AN EXPLICIT UNIVERSAL CYCLE FOR THE $(n-1)$-PERMUTATIONS OF AN n-SET

FRANK RUSKEY AND AARON WILLIAMS

Abstract. We show how to construct an *explicit* Hamilton cycle in the directed Cayley graph $\overrightarrow{\mathrm{Cay}}(\{\sigma_n, \sigma_{n-1}\} : \mathbb{S}_n)$, where $\sigma_k = (1\ 2\ \cdots\ k)$. The existence of such cycles was shown by Jackson (Discrete Mathematics, **149** (1996) 123–129) but the proof only shows that a certain directed graph is Eulerian, and Knuth (Volume 4 Fascicle 2, Generating All Tuples and Permutations (2005)) asks for an explicit construction. We show that a simple recursion describes our Hamilton cycle and that the cycle can be generated by an iterative algorithm that uses $O(n)$ space. Moreover, the algorithm produces each successive edge of the cycle in constant time; such algorithms are said to be *loopless*.

1. Introduction and motivation

There are many proofs in the mathematical literature showing the existence of Hamilton cycles or Eulerian cycles in important families of graphs. However, turning these proofs into efficient algorithms often represents a significant challenge.

An interesting case in point is the well-known De Bruijn cycle, which is a length k^n circular string over a k-ary alphabet with the property that every length n string occurs as a substring. The existence of De Bruijn cycles is commonly presented in undergraduate discrete mathematics courses as a consequence of a certain graph being Eulerian. However, it is not widely known how to efficiently generate a De Bruijn cycle. In the authors' view two aspects of this question have particular importance.

- **Space, not time, is the primary enemy.** A naïve solution would be to build the graph and then use a Eulerian cycle algorithm to produce the cycle. This will be practical for small values of n and k, but for large values space will be the limiting factor long before time becomes a factor. In general, we need to be able to generate the Hamilton or Eulerian cycle *without* building the graph, or storing exponentially-long sublists. There are algorithms for building De Bruijn cycles that use space $O(n)$. The earliest of these is due to Fredricksen and Maiorana [3] and is presented in Knuth [8].
- **The development of efficient algorithms reveals structure.** It is often worthwhile to turn a proof into an algorithm, or to develop an alternate proof, because the

Research supported in part by NSERC.

> process often results in a deeper structural understanding of the cycles being listed. For example, the efficient algorithm due to Fredricksen and Maiorana is based on necklaces, Lyndon words, and is related to pattern-matching and Lyndon factorization.

As another example from the Hamiltom cycle domain, Eades, Hickey, and McKay [2] considered the graph $G(n, k)$ whose vertices are all length n bitstrings with density k and where two bitstrings are joined by an edge if they differ by transposing two adjacent bits. They showed that $G(n, k)$ is Hamiltonian if and only if n is even and k is odd. The proof is inductive and relies on the fact that the graph has a spanning subgraph that is the prism of two "combs." However, it was not at all clear how to turn that proof into an efficient algorithm. Eventually an algorithm that mimics the proof was found that uses $O(n)$ space and take time $O(1)$ per bitstring generated [5].

In the present paper we are considering the construction of a "universal cycle" for the $(n-1)$-permutations of an n-set (which we take to be $\{1, 2, \ldots, n\}$). Here a *universal cycle* is a circular string of length $n!$ what contains each of the $n!$ different $(n-1)$-permutations as a (contiguous) substring. For example, 321312 is a such a universal cycle for $n = 3$, since its substrings are 32, 21, 13, 31, 12, and 23.

More general universal cycles were introduced by Chung, Diaconis, and Graham [1] as a way of extending the de Bruijn cycle idea to combinatorial objects in general. The existence of a universal cycle for the k-permutations of an n-set was shown by Jackson [6] when $k < n$. His proof sets up a certain natural Eulerian graph, call it $J_{k,n}$, and shows that any Eulerian cycle in that graph corresponds to the required universal cycle. However, no explicit construction of the cycle is indicated. The problem for $k = n - 1$ is discussed by Knuth [8] in Exercise 112 of Section 7.2.1.2. On page 121 of [8] we find the following quote:

> "At least one of these cycles must almost surely be easy to describe and to compute, as we did for de Bruijn cycles in Section 7.2.1.1. But no simple construction has yet been found."

The purpose of this paper is to provide such a description and computational method. We will show how to construct a particular universal cycle. Our algorithm takes space $O(n)$ and uses a constant amount of time between successive outputs of characters in the cycle. To be precise regrading the space requirement: The algorithm uses a constant number of arrays, each with $O(n)$ indices, and each storing integers of value at most n. Similarly regarding time, we use a constant number of operations (comparisons, increments, decrements, and parity tests) on integers of value at most n.

Universal cycles for the permutations of an n-set are not directly possible unless $n \leq 2$. However, every $(n-1)$-permutation of an n-set can be uniquely extended to a permutation of an n-set by appending the unique missing symbol. Thus, universal cycles for $(n-1)$-permutations can be viewed as universal cycles for permutations. For example, 321312

produces the permutations 32$\underline{1}$, 21$\underline{3}$, 13$\underline{2}$, 31$\underline{2}$, 12$\underline{3}$, and 23$\underline{1}$, where the appended missing symbols are underlined. For this reason, our results add to the already sizeable literature on generating permutations. A good survey is provided by Sedgewick [11] and more recent developments are to be found in Knuth [8].

