (2) always has a unique solution within $\{1, 2, \ldots, n\}$. Therefore the seed generator will always generate at least one seed in $\{1, 2, \ldots, n\}$.

Let S be the entire set of seeds generated. It is necessary and sufficient to prove that S has the following properties:

Property 1 *No two elements in S are contained in the same cycle of ρ_n.*

Property 2 *$|S|$ is equal to the number of cycles in ρ_n.*

Proof of Property 1 We proceed in two steps. Firstly we show that any two elements generated in different executions of Step 5.3 are in different cycles. Secondly we show that any two elements generated in a particular execution of Step 5.3 are in different cycles.

Proposition 1 *If i and j are members of the same cycle then* $\gcd(i, m) = \gcd(j, m)$.

Proof Let $f = \gcd(i, m)$ and $f' = \gcd(j, m)$ and, without loss of generality, let $f \leq f'$. Since i and j are in the same cycle, $i \equiv 2^k j \pmod{m}$ for some integer k. Thus $m | i - 2^k j$. By definition, $f'|j$ and $f'|m$ and so $f'|i$. But f' can not be greater than f, which is the greatest common divisor of m and i. Hence $f' = f$.

□

Proposition 2 *The elements of S generated by different executions of Step 5.3 are not in the same cycle.*

Proof Let $a = \frac{hm}{d} \in S$ come from an iteration of Step 5.3 where h is the solution of equations (2). Then $\gcd(h, q_i^{u_i}) | g_i^{k_i}$ by Theorem 2-(1) and $\gcd(g_i^{k_i}, q_i^{u_i}) = 1$ by Theorem 4-(6). Therefore $\gcd(h, q_i^{u_i}) = 1$ for all prime factors q_i of d and hence $\gcd(h, d) = 1$. Thus $\gcd(a, m) = \gcd(\frac{hm}{d}, \frac{md}{d}) = \frac{m}{d}$. Let $a' = \frac{h'm}{d'} \in S$ come from another iteration of Step 5.3, when $d' \neq d$. Similarly, we have $\gcd(a', m) = \frac{m}{d'}$. Since $d \neq d'$ we know, by Proposition 1, that a and a' are not in the same cycle. □

Proposition 3 *Elements of S generated within one execution of Step 5.3 are not in the same cycle.*

Proof Let $\frac{hm}{d}, \frac{h'm}{d} \in S$ be two elements generated within one execution of Step 5.3 and h and h' correspond to $(k_1, k_2, \ldots, k_s)$ and $(k'_1, k'_2, \ldots, k'_s)$, respectively, where $0 \leq k_i, k'_i < c_i$, $(k_1, k_2, \ldots, k_s) \neq (k'_1, k'_2, \ldots, k'_s)$. Let l be the smallest integer such that $k_l \neq k'{}_l$. Without loss of generality, suppose $k_l > k'_l$.

Suppose, contrary to the claim of the proposition, that $\frac{hm}{d}$ and $\frac{h'm}{d}$ are members of the same cycle. Then $\frac{hm}{d} \equiv \frac{h'm}{d} 2^z \pmod{m}$ and hence $h \equiv h' \, 2^z \pmod{d}$ for some integer z. Since $q_i^{u_i}|d$, $h \equiv h' 2^z \pmod{q_i^{u_i}}$. Then, by Theorem 4-(7), we have $\text{ind}_{g_i} h \equiv \text{ind}_{g_i} h' + z w_i \pmod{t_i}$. Since $h \equiv g_i^{k_i} \pmod{q_i^{u_i}}$, by Theorem 4-(7), $\text{ind}_{g_i} h \equiv k_i \pmod{t_i}$ and similarly $\text{ind}_{g_i} h' \equiv k'_i \pmod{t_i}$. Thus we derive

$$zw_i \equiv k_i - k'_i \pmod{t_i} \tag{4}$$

By the choice of l, $zw_i \equiv 0 \pmod{t_i}$ for $i = 1, 2, \ldots, l-1$. and $zw_l \not\equiv 0 \pmod{t_l}$. Dividing both sides by a_i, we derive $b_i | (zw_i/a_i)$, since $t_i | zw_i$ for all $i = 1, 2, \ldots, l-1$. Since $\gcd(w_i/a_i, b_i) = 1$, $b_i | z$ for $i = 1, 2, \ldots, l-1$.

