Faster Generation of Shorthand Universal Cycles for Permutations

Frank Ruskey* and Aaron Williams

Dept. of Computer Science, University of Victoria, CANADA

Abstract. A universal cycle for the k-permutations of $\langle n\rangle = \{1, 2, ..., n\}$ is a circular string of length n_k that contains each k-permutation exactly once as a substring. Jackson (Discrete Mathematics, 149 (1996) 123–129) proved their existence for all $k \leq n-1$. Knuth (*The Art of Computer Programming, Volume 4*, Fascicle 2, Addison-Wesley, 2005) pointed out the importance of the $k = n-1$ case, where each $(n-1)$-permutation is "shorthand" for exactly one permutation of $\langle n\rangle$. Ruskey-Williams (ACM Transactions on Algorithms, in press) answered Knuth's request for an explicit construction of a shorthand universal cycle for permutations, and gave an algorithm that creates successive symbols in worst-case $O(1)$-time. This paper provides two new algorithmic constructions that create successive blocks of n symbols in $O(1)$ amortized time within an array of length n. The constructions are based on a 300 year-old bell-ringer pattern, and the recent shift Gray code by Williams (SODA, (2009) 987-996) and are both implemented in C. For the bell-ringers algorithm, we show that the majority of changes between successive permutations are full rotations; asymptotically, the ratio of them is $(n-2)/n$.

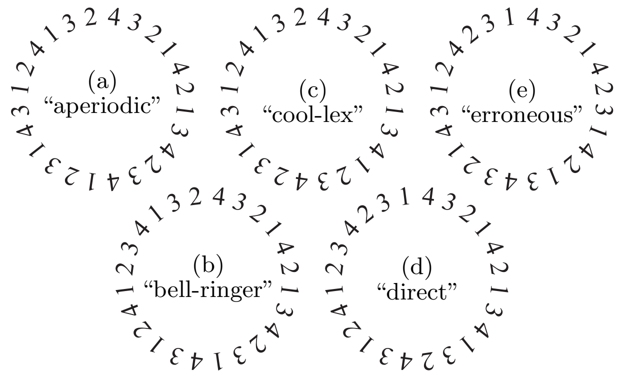


Fig. 1. Cycles (a)-(d) are Ucycles for $\Pi_3(4)$, or equivalently shorthand Ucycles for $\Pi(4)$, and 4 is a periodic symbol in (b)-(d). Cycle (e) contains the erroneous substring 242, as well as an extra copy of 142.

1 Introduction

A *universal cycle* (or *Ucycle*) [1] is a circular string containing every object of a particular type exactly once as substring. For example, consider the circular string in Figure 1(a). Starting from 12 o'clock, and proceeding clockwise, its substrings of length three are

$$432, 321, 214, 142, \ldots, 413, 132, 324, 243. \tag{1}$$

In total there are 24 substrings, and these substrings include every 3-permutation of $\langle 4\rangle$ exactly once. For this reason, Figure 1(a) is a *Ucycle for the* 3*-permutations of* $\langle 4\rangle$. Let $\Pi_k(n)$ denote the set of all k-permutations of $\langle n\rangle$. Notice that $|\Pi_k(n)| = n_k = n(n-1)\cdots(n-k+1)$ (the *falling factorial*). In the special case $k = n$, we use $\Pi(n)$ to represent the permutations of $\langle n\rangle$. Jackson

* Research supported in part by NSERC

proved that Ucycles of $\Pi_k(n)$ exist whenever $k < n$ [6]. On the other hand, the reader should take a moment to convince themselves that Ucycles of $\Pi(n)$ do not exist when $n \geq 3$.

This introductory section describes four interpretations for Ucycles of $\Pi_{n-1}(n)$, then discusses applications, relevant history, and concludes with an outline of our new results.

1.1 Interpretations

To describe the first interpretation, notice that $|\Pi_{n-1}(n)| = n_{n-1} = n! = |\Pi(n)|$. Each $(n-1)$-permutation of $\langle n \rangle$ extends to a unique permutation of $\langle n \rangle$ by appending its *missing symbol* from $\langle n \rangle$. For example, the first string in (1) is 432, and it is missing the symbol 1. We say that the substring 432 is *shorthand* for the permutation 4321. Similarly, the substrings in (1) are shorthand for the following list of permutations

$$4321, 3214, 2143, 1423, 4213, \ldots, 2413, 4132, 1324, 3241, 2431. \tag{2}$$

For this reason, Ucycles for $\Pi_{n-1}(n)$ can be interpreted as Ucycles for permutations, and are called *shorthand Ucycles for $\Pi(n)$*. The substrings comprising $\Pi_{n-1}(n)$ are known as the Ucycle's *substrings*, and the extended strings comprising $\Pi(n)$ are the Ucycles's *permutations*.