We don't expect our algorithms to be a fast way to generate permutations using the usual model of computation, since at least $n-1$ of the n values change at each step. However, they will be fast if a circular representation is used; for example, when using linked lists or a circular array. In a circular array we maintain a start position and do arithmetic on indices mod n. They will also be fast if the permutation is stored as a computer word. For example, we can store the permutations up to $n = 16$ by dividing 64 bit words into 4 half-bytes. The shifts can then be accommodated in a few machine instructions.

Finally, we mention that additional symbols can also be used to create universal cycles whose substrings are *order isomorphic* to permutations. For example, 421423 produces the permutations 321, 213, 132, 312, 123, and 231. Recently Johnson [7] proved a conjecture in [1] by showing that $n+1$ symbols are always sufficient for constructing these universal cycles.

The paper is organized as follows. In Section 2 we give our explicit construction as a certain recursively defined string. Then, in Section 3, we show that this string can be generated by an algorithm that uses only a constant amount of computation between the output of successive symbols of the string — the first such algorithm for a universal cycle. In Section 4, we give further properties of our recursive construction; first some results on the number of σ_n or σ_{n-1} operations that are used, then that our ordering has an efficiently computable ranking function, and finally that it is "multiversal," in a sense to be described later. We conclude with Section 5, which contains some open problems.

2. An explicit construction

Initially, we will couch our discussion in terms of finding Hamilton paths in certain directed Cayley graphs. Cayley graphs are denoted $X = \overrightarrow{\mathrm{Cay}}(\{\alpha_1, \alpha_2, \cdots, \alpha_k\} : \mathbb{G})$. Here $\{\alpha_1, \alpha_2, \cdots, \alpha_k\}$ is a generating set of a group $\mathbb{G}$. The vertices of X are the elements of $\mathbb{G}$ and the edges are all of the form $g \to \alpha_i g$; these edges are usually thought of as being labelled with α_i. In an *undirected* Cayley graph, if α is in the generating set, then its inverse α^- is also in the generating set. Driven by the question of Lovász of whether there is a Hamilton cycle in all *undirected* Cayley graphs, there is a significant literature of results about Hamilton cycles in Cayley graphs. A survey may be found in Gallian and Witte [4]; see also Pak and Radoš Radoičić [10].

In the solution to Exercise 112 of Section 7.2.1.2 Don Knuth implicitly poses the problem of finding an explicit expression for universal cycles of $(n-1)$-permutations of an n-set [8]. This problem is equivalent to generating permutations of an n-set by rotations of the form

$(1\ 2\ \cdots\ n)$ or $(1\ 2\ \cdots\ n{-}1)$; i.e., it is equivalent to asking whether the Cayley graph

$$\Xi_n := \overrightarrow{\mathrm{Cay}}(\{\sigma_n, \sigma_{n-1}\} : \mathbb{S}_n)$$

is Hamiltonian. We use $\mathbb{S}_{k,n}$ to denote the set of k-permutations of the n-set $[n] = \{1, 2, \ldots, n\}$. In the case where $k = n$ we use $\mathbb{S}_n$. Although we do not use this fact below, it is interesting to note that a short proof reveals that the graph Ξ_n is the *line graph* of the Jackson graph $J_{n-1,n}$.

Consider the binary string S_n defined by the following recursive rules. The base case is $S_2 = 00$. Let $S_n = x_1 x_2 \cdots x_{n!}$ where $\overline{x}$ denotes flipping the bit x. Then, for $n > 2$,

$$S_{n+1} := 001^{n-2}\ \overline{x}_1 001^{n-2}\ \overline{x}_2 \cdots 001^{n-2}\ \overline{x}_{n!}. \tag{1}$$

We use above the usual convention that if w is a string and m is an integer then w^m is w concatenated together m times, $w^m = ww\cdots w$; also w^0 is the empty string.

Below we list S_3, S_4, and S_5. Each S_i is of the form ww since 00 has this property and the recurrence (1) preserves it.

$$\begin{aligned}
S_3 &= 00\ \bar{0}\ 00\ \bar{0} \quad = \quad 00\ 1\ 00\ 1.\\
S_4 &= 001\ \bar{0}\ 001\ \bar{0}\ 001\ \bar{1}\ 001\ \bar{0}\ 001\ \bar{0}\ 001\ \bar{1} = 001\ 1\ 001\ 1\ 001\ 0\ 001\ 1\ 001\ 1\ 001\ 0.\\
S_5 &= (\ 0011\ 1\ 0011\ 1\ 0011\ 0\ 0011\ 0\ 0011\ 1\ 0011\ 1\\
&\qquad 0011\ 0\ 0011\ 0\ 0011\ 1\ 0011\ 1\ 0011\ 0\ 0011\ 1\ )^2
\end{aligned}$$

Now define the mapping ϕ by $0 \to \sigma_n$ and $1 \to \sigma_{n-1}$ where $\sigma_k = (1\ 2\ \cdots\ k)$.