Let $y = \mathrm{lcm}(b_1, b_2, \ldots, b_{l-1})$. Then $y|z$, so $z = vy$ for some integer v. When $i = l$, Equation (4) reads

$$yw_l v \equiv k_l - k_l' \pmod{t_l}$$

By Theorem 2-(1) we know that $\gcd(yw_l, t_l)|k_l - k'_l$. By Theorem 1-(2, 3) we derive

$$\begin{aligned}
\gcd(yw_l, t_l) &= \gcd(\gcd(y, t_l)\gcd(w_l, t_l), t_l) \\
&= \gcd(\gcd(y, t_l)a_l, t_l) \\
&= a_l \gcd(\gcd(y, t_l), b_l) \\
&= a_l \gcd(y, \gcd(t_l, b_l)) \\
&= a_l \gcd(y, b_l) \\
&= c_l
\end{aligned}$$

so that $c_l|k_l - k_l'$. But this is impossible since $0 < k_l - k_l' < c_l$. Thus $\frac{hm}{d}$ and $\frac{h'm}{d}$ are not in the same cycle. □

Property 1 follows immediately from Propositions 2 and 3. □

Proof of Property 2 Clearly each divisor d of m corresponds to a set of cycles. The union of all these disjoint cycles contain a total of $\varphi(d)$ integers. These $\varphi(d)$ integers are broken up into $\frac{\varphi(d)}{r_d}$ disjoint cycles each of length r_d, where r_d is the order of 2 (mod d). Thus the total number of cycles is $\sum_{d|m, d\neq 1} \frac{\varphi(d)}{r_d}$. This formula has been obtained previously [All84]. We only need to show that

$$|S| = \sum_{d|m, d\neq 1} \frac{\varphi(d)}{r_d} \tag{5}$$

By Proposition 3, for each divisor d of m, the execution of steps 5.1 to 5.3 generates a subset $S_d \subset S$. Each $(k_1, k_2, \ldots, k_s) \in \{(k_1, k_2, \ldots, k_s)|0 \leq k_i < c_i\}$ corresponds to an element in S. Hence $|S_d| = c_1 c_2 \cdots c_s$. Since $g_i^{w_i} \equiv 2 \pmod{q_i^{u_i}}$, 2 has order $\varphi(q_i^{u_i})/\gcd(w_i, \varphi(q_i^{u_i})) = t_i/a_i = b_i$ by Theorem 3-(6). Then the least positive integer x such that $2^x - 1 \equiv 0 \pmod{q_i^{u_i}}$, $i = 1, 2, \ldots, s$ must be $\mathrm{lcm}(b_1, b_2, \ldots, b_s)$ by Theorem 3 - (5). By Theorem 2-(3), $\mathrm{lcm}(b_1, b_2, \ldots, b_s)$ is the least positive integer x such that $2^x - 1 \equiv 0 \pmod{d}$ and hence the order of 2 (mod d). That is, $r_d = \mathrm{lcm}(b_1, b_2, \ldots, b_s)$. We now observe that

$$\begin{aligned}
c_1 c_2 \cdots c_s &= a_1 \gcd(\mathrm{lcm}(b_0), b_1) a_2 \gcd(\mathrm{lcm}(b_1), b_2) \cdots a_s \gcd(\mathrm{lcm}(b_1, b_2, \ldots, b_{s-1}), b_s) \\
&= (a_1 a_2 \cdots a_s) \gcd(\mathrm{lcm}(b_1), b_2) \cdots \gcd(\mathrm{lcm}(b_1, b_2, \ldots, b_{s-1}), b_s) \\
&= \frac{(a_1 a_2 \cdots a_s)(b_1 b_2 \cdots b_s)}{\mathrm{lcm}(b_1, b_2, \ldots, b_s)} \\
&= \frac{t_1 t_2 \cdots t_s}{\mathrm{lcm}(b_1, b_2, \ldots, b_s)} \\
&= \frac{\varphi(d)}{r_d}
\end{aligned}$$

where the third equality is obtained by repeated use of Theorem 1-(1, 3) and the fifth equality is obtained from Theorem 3-(1, 4). Hence $|S_d| = \frac{\varphi(d)}{r_d}$. By Proposition 2, $S_d \cap S_{d'} = \emptyset$ for different divisors d and d', so that $|S| = \sum_{d|m} |S_d|$ follows from equation (5). □

4 Space and Time Complexity

We want to show that seed generation can be accomplished within linear time and in-place. If so, then the corresponding algorithm for realising the perfect shuffle, as outlined in the introduction, would also be in-place and linear time.