The second interpretation of a shorthand Ucycle for $\Pi(n)$ is its *binary representation.* Given a length n substring in a shorthand Ucycle for $\Pi(n)$, the *next symbol* is the symbol that follows this substring in the shorthand Ucycle, and the *next substring* is the length n substring that ends with this next symbol. That is, if $s_1 s_2 \cdots s_{n-1}$ is a substring in a shorthand Ucycle for $\Pi(n)$, then the next symbol is some $x \in \langle n \rangle$, and the next substring equals $s_2 s_3 \cdots s_{n-1} x$. However, since $s_2 s_3 \cdots s_{n-1} x$ is in $\Pi_{n-1}(n)$, then there are only two choices for x. In particular, x equals the missing symbol from $s_1 s_2 \cdots s_{n-1}$ or equals s_1, and this dichotomy gives rise to the binary representation. More specifically, the ith bit in the binary representation is 0 if the substring starting at position i has its next symbol equal to its missing symbol; otherwise the substring starting at position i has its next symbol equal to its first symbol and the ith bit in the binary representation is 1. The binary representation can be visualized by placing two copies of the shorthand Ucycle for $\langle n \rangle$ above itself, with the second copy left-shifted $(n-1)$ positions. This comparison vertically aligns the first symbol of each length n substring with its next symbol. Accordingly, 1s are recorded precisely when the vertically aligned symbols are equal. This is illustrated below for Figure 1(a), where $\underline{4}$ denotes the first symbol in the shorthand Ucycle (above) and in its rotation (below).

$$\begin{array}{c} \underline{4}32142134234123143124132 \\ 142134234123143124132\underline{4}32 \\ \hline 001100011000101100100011 \end{array}$$

To check this binary string, the first substring is 432, and its next symbol and missing symbol are both 1. Therefore, the first bit is 0. On the other hand, the third substring is 214, and its next symbol and first symbol are both 2. Therefore, the third bit is 1. (As above, the shorthand Ucycle for $\Pi(n)$ is assumed to "start" with $n\ (n-1)\ \cdots 2$.)

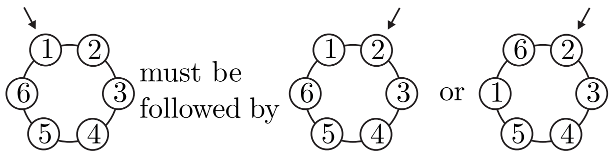
The binary representation can also be described in terms of permutations, our third interpretation. If $p_1 p_2 \cdots p_n$ is a permutation in a shorthand Ucycle for $\Pi(n)$, then the *next permutation* begins with $p_2 p_3 \cdots p_{n-1}$. Therefore, the next permutation is either $p_2 p_3 \cdots p_{n-1} p_n p_1$ or $p_2 p_3 \cdots p_{n-1} p_1 p_n$. These two cases are obtained by rotating the first symbol of $p_1 p_2 \cdots p_n$ into one of the last two positions. More precisely, if the ith bit in the binary representation is 0, then the ith permutation is transformed into the $(i+1)$st permutation by the rotation $\sigma_n = (1\ 2\ \cdots\ n)$, and if the ith bit is 1, then the rotation $\sigma_{n-1} = (1\ 2\ \cdots\ n)$ is used. Thus, permutations in a shorthand Ucycle for

$\Pi(n)$ are in a *circular Gray code* using σ_n and σ_{n-1}. For example, in (2), the first permutation 4321 is transformed into the second permutation 3214 by σ_4 (first bit in the binary representation is 0). On the other hand, the third permutation 2143 is transformed into the fourth permutation 1423 by σ_3 (the third bit in the binary representation is 1). Conversely, every circular Gray code of $\Pi(n)$ using σ_n and σ_{n-1} provides a shorthand Ucycle for $\Pi(n)$ by appending the first symbols of each permutation.