Theorem 2.1. *The list $\phi(S_n)$ is a Hamilton cycle in the directed Cayley graph Ξ_n.*

Proof. In listing the Hamilton cycle we use one-line notation for the permutations, starting with $n\ n{-}1\ \cdots\ 2\ 1$, and think of the cycles σ_{n-1} and σ_n as acting on the positions in the one-line notation. Thus, in a slight abuse of notation,

$$\phi(S_3) = 321, 213, 132, 312, 123, 231,$$

since S_3 implies the successive application of σ_3, σ_3, σ_2, σ_3, σ_3, and finally σ_2 to map the last permutation to the first.

Our proof strategy is to give an explicit listing of permutations of $[n]$ with the required properties and then show that it is equivalent to (1). Recursively define a *circular* list $\Pi(n) = \Pi(n)_0, \Pi(n)_1, \ldots, \Pi(n)_{n!-1}$ of permutations of $[n]$. For small values of n, define $\Pi(1) = 1$, $\Pi(2) = 21, 12$, and $\Pi(3) = \phi(S_3)$. Every n-th permutation of $\Pi(n)$ is defined as follows.

$$\Pi(n)_{jn} := n\Pi(n-1)_j. \tag{2}$$

The $n-1$ permutations that follow $n\pi$, where $\pi = \Pi(n-1)_j$, are defined to be

$$\sigma_n(n\pi),\ \sigma_n^2(n\pi),\ \sigma_{n-1}(\sigma_n^2(n\pi)),\ \ldots,\ \sigma_{n-1}^{n-3}(\sigma_n^2(n\pi)). \tag{3}$$

The list $\Pi(4)$ is shown in Table 1, column (d). The permutation $n\pi$ followed by the permutations above comprise the sublist $\Pi(n)_{jn}, \Pi(n)_{jn+1}, \ldots, \Pi(n)_{(j+1)n-1}$ and these permutations are all distinct since the position of n is successively in the n different positions 1, n, $n-1$, …, 2. Furthermore, because we can recover π from any permutation in this sublist, the uniqueness of every permutation in $\Pi(n)$ follows inductively from the uniqueness of every permutation in $\Pi(n-1)$.

It remains only to prove that successive permutations differ by σ_n or σ_{n-1} and that the list is circular. It is clear from (3) that successive permutations differ by σ_n or σ_{n-1}, except for those that precede the one of the form $n\pi$ successively followed by $n\pi$. Let $a\tau z$ be a permutation of $1, 2, \ldots, n-1$ where a and z are numbers and τ is a sequence (of length $n-3$). Note that the last permutation of (3) is

$$\sigma_{n-1}^{n-3}(\sigma_n^2(na\tau z)) = \sigma_{n-1}^{-2}(\sigma_n^2(na\tau z)) = \sigma_{n-1}^{-2}(\tau z n a) = zn\tau a.$$

Now suppose that $\pi = \Pi(n-1)_j = a\tau z$ and $\pi' = \Pi(n-1)_{j+1}$. Inductively, either $\pi' = \sigma_{n-1}(\pi)$ or $\pi' = \sigma_{n-2}(\pi)$. Observe that

$$\sigma_n(zn\tau a) = n\tau az = n\sigma_{n-2}(a\tau z), \text{ and} \tag{4}$$

$$\sigma_{n-1}(zn\tau a) = n\tau za = n\sigma_{n-1}(a\tau z). \tag{5}$$

Since the successor of $a\tau z$ is either $\sigma_{n-2}(a\tau z)$ or $\sigma_{n-1}(a\tau z)$, the transition to the permutation π' is also of the correct form; successive permutations in $\Pi(n)$ differ by σ_n or σ_{n-1}. The circularity of the list follows inductively from the circularity of the list $\Pi(n-1)$ (alternatively we could use Lemma 2.2 below). Furthermore, in terms of the mapping ϕ defined earlier, the bits are flipped; a 0 (σ_{n-1}) transition in $\Pi(n-1)$ becomes a 1 (σ_{n-1}) transition in $\Pi(n)$ by (5), and a 1 (σ_{n-2}) transition in $\Pi(n-1)$ becomes a 0 (σ_n) transition in $\Pi(n)$ by (4). □

Lemma 2.2. *Any Hamilton path in Ξ_n is, in fact, a Hamilton cycle.*

Proof. Suppose that $\Pi = \Pi_1, \Pi_2, \ldots, \Pi_{n!}$ is a Hamilton path in Ξ_n that is not a Hamilton cycle. In particular $\sigma_n(\Pi_{n!}) \neq \Pi_1$ and $\sigma_{n-1}(\Pi_{n!}) \neq \Pi_1$. Thus $\Pi_{n!} \neq \sigma_n^-(\Pi_1)$ and $\Pi_{n!} \neq \sigma_{n-1}^-(\Pi_1)$. We must then have that $\sigma_n^-(\Pi_1) \to \sigma_{n-1}(\sigma_n^-(\Pi_1))$ and $\sigma_{n-1}^-(\Pi_1) \to \sigma_n(\sigma_{n-1}^-(\Pi_1))$ are distinct edges in $\Pi(n)$. However, an easy calculation shows that $\sigma_{n-1}^-\sigma_n = \sigma_n^-\sigma_{n-1} = (n{-}1\ n)$ and thus these permutations are identical. This contradiction shows that Π is a Hamilton cycle. □

The proof shows that in fact the lemma is true for any Cayley graph on two generators ρ and τ for which $\tau^-\rho$ is an involution.