For each prime factor of m, the algorithm requires space to store the factor and the power to which it is raised and the corresponding two primitive roots and two indices. Step 3 and the solution of equation (2) in Step 5.3 also requires space proportional to the number of prime factors. Since the number of distinct prime factors of an integer i is no greater than $\log_2 i$, the number of bits used by the algorithm is bounded by $O(\log^2 n)$.

Our measure of time complexity is a count of arithmetic operations on integers n represented by bit strings of length $O(\log n)$. The operations include multiplication, division and modulo, and hence the model is not realistic for integers of unlimited size. Nevertheless, it is reasonable for the application we have in mind, namely for integers that fit into a computer word. We examine each step of the algorithm.

Step 1 is trivial requiring constant time.

Step 2 the prime factorisation of m, can be accomplished by elementary methods in $O(\sqrt{n})$ time and there are faster algorithms. See for example [Sha69].

Step 3 computes primitive roots for all p_i's and p_i^2's. We show that this can be done in $O(n)$ time by applying some known results regarding the distribution of primitive roots.

If m has primitive roots, we can use Theorem 4-(2) to test a candidate primitive root q of m. It is necessary to compute the prime factors of $m-1$, requiring time $O(\sqrt{m})$, or better as in the above cited reference. These factors may be stored in $O(\log^2 m)$ space since there can be no more than $\log_2 m$ of them. For each factor q we then compute $g^{(m-1)/q} \pmod{m}$, which can be done in $O(\log m)$ time. Hence q can be tested in $O(m^{1/2} \log m)$ time.

Let $g(p)$ denote the least positive primitive root of p. In [Wan59, Bur62] we are given that $g(p) \leq p^{1/4+o(1)}$. Consequently, we can test a sequence of candidate roots of m, starting at 2, and be guaranteed to find a root in time $O(m)$.

A randomised search for a primitive root is also feasible because there are in fact "many" primitive roots. In [RS62] we are given a lower bound of $\Omega(\frac{n}{\log \log n})$ on $\varphi(n)$. Since $\varphi(n) = n - 1$ when n is prime, by Theorem 4-(3) we have a lower bound of $\Omega(\frac{n}{\log \log n})$ on the number of primitive roots. Thus an algorithm which picks candidate roots at random has a chance of success of $1/(c \log\log n)$ for come constant c. Hence the expected number of tests necessary to find a root with some fixed probability slowly increases with n. For a particular prime p_i each test requires $O(p_i^{1/2} \log p_i)$ time, so the total expected time over all tests is $O(p_i)$.

When $e_i \geq 2$, we need to compute a primitive root of p_i^2. Theorem 4-(5) assures us that a primitive root p_i^2 can be computed in time $O(\log p_i)$ and Theorem 4-(4) tells us that no more than two roots need to be computed for any particular prime factor.

Since the computation for a primitive root of p_i can be done in $O(p_i)$ time and $\sum_{i=1}^{s} p_i \leq n$, the computation of a primitive root of p_i for $i = 1, 2, \ldots, s$ has time bound $O(m)$. Therefore Step 3 can always be completed in $O(n)$ time.

Step 4 finding the index of 2 with respect to the primitive roots, can be done by sequential trial and error searching. It thus takes time $O(\Sigma p_i)$, which is $O(n)$.

Step 5 is repeated for all divisors, of which there can be no more than $2\sqrt{n}$. A better estimate is given in [BS96, page 234]. Assume that the systematic generation of divisors increments e_i for one and only one prime factor at each iteration. We consider each of the substeps and multiply an upper bound for each step by $\sqrt{n}$ to obtain an upper bound on the total time over all iterations.

Step 5.1 must compute the index of 2 with respect to $g_{i,2}$ modulo $p_i^{u_i}$ only when $u_i \geq 3$. Theorems 4-(4) and 4-(8) assure us that the new index can be computed from the previous index (with respect to u_{i-1}) in no more than p_i steps. Since $u_i \geq 3$, $p_i \leq n^{1/3}$. Thus the total time requirement over all iterations of this step is $O(n^{1/2}n^{1/3}) = O(n^{5/6})$.