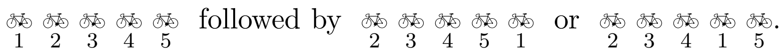
The fourth interpretation for shorthand Ucycles for $\Pi(n)$ is their equivalence to Hamiltonian cycles in the directed Cayley graph on $\Pi(n)$ with generators σ_n and σ_{n-1} (see [16] for more details).

1.2 Some "Applications"

The binary representation provides a natural application for shorthand Ucycles of $\Pi(n)$: $n!$ permutations are encoded in $n!$ bits. The Gray code interpretation also provides applications. The rotations σ_n and σ_{n-1} can also be described respectively as *prefix shifts* of length n and $n-1$. Prefix shifts of length n and $n-1$ can be performed as basic operations within linked lists or circular arrays. In particular, a prefix shift of length n simply increments the starting position within a circular array, whereas a prefix shift of length $n-1$ increments the starting position and then performs an adjacent-transposition of the last two symbols. This is illustrated below with the permutation 123456 (below left) being followed by 234561 or 234516 (below right) with the arrow denoting the starting position and successive symbols being read clockwise within the circular array.



In applications, σ_n and σ_{n-1} may also have special significance. For example, in cycling it is customary for riders in a breakaway group to organize themselves in a line. Riders in the front reduce the wind-resistance for the riders behind, and at regular intervals the lead rider gives up their position to conserve energy. The easiest positions for the lead rider to reinsert themselves is the last or second-last positions, since this involves the least amount of disruption to the other cyclists



By proceeding in an order provided by a shorthand universal cycle for $\Pi(n)$, it can be assured that no rider is given an unfair advantage by continually slipstreaming behind riders of larger sizes (who would further reduce wind-resistance) or from the same team (who would allow the trailing cyclist to ride in an optimal position).

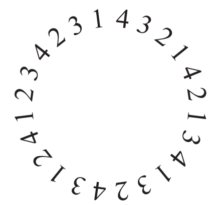
1.3 History

Ucycles for combinatorial objects were introduced by Chung, Diaconis, and Graham [1] as natural generalizations of *de Bruijn cycles* [3], which are circular strings of length 2^n that contain every binary string of length n exactly once as substring. They pointed out that Ucycles of $\Pi(n)$ do not exist, and suggested using order-isomorphism to represent the permutations. A string is *order-isomorphic* to a permutation of $\langle n \rangle$ if the string contains n distinct integers whose relative orders are the same as in the permutation. For example, 321341 is an *order-isomorphic Ucycle for* $\Pi(4)$ since its substrings — $321, 213, 134, 341, 413, 132$ — are order-isomorphic to $321, 213, 123, 231, 312, 132$.

Recently, Johnson [8] resolved a long-standing conjecture [1] that only $n+1$ symbols are necessary for constructing order-isomorphic Ucycles for $\Pi(n)$.

Knuth suggested the use of what we call *shorthand-isomorphism*, and asked for an explicit construction [10]. An explicit construction was discovered by Ruskey-Williams [16], who also provided an algorithm that creates each successive symbol in the Ucycle (or bit in the binary representation) in worst-case O(1)-time. The construction has the property that the symbol n appears in every nth position in the Ucycle. For this reason, it is said to have a *periodic symbol*. See Figure 1(d) for the construction when $n = 4$, and notice that 4 appears in every 4th position. (Figure 1(a) illustrates that shorthand Ucycles of $\Pi(n)$ do not necessarily contains a periodic symbol, and it is also obvious that a shorthand Ucycles of $\Pi(n)$ can contain at most one periodic symbol when $n \geq 3$.)

When a shorthand Ucycle for $\Pi(n)$ has n as its periodic symbol, then its remaining symbols are divided into blocks of length $n-1$. Furthermore, each one of these blocks must be a distinct permutation of $\langle n-1 \rangle$. Given this situation, the permutations of $\langle n-1 \rangle$ are called the *sub-permutations*. Figure 2 summarizes the substrings, permutations, binary representation, and sub-permutations of Figure 1(d).



substrings: $432, 321, 214, 142, \ldots$
permutations: $4321, 3214, 2143, 1423, \ldots$
binary representation: $001 \cdots$

periodic symbol: 4
sub-permutations: $321, 213, 132, 312, 123, 231$

Fig. 2. The direct construction for $n = 4$.