The universal cycle for $(n-1)$-permutations of $[n]$ is obtained by recording the first symbol in each of the permutations in $\Pi(n)$. We use U_n to denote the resulting universal cycle.

3. A loopfree algorithm

Suppose that in our recurrence (1) for S_{n+1} that for each "new" bit we record the value n, and apply this idea recursively. Call the corresponding new sequence R_{n+1}. That is, $R_2 = 11$, and for $n > 1$,

$$R_{n+1} = n^n y_1 n^n y_2 \cdots n^n y_{n!},$$

where $R_n = y_1 y_2 \cdots y_{n!}$. For example

$$R_4 = 333\ 2\ 333\ 2\ 333\ 1\ 333\ 2\ 333\ 2\ 333\ 1.$$

The sequence R_4 is exactly the sequence that is obtained by recording the most significant position that changes when counting with the multi-radix numbers with parameters $2 \times 3 \times 4$, when the numbers are indexed 1, 2, 3, from left-to-right. See Table 1, columns (a) and (b). In general, R_n gives us the positions when counting with multi-radix numbers $2 \times 3 \times \cdots \times n$.

These observations suggest that we may be able to efficiently generate the S_n sequence by modifying the classic algorithm for counting with multi-radix numbers. In the classic algorithm the multi-radix number is stored in the array $a_{n-1} \cdots a_2 a_1$ and j is used to represent the rightmost, or smallest, index where a_j is not at its maximum value. The next multi-radix number is obtained by incrementing a_j and setting all values to its right to 0. Now suppose that we just incremented the integer in position j so that the multi-radix number is $a_{n-1} \cdots a_j a_{j-1} \cdots a_1 = a_{n-1} \cdots a_j 0 \cdots 0$. Then the corresponding R_n value is $n - j$ and so the non-recursive part of the S_j sequence that we are listing is going through the pattern 001^{n-j-1} or the pattern 110^{n-j-1}, depending on whether j is odd or even, respectively. For proposition P we use the notation $[\![P]\!]$ to mean the value 1 if P is true and the value 0 if P is false; also $\oplus$ denotes exclusive-or. The expression $[\![j \text{ even } \oplus a_j \leq 1]\!]$ gives the correct value of the bit to be output. Below is the entire algorithm, rendered in pseudo-code.

```
a_{n+1} a_n ··· a_1 ← 0 0 ··· 0;
repeat
        j ← 1;
        while a_j = n − j do a_j ← 0; j ← j + 1; od;
        output( [[j even ⊕ a_j ≤ 1]] );
        a_j ← a_j + 1;
until j ≥ n;
```

There is an loopless algorithm for listing multi-radix numbers as a Gray code in which the value in only one position changes and that change is by ± 1 (see, for example, Williamson [12], pg. 112, or Knuth [8], pg. 20). Together with the ideas used in the previous "counting" algorithm, we can adapt those loopless algorithms to get a loopless algorithm for generating S_n or our universal cycle. In the Gray code for multi-radix numbers, the values in a given position alternately increase and decrease. Furthermore, the values change in exactly the

(a) 234	(b) R_4	(c) 234	(d) S_4	(e) $\mathbb{S}_4$	(f) U_4	(g) rank
000	3	000	. . 0	4321	4	0
001	3	001	. . 0	3214	3	1
002	3	002	. . 1	2143	2	2
003	2	003	. 1 .	1423	1	3
010	3	013	. . 0	4213	4	4
011	3	012	. . 0	2134	2	5
012	3	011	. . 1	1342	1	6
013	2	010	. 1 .	3412	3	7
020	3	020	. . 0	4132	4	8
021	3	021	. . 0	1324	1	9
022	3	022	. . 1	3241	3	10
023	1	023	0 . .	2431	2	11
100	3	123	. . 0	4312	4	12
101	3	122	. . 0	3124	3	13
102	3	121	. . 1	1243	1	14
103	2	120	. 1 .	2413	2	15
110	3	110	. . 0	4123	4	16
111	3	111	. . 0	1234	1	17
112	3	112	. . 1	2341	2	18
113	2	113	. 1 .	3421	3	19
120	3	103	. . 0	4231	4	20
121	3	102	. . 0	2314	2	21
122	3	101	. . 1	3142	3	22
123	1	100	0 . .	1432	1	23

TABLE 1. (a) Counting in multi-radix base $2 \times 3 \times 4$, (b) the R_4 sequence, (c) the corresponding multi-radix Gray code, (d) indented version of S_4, (e) the list $\Pi(4)$, (f) the universal cycle U_4, and (g) the rank of each permutation.

positions given by the R_n sequence. In the implementation we maintain a direction array d where $+1$ means increase and -1 means decrease We also maintain an array f of "focus pointers" which allow instant access to the next position whose value will change (we set $f_n = n{+}1$ (instead of n) so that the last iteration is handled correctly). See Table 1, column (c), for an example.