Step 5.2 will recompute one or more c_i. Suppose we must recompute all of them, of which there are $O(\log^2 n)$. Each requires a greatest common divisor calculation, which requires time $O(\log n)$, [Knu81, page 320]. Thus this step requires time $O(\log^3 n)$ for each iteration, and thus the total time requirement over all iterations of this step is $O(n)$.

Step 5.3 can solve the set of simultaneous equations using Theorem 2-(2). Note that within the set of $\left(\frac{M}{m_i}\right)^{\varphi(m_i)}$s only one element changes per iteration. Thus the set of solutions at Step 5.3 can be obtained at the expense of one addition and one multiplication per solution. Thus the total time requirement over all iterations of this step is $O(c)$, where c is the number of cycles in the perfect shuffle permutation. Clearly the total time requirement is $O(n)$.

Since the time spent at each step requires time $O(n)$ the total time required is itself $O(n)$. Although some of the steps can be computed more quickly than by the simple methods here indicated, there are apparently unavoidable bottle-necks at or close to $O(n)$.

5 Conclusions

We have shown that it is possible to realise the perfect shuffle permutation in-place and in linear time by way of computing the position of one element in each of the cycles comprising the permutation. We have exposed the structure of the permutation in terms of elementary number theoretic concepts and find that it is possible to compute a seed set, and hence the permutation itself, within these time and space bounds.

We have also shown that the cycles of the perfect shuffle permutation have a nice algebraic structure. We believe this algebraic analysis can be used to study the multi-way perfect shuffle.

The method described provides an alternative to that given in [EM00] by trading off a big saving in the number of element moves against increased calculation of indices.

Acknowledgement The authors are most grateful to the referees for many valuable comments and suggestions, which have made the final result much stronger and have improved the presentation of the paper considerably.

References

[Agn72] J. Agnew. *Explorations in Number Theory.* Brooks/Cole, Monterey,California, 1972.

[All84] J.-P. Allouche. Suites infinies à répétitions bornées. *Séminaire de Théorie des Nombres de Bordeaux*, exposé 20, 1983-1984.

[Bat91] K. E. Batcher. Decomposition of perfect shuffle networks. In *Proceedings of the 1991 International Conference on Parallel Processing*, volume I, Architecture, pages I255–I262, Boca Raton, FL, August 1991. CRC Press.

[BS96] E. Bach and J. Shallit. *Algorithmic Number Theory, Vol 1.* The MIT Press, 1996.

[Bur62] D. A. Burgess. On character sums and primitive roots. *Proc. London Math. Soc.*, 12:179–192, 1962.

[DGK83] P. Diaconis, R. L. Graham, and W. Kantor. The mathematics of perfect shuffles. *Advances in Applied Mathematics*, 4(2):175–196, 1983.

[EM00] J. Ellis and M. Markov. In-situ, stable merging by way of the perfect shuffle. *The Computer Journal*, 43(1):40–53, 2000.

[FMP95] Faith E. Fich, J. Ian Munro, and Patricio V. Poblete. Permuting in place. *SIAM Journal on Computing*, 24(2):266–278, 1995.

[Jon64] B. W. Jones. *Theory of Numbers, Vol 1.* Holt, Rinehart, Winston, 1964.

[Knu73] Donald Knuth. *The Art of Computer Programming, Vol. 3, Sorting and Searching.* Addison-Wesley, 1973.

[Knu81] D. E. Knuth. *The Art of Computer Programming, vol.2, Seminumerical Algorithms.* Addison-Wesley, 1981.

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufman, San Mateo, 1992.

[MM87] S. Medvedoff and K. Morrison. Groups of perfect shuffles. *Math. Magazine*, 60(1):3–14, 1987.

[RS62] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math.*, 6:64–94, 1962.

[Sha69] D. Shanks. Class number, a theory of factorization, and genera. *Number Theory Institute, 1969, Proceedinings of Symposia in Pure Mathematics*, 20:415–440, 1969.

[Sto71] H. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, C-20(2):153–161, 1971.

[Wan59] Y. Wang. On the least primitive root of a prime. *Acta Math. Sinica*, 9:432–441, 1959.