Given this terminology, there is a succinct description of the construction in [16]: The permutations for n are the sub-permutations for $n+1$. For example, the Ucycle for $n = 3$ is 321312, and so the permutations are $321, 213, 132, 312, 123, 231$. Notice that the permutations are identical to the sub-permutations found in Figure 2. For this reason, the construction in [16] can be described as the *direct construction*. The binary representation of the direct construction also has a nice description. If $x_1 x_2 \cdots x_{n!}$ is the binary string representation for n, then

$$001^{n-2}\ \overline{x}_1 001^{n-2}\ \overline{x}_2 \cdots 001^{n-2}\ \overline{x}_{n!} \tag{3}$$

is the binary string representation for $n+1$, where $\overline{x_i} = 1 - x_i$, and 1^{n-2} represents $n-2$ copies of 1. For example, the binary representation for $n = 2$ is 00, and so the binary representation for $n = 3$ and $n = 4$ are

$$00\ \bar{0}\ 00\ \bar{0}\ = 001001 \quad \text{and.}$$
$$001\ \bar{0}\ 001\ \bar{0}\ 001\ \bar{1}\ 001\ \bar{0}\ 001\ \bar{0}\ 001\ \bar{1} = 001100110010001100110010.$$

More generally, Ucycles are one of the basic concepts within the research area of *combinatorial generation*. For exceptional coverage of this area the reader is encouraged to consult the new fascicles of *The Art of Computer Programming* [10–12], which devotes over 400 pages to the subject.

1.4 Our New Approach

This paper expands upon [16] by providing two additional constructions and algorithms. The first construction leads to a bell-ringing method that is over 300 years-old and is folklore in the campanology community, although does not appear to be explicitly mentioned in print — it was brought to our attention by Holroyd [5]. This construction is illustrated in Figure 1(b).

The second construction uses the recently discovered shift Gray code of Williams [20], which generalizes *cool-lex order* [14] [15]. Aside from the intrinsic interest of having additional answers to Knuth's query, both of these constructions have two distinct advantages over the previous construction found in [16].

First, each construction contains fewer 1s in its binary representation. The bell-ringers algorithm is particularly advantageous in this regard since we show in Theorem 3 that the vast majority of bits are 0; asymptotically, the ratio of them is $(n-2)/n$.

Second, the resulting algorithms are faster in the following scenario: Suppose an application wants a shorthand Ucycle for the permutations of $\{1, 2, \ldots, n\}$. Since $n!$ is prohibitively large, the shorthand universal is given to the application n symbols at a time within an array of length n. Our new algorithm updates this array in average-case $O(1)$-time. In other words, it provides successive blocks of n symbols in *constant amortized time*. This is an improvement over the previous algorithm [16], which would have required $\Omega(n)$-time to obtain the next n symbols.

Sections 2 and 3 describe these new constructions, respectively. In both cases the strategy is to generate a list of $\Pi(n)$ that will become the sub-permutations for the shorthand Ucycle for $\Pi(n+1)$. Central to this strategy is a proof that the result is in fact a shorthand Ucycle for $\Pi(n+1)$. Towards this goal, we introduce the following notation and definition.

Definition 1. *Let $P = p_1, p_2, \ldots, p_n!$ be a list of permutations. By $\mathsf{R}(P)$ we denote the cyclic string $(n{+}1)p_1(n{+}1)p_2 \cdots (n{+}1)p_{n!}$. The list P is* recyclable *if $\mathsf{R}(P)$ is a shorthand universal cycle for the permutations of $\langle n{+}1\rangle$.*

It is important to note that not every order of $\Pi(n)$ is recyclable. This is illustrated by Figure 1(e). In fact, the two types of "errors" that appear in Figure 1(e) are the only two types of errors that can occur. This fact is illustrated in the following theorem.

Theorem 1. *A circular list of permutations $P = p_1, p_2, \ldots, p_{n!}$ is recyclable if and only if the following two conditions are satisfied.*

- *If α and β are two successive permutations, then $\alpha_i^{-1} - \beta_i^{-1} \leq 1$ for all $i \in \langle n\rangle$.*
- *If α and β are two successive permutations and α' and β' are also two successive permutations, then whenever there is a $j \in \langle n\rangle$ such that*

$$\alpha_{j+1} \cdots \alpha_n \beta_1 \cdots \beta_{j-1} = \alpha'_{j+1} \cdots \alpha'_n \beta'_1 \cdots \beta'_{j-1},$$

  then $\alpha = \alpha'$ (and hence $\beta = \beta'$).