Thus the values of j from the counting algorithm are exactly the same in the Gray code algorithm, except that in the Gray code algorithm j is the position where a value changes. The only complication arises because the values in a given position can be decreasing, and so the test "$a_j \leq 1$" is not sufficient. Fortunately, all algorithms that looplessly implement the Gray code maintain an array of directions $d_{n-1} \cdots d_2 d_1$ for each position, where $d_i \in$

$\{+1, -1\}$, indicating whether the values in that position are currently increasing $(+1)$ or decreasing (-1). If $d_j = +1$ then we can continue to test $a_j \leq 1$, but to account for $d_j = -1$, we need to test

$$(a_j \leq 1 \ \textbf{and} \ d_j = 1) \ \textbf{or} \ (a_j \geq n - j - 1 \ \textbf{and} \ d_j = -1).$$

We can "optimize" this condition. Notice that the test $(a_j \leq 1 \ \textbf{and} \ d_j = 1)$ can be replaced by $a_j - d_j \leq 0$. This change is possible because if $d_j = -1$ then $a_j - d_j$ is guaranteed to be greater than zero because $a_j \geq 0$. Therefore, if $a_j - d_j \leq 0$, then this immediately implies that $d_j = 1$ and so $a_j - 1 \leq 0$, which is equivalent to the original test $a_j \leq 1$. Likewise, the test $(a_j \geq n-j-1 \ \textbf{and} \ d_j = -1)$ can be replaced by $a_j - d_j \geq n - j$. Below is our loopless algorithm in full detail.

$a_{n+1}a_n \cdots a_1 \leftarrow 0\ 0\ 0\ \cdots\ 0$;
$d_n d_{n-1} \cdots d_1 \leftarrow 1\ 1\ 1\ \cdots\ 1$;
$f_n f_{n-1} \cdots f_1 \leftarrow n{+}1\ n{-}1\ n{-}2\ \cdots\ 1$;
repeat
    $j \leftarrow f_1$; $f_1 \leftarrow 1$;
    output($[\![\ j$ even $\oplus\ (a_j - d_j \leq 0 \ \textbf{or} \ a_j - d_j \geq n - j)\ ]\!]$);
    $a_j \leftarrow a_j + d_j$;
    if $a_j = 0$ **or** $a_j = n - j$ **then** $d_j \leftarrow -d_j$; $f_j \leftarrow f_{j+1}$; $f_{j+1} \leftarrow j + 1$; **fi**;
until $j \geq n$;

It is also possible to output the universal cycle itself in a loopless manner, but an additional circular array is required to hold the current permutation. To follow are the details. Define an array $\pi_1\pi_2 \cdots \pi_n$ initialized to $n\ n{-}1\ \cdots\ 1$ and an index t that will be incremented mod n on each iteration of the algorithm. We will think of π as a circular array. The index t is the position of the last element of π, so initially $t = n$. As each bit of S_n is determined, we will ouput the first element of π (i.e., the one in position $t + 1$). If the bit is a 1, so that σ_{n-1} is acting on π then we need to swap the last two elements: $\pi_{t-1} \leftrightarrow \pi_t$. In other words the output statements in the preceding code fragments is replaced with the following code where *expr* is the expression inside of the output statement in either the previous counting algorithm of the previous loopless algorithm.

$t' \leftarrow t$; $\quad t \leftarrow (t + 1) \bmod n$;
output($\pi_{(t+1) \bmod n}$);
if *expr* $= 1$ **then** $\pi_t \leftrightarrow \pi_{t'}$ **fi**;

Finally, we note that every permutation can be output in a circular fashion by outputting π and t. We could also use a linked list, which would give a loopless permutation generation algorithm.

4. Further properties

In this section we explore further properties of Ξ_n and our Hamilton cycle.

4.1. **How many of each rotation is used?** It is clear from the recurrence relation (1) that the number, call it f_n, of σ_n's in $\phi(S_n)$ satisfies the recurrence relation

$$f_{n+1} = \begin{cases} 2 & \text{if } n = 1 \\ 3n! - f_n & \text{if } n > 1. \end{cases} \tag{6}$$

This recurrence relation can be iterated to obtain

$$f_n = 2(-1)^n - 3\sum_{k=1}^{n-1}(-1)^k(n-k)!,$$

from which it follows that

$$f_n \sim 3(n-1)! \quad \text{or} \quad \frac{f_n}{n!} \sim \frac{3}{n}.$$

Interestingly, this sequence appears in OEIS [9] as A122972$(n+1)$ as the solution to the "symmetric" recurrence relation $a(n+1) = (n-1)\cdot a(n) + n\cdot a(n-1)$. The values of f_n for $n = 1, 2, \ldots, 10$ are 1, 2, 4, 14, 58, 302, 1858, 13262, 107698, 980942.

Consider the cosets induced by σ_n; there are $n!/n = (n-1)!$ of them. In a Hamilton cycle there must be at least one σ_{n-1} edge that leaves each coset, and thus there must be at least $(n-1)!$ of them. Alternatively, consider the cosets induced by σ_{n-1}; there are $n!/(n-1) = n\cdot(n-2)!$ of them. In a Hamilton cycle there must be at least one σ_n edge that leaves the coset, and thus there must be at least $n\cdot(n-2)!$ of them. We can make a stronger statement regarding the σ_n edges.