Proof. Let $X = \mathsf{R}(P) = (n{+}1)p_1(n{+}1)p_2 \cdots (n{+}1)p_{n!}$. The first condition guarantees that any n successive symbols from X are distinct; that is, they are all n-permutations of $\langle n{+}1\rangle$. The condition says that no symbol in the successor moves left more than one position (it could move right).

The second condition will guarantee that all of the length n substrings of X are distinct. Since the length of X is obviously $(n+1)!$, this will finish the proof. Let ω be a substring of X of length n. Clearly, if ω does not contain $n{+}1$ then it is distinct, since in that case we must have $\omega = p_i$ for some i. If ω contains $n{+}1$, then $\omega = \alpha_{j+1} \cdots \alpha_n (n{+}1) \beta_1 \cdots \beta_{j-1}$ for some value of $j \in \langle n\rangle$ and successive permutations α and β. If ω is not distinct, then there would be some other two permutations α' and β' such that $\omega' = \alpha'_{j+1} \cdots \alpha'_n (n{+}1) \beta'_1 \cdots \beta'_{j-1}$. ◻

2 The Bell-Ringer Construction and 7-order

The bell-ringer construction is based on an ordering of the permutations of $\langle n\rangle$ that we call "seven order" and denote 7-order. It is a recursive method in which every permutation π of $\langle n{-}1\rangle$ is expanded into n permutations of $\langle n\rangle$ by inserting n in every possible way into π. These insertions are done in a peculiar order that is reminiscent the way the number seven is normally written, and is what inspires us to call it 7-order. That is, the first permutation is $n\pi$, the second is πn, and the remaining permutations are obtained by moving the n one position to the left until it is in the second position. We use the notation 7_n to denoted the 7-order of $\langle n\rangle$. Thus $7_2 = 21, 12$ and $7_3 = 321, 213, 231, 312, 123, 132$ and the first 4 permutations of 7_4 are $4321, 3214, 3241, 3421$.

Theorem 2. *The list 7_n is recyclable.*

Proof. We use Theorem 1. The first condition is clearly met by 7_n.

To verify the second condition, our strategy is to show that every symbol in α is determined by $\beta_1 \cdots \beta_{j-1}$ and $\alpha_{j+1} \cdots \alpha_n$. First note that either n is in $\alpha_{j+1} \cdots \alpha_n$ or it is in $\beta_1 \cdots \beta_{j-1}$. If n is in $\alpha_{j+1} \cdots \alpha_n$, then $\alpha = \beta_1 \cdots \beta_{j-1} x \alpha_{j+1} \cdots \alpha_n$, where $x = \langle n\rangle \setminus \{\beta_1, \ldots, \beta_{j-1}, \alpha_{j+1}, \ldots, \alpha_n\}$. If $n = \beta_k$ is in $\beta_1 \cdots \beta_{j-1}$, but $n \neq \beta_1$, then $\alpha = \beta_1 \cdots \beta_{k-1}\beta_{k+1} n \beta_{k+2} \cdots \beta_{j-1} x \alpha_{j+1} \cdots \alpha_n$. If $\beta_1 = n$ then the result follows by induction. $\square$

We define the *bell-ringer order* to be the order of permutations obtained by recycling 7_n.

2.1 The binary interpretation of bell-ringer order

In this subsection we infer the recursive 0/1 structure of $\mathsf{R}(7_n)$. Since the ns are n apart, every nth and $(n+1)$st bits are 0. (This is because $ns_1s_2 \cdots s_{n-1} \in \Pi(n)$ and $s_1s_2 \cdots s_{n-1}n \in \Pi(n)$ when $s_1s_2 \cdots s_{n-1}$ is a sub-permutation.) We thus may think of the 0/1 string as having the form

$$\Upsilon(n) \;=\; 00\ B(n)_1\ 00\ B(n)_2\ 00\ \cdots\ 00\ B(n)_{(n-1)!},$$

where $|B(n)_j| = n-2$. The initial 0 represents the σ_n that takes $n(n-1)\cdots 21$ to $(n-1)\cdots 21n$. We will now describe how to get $\Upsilon(n+1)$ from $\Upsilon(n)$.