Lemma 4.1. *The least number of σ_n edges in any Hamilton cycle in Ξ_n is $2n(n-2)!-2$.*

Proof. First, observe that

$$\sigma_{-1}^{-}\sigma_n\sigma_{n-1}^{-}\sigma_n = (n{-}1\ n)(n{-}1\ n) = id.$$

The two σ_n edges above are incident with the same unordered pair of cosets induced by σ_{n-1}. Thus if we contract each coset into a singe super-vertex, then the resulting graph, call it Q_n, is undirected in the sense that every directed edge is paired with an edge in a 2-cycle. Furthermore, it is not hard to see that if one of those σ_n edges is used in a Hamilton cycle, then so must the other. Thus a Hamilton cycle in Ξ_n becomes a connected spanning subgraph of Q_n. Since a minimal connected spanning subgraph is a spanning tree, and any spanning tree has $n\cdot(n-2)!-1$ edges, the number of σ_n edges is at least $2n(n-2)!-2$. □

Figure 1 shows the Cayley graph X_4. Note that the contracted graph Q_4 is the 3-cube. The red edges show the Hamilton cycle S_4. In this case S_n corresponds to a spanning tree in Q_n, but this is not the case for $n \geq 6$.

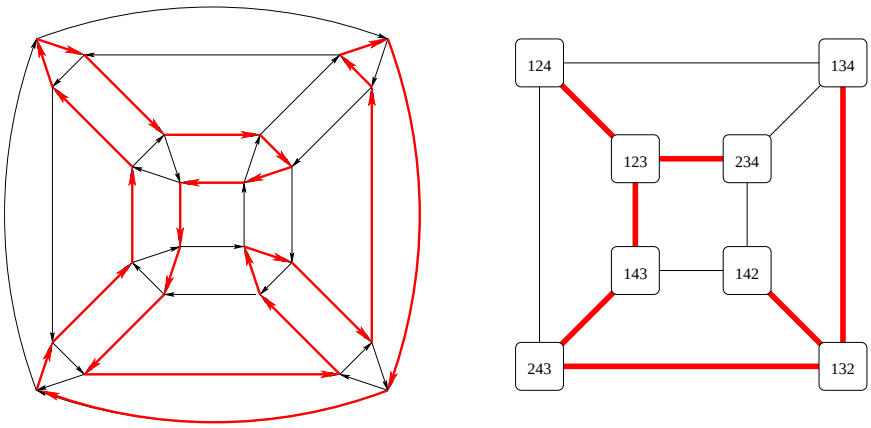


FIGURE 1. The Cayley graph Ξ_4 on the left. The graph Q_4 on the right. The thick (red) edges indicate the Hamilton cycle S_4.

4.2. **Ranking.** The *rank* of a permutation π is the value r for which $\Pi(n)_r = \pi$. Our recursive equation for the rank depends on the position of n within the permutation being ranked. From the definition of $\Pi(n)$ we can infer that

$$\begin{aligned}&\operatorname{rank}(a_1 a_2 \cdots a_{k-1} n a_{k+1} \cdots a_n)\\ &= \begin{cases} 0 & \text{if } n = 1,\\ n \cdot \operatorname{rank}(a_2 a_3 \cdots a_n) & \text{if } k = 1,\\ n - k + 1 + n \cdot \operatorname{rank}(a_n a_{k+1} \cdots a_{n-1} a_1 \cdots a_k) & \text{if } k > 1. \end{cases}\end{aligned}$$

The expression $n - k + 1$ accounts for the position of the n, and the rest comes from the recursive part of the definition of $\Pi(n)$. We can also express the rank as

$$\operatorname{rank}(\ \alpha n \beta\ ) = \begin{cases} 0 & \text{if } \alpha = \beta = \epsilon,\\ n \cdot \operatorname{rank}(\ \beta\ ) & \text{if } \alpha = \epsilon,\\ n - |\alpha| + n \cdot \operatorname{rank}(\ \sigma(\beta)\alpha\ ) & \text{otherwise ,} \end{cases}$$

where $\sigma(\beta)$ is β rotated one position to the right.

Implemented in the obvious manner, these recurrence relations lead to algorithms that use $O(n^2)$ arithmetic operations on integers as large as $n!$.

4.3. **Multiversal Cycle Property.** In this section we prove that the sequence $\Pi(n) = \Pi(n)_0, \Pi(n)_1, \ldots, \Pi(n)_{n!-1}$, written out as a long string of symbols by concatenating each permutation, is a "multiversal cycle". We denote this "flattening" of $\Pi(n)$ as $\coprod(n)$. For

example, consider $\coprod(3) = 321\ 213\ 132\ 312\ 123\ 231$. Starting in positions 0,1, or 2 and advancing the position in increments of 3, recording the first two symbols, we obtain

0	32	21	13	31	12	23
1	21	13	32	12	23	31
2	12	31	23	21	32	13

In each case a complete set of all 2-permutations of [3] is obtained. The purpose of this section is to prove that this property holds in general.

Definition 4.2. A *multiversal* cycle for the $(n-1)$-permutations of an n-set is a circular string $a_0a_1\cdots a_{N-1}$ of length $N = n \cdot n!$ such that, for all $m = 0, 1, \ldots, n-1$,

$$\{a_{m+in}\cdots a_{m+in+n-2} \mid i = 1, 2, \ldots, n!\} = \mathbb{S}_{n-1,n}, \tag{7}$$

where arithmetic in the indices is taken mod n.

Before getting to the main theorem in this section we prove a technical lemma.

Lemma 4.3. *For all $i \neq 0, -1 \bmod n$, if $\coprod(n) = a_0a_1\cdots a_{N-1}$, then*

$$a_i = a_{i+n-1}.$$

Proof. Because $i \neq 0 \bmod n$ the numbers a_i and a_{i+n-1} lie in two successive permutations of $\Pi(n)$. The conclusion now follows since successive permutations differ by σ_n or σ_{n-1}. The $i \neq -1 \bmod n$ condition is necessary when they differ by σ_{n-1}. □

Theorem 4.4. *The string $\coprod(n) = a_0a_1\cdots a_{n!-1}$ is a multiversal cycle.*

Proof. The proof is by induction on the value m in the definition. The base case $m = 0$ satisfies (7) because $\Pi(n)_0, \Pi(n)_1, \ldots, \Pi(n)_{n!-1}$ is a listing of all permutations of $[n]$, so ignoring the last character of each permutation gives a complete listing of all $(n-1)$-permutations of $[n]$. Similarly, when $m = 1$, ignoring the first character of each permutation also gives a complete listing of all $(n-1)$-permutations of $[n]$. We now argue by contradiction. Suppose that there are some values $m > 1$, i and i', with $i \neq i'$, such that

$$a_{m+in}\cdots a_{m+in+n-2} = a_{m+i'n}\cdots a_{m+i'n+n-2}. \tag{8}$$

Inductively, we know that

$$a_{m-1+in}\cdots a_{m-1+in+n-2} \neq a_{m-1+i'n}\cdots a_{m-1+i'n+n-2}.$$

Thus it must be the case that $a_{m-1+in} \neq a_{m-1+i'n}$. However, applying Lemma 4.3 to a_{m-1+in} and $a_{m-1+i'n}$ gives

$$a_{m-1+in} = a_{m-1+in+n-1} \text{ and } a_{m-1+i'n} = a_{m-1+i'n+n-1},$$

so long as $m \neq 0, 1$. But by (8) we now have

$$a_{m-1+in} = a_{m-1+in+n-1} = a_{m+in+n-2} = a_{m+i'n+n-2} = a_{m-1+i'n+n-1} = a_{m-1+i'n},$$

which is a contradiction. □

The careful reader will have noted that Lemma 4.3 and Theorem 4.4 apply to *any* Hamilton cycle in Ξ_n since the only property that we use is that successive permutations differ by σ_{n-1} or σ_n.

5. Final Remarks, Open Problems

In this paper we have developed an explicit algorithm for generating a universal cycle for the $(n-1)$-permutations of an n-set. This is the first universal cycle for which a loopless algorithm has been discovered.

Below is a list of open problems inspired by this work.

- Can the results of this paper be extended to k-permutations of $[n]$ for $1 \leq k < n-1$?
- Among all Hamilton cycles in Ξ_n we determined in Lemma 4.1 the least number of σ_n edges that need to be used in a Hamilton cycle in Ξ_n. What is the least number of σ_{n-1} edges that need be used? In our construction, the number of σ_n edges is asymptotic to $3/n$ and the number of σ_{n-1} edges is asymptotic to $(n-3)/n$. Is there a general construction that uses more σ_n edges than σ_{n-1} edges?
- Can the results of this paper be extended to the permutations of a multiset? That is, given multiplicities $n_0, n_1, \ldots, n_t$, where n_i is the number of times i occurs in the multiset and $n = n_0 + n_1 + \cdots + n_t$, is there a circular string $a_1 a_2 \cdots a_N$ of length $N = \binom{N}{n_0, n_1, \ldots, n_t}$ with the property that

  $$\{a_i\ a_{i+1}\ \cdots\ a_{i+n-2}\ \iota(a_i, a_{i+1}, \ldots, a_{i+n-2}) \mid 1 \leq i \leq N\}$$

  is equal to the set of all permutations of the multiset. Since the length of $a_i a_{i+1} \cdots a_{i+n-2}$ is $n-1$ it is not a permutation of the multiset; one character is missing. The function ι gives the missing character. We call these strings *shorthand universal cycles*. The current paper gave a shorthand cycle for permutations of $[n]$.
- It would be interesting to gain more insight in to the ranking process. Is there a way to iterate the recursion so that it can be expressed as a sum?

References

[1] Fan Chung, Persi Diaconis, and Ron Graham, *Universal cycles for combinatorial structures*, Discrete Mathematics, 110 (1992) 43–59.

[2] P. Eades, M. Hickey, and R.C. Read, *Some Hamilton paths and a minimal change algorithm*, Journal of the ACM, 31 (1984) 19–29.

[3] H. Fredricksen and J. Maiorana, *Necklaces of beads in k colors and k-ary de Bruijn sequences*, Discrete Mathematics, 23 (1978) 207-210.

[4] J. Gallian and D. Witte, *A survey: hamiltonian cycles in Cayley graphs*, Discrete Mathematics, 51 (1984) 293–304.
[5] T. Hough and F. Ruskey, *An Efficient Implementation of the Eades, Hickey, Read Adjacent Interchange Combination Generation Algorithm*, Journal of Combinatorial Mathematics and Combinatorial Computing, 4 (1988) 79–86.
[6] B. Jackson, *Universal cycles of k-subsets and k-permutations*, Discrete Mathematics, 149 (1996) 123–129.
[7] R. Johnson, *Universal cycles for permutations*, Discrete Mathematics, to appear.
[8] D.E. Knuth, *The Art of Computer Programming, Volume 4, Generating All Tuples and Permutations*, Fascicle 2, Addison-Wesley, 2005.
[9] N.J.A. Sloane, *The Online Encyclopedia of Integer Sequences*, `http://www.research.att.com/~njas/sequences/`.
[10] Igor Pak and Radoš Radoičić, *Hamiltonian paths in Cayley Graphs*, manuscript, 2004.
[11] Robert Sedgewick, *Permutation Generation Methods*, Computing Surveys, 9 (1977) 137-164.
[12] S. Gill Williamson, *Combinatorics for Computer Science*, Computer Science Press, 1985.

6. Appendix

This is a not-for-publication appendix of additional information and tables that may assist the referees. It will also be posted on the author's publication website.

Multiversal cycle table for $n = 4$.

0	432	321	214	143	423	213	134	342	412	132	324	241
	431	312	124	243	413	123	234	341	421	231	314	142
1	321	214	143	423	213	134	342	412	132	324	241	431
	312	124	243	413	123	234	341	421	231	314	142	432
2	213	142	431	234	132	341	423	124	321	243	412	314
	123	241	432	134	231	342	413	214	312	143	421	324
3	132	421	314	342	321	413	234	241	213	432	124	143
	231	412	324	341	312	423	134	142	123	431	214	243

Here is a java class to output S_n by counting.

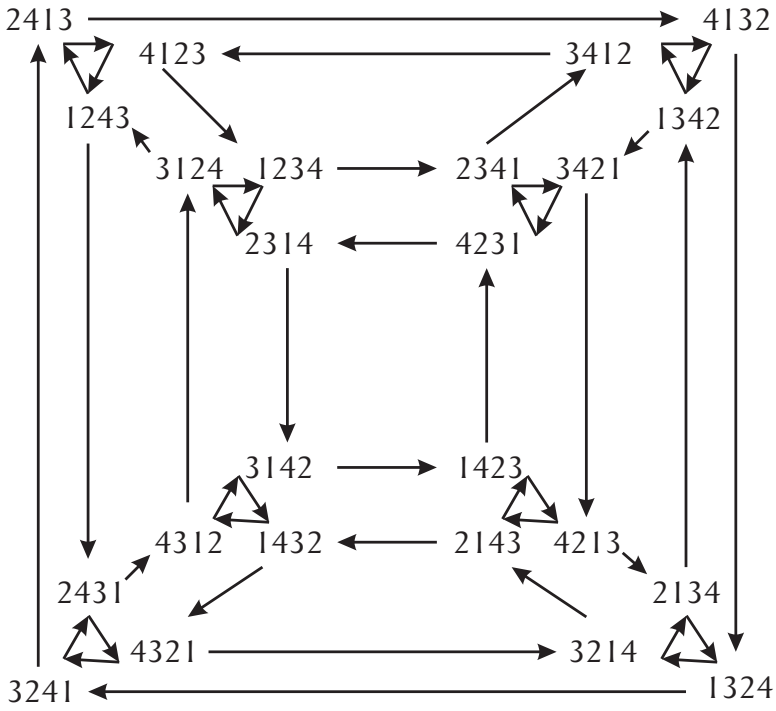
```
class C {
   public static void main ( String[] args ) {
      int n = Integer.parseInt( args[0] );
      int[] a = new int[n+2];
      for (int i=0; i<n+2; ++i) a[i] = 0;
      int j = 0;
      while (j <= n) {
         j = 1;
         while (a[j] == n-j) { a[j] = 0;  ++j; }
         ++a[j];
         System.out.print( ((j%2==1)^(a[j]>2))?0:1 );
      }
      System.out.println();
   }
}
```

Here is a java class to output S_n by the multi-radix Gray code.

```
// Print out 0/1 Cayley graph sequence by counting.
// Last bit needs to be flipped.

class L {
   static int[] a = new int[128];
   static int[] d = new int[128];
   static int[] f = new int[128];

   public static void main ( String[] args ) {
```

```
      for (j=0; j<n+2; ++j) { a[j] = 0; }
      for (j=0; j<n+2; ++j) { d[j] = 1; }
      for (j=0; j<n+2; ++j) { f[j] = j; }  f[n] = n+1;
      do { j = f[1];  f[1] = 1;
           System.out.print( ((j%2==1)^(a[j]-d[j]>0&&a[j]-d[j]<n-j))?0:1 );
           a[j] = a[j] + d[j];
           if (a[j]==0 || a[j]==n-j) {
              d[j] = -d[j];  f[j] = f[j+1];  f[j+1]=j+1;
           }
      } while (j < n);

   }
}
```

Here is a labelled version of Ξ_4.



Dept. of Computer Science, University of Victoria, CANADA

URL: http://www.cs.uvic.ca/~ruskey

Dept. of Computer Science, University of Victoria, CANADA