We use several strings which are defined below. We omit n from the first two notations since it will be clear from context.

$$A = 1^{n-2}, \quad Z_k = 0^{n-k-2}10^{k-1}, \text{ and } \quad X_{n-1} = A00Z_100Z_200\cdots 00Z_{n-3}.$$

Note that $|A| = |Z_k| = n-2$ and $|X_n| = (n-1)(n-2)$. Given $\Upsilon(n)$, the 0/1 string for $n+1$ is

$$\Upsilon(n+1) \;=\; 00\ X_n\ 00\ 1B(n)_1\ 00\ X_n\ 00\ 1B(n)_2\ 00\ \cdots\ 00\ X_n\ 00\ 1B(n)_{(n-1)!}.$$

Here are the bitstrings for $\Upsilon(4)$ and $\Upsilon(5)$, where the initial case is $\Upsilon(3) = 00\ 1\ 00\ 1 = 00\ B(3)_1\ 00\ B(3)_2$. First, $\Upsilon(4) = 00\ 11\ 00\ 01\ 00\ 1\underline{1}\ 00\ 11\ 00\ 01\ 00\ 1\underline{1}$, which can also be written as $\Upsilon(4) = 00\ A\ 00\ Z_1\ 00\ 1B(3)_1\ 00\ A\ 00\ Z_1\ 00\ 1B(3)_2 = 00\ X_2\ 1B(3)_1\ 00\ X_2 00\ 1B(3)_2$. And $\Upsilon(5)$ is

$$\begin{array}{ll}
00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11} & (= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_1 = 00\ X_3\ 00\ 1B(4)_1)\\
00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{01} & (= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_2 = 00\ X_3\ 00\ 1B(4)_2)\\
00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11} & (= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_3 = 00\ X_3\ 00\ 1B(4)_3)\\
00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11} & (= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_4 = 00\ X_3\ 00\ 1B(4)_4)\\
00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{01} & (= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_5 = 00\ X_3\ 00\ 1B(4)_5)\\
00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11} & (= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_6 = 00\ X_3\ 00\ 1B(4)_6).
\end{array}$$

From these recursions we can determine the number of 1's.

Theorem 3. *The number of 1s in* $\Upsilon(n)$ *is* $2((n-1)!-1)$.

Proof. If c_n is the number of 1s then our recursive construction implies that $c_{n+1} = c_n + 2(n-2)(n-2)!$ with $c_3 = 2$. The solution of this recurrence relation is $2((n-1)!-1)$. □

Asymptotically, this means that the relative frequency of σ_{n-1} transitions is about $2/n$ and the relative frequency of σ_n transitions is asymptotically $(n-2)/n$. This answers an open question listed at the end of [16]; it asks whether there is a listing that uses more σ_ns than σ_{n-1}s. The bell-ringers listing clearly does so.

2.2 Iterative Rules

Now let us describe a rule for transforming one permutation of $\langle n-1\rangle$ in 7-order into the next. This is useful for efficiently generating 7-order and the bell-ringer shorthand Ucycle, as well as proving a *folklore* result from campanology.

Let $\boldsymbol{s} = s_1 s_2 \cdots s_{n-1} \in \Pi(n-1)$. Let h be the index such that $s_h = n-1$. If $h = 2$, then let i be the maximum value such that

$$s_1\ s_2\ \cdots\ s_i = s_1\ n-1\ n-2\ \cdots\ n-i+1$$

and let j be chosen to maximize the value of s_j such that $i+1 \leq j \leq n-1$. (Notice that j is undefined when $i = n-1$, and otherwise $j > i+1$.) The next permutation in 7-order is

$$\text{7}(\boldsymbol{s}) = \begin{cases} s_1 s_2 \cdots s_{h-2} s_h s_{h-1} s_{h+1} s_{h+2} \cdots s_{n-1} & \text{if } h > 2 \quad (4a) \\ s_2 s_3 \cdots s_i s_1 s_{i+2} s_{i+2} \cdots s_{j-2} s_j s_{j-1} s_{j+1} s_{j+2} \cdots s_{n-1} & \text{if } h{=}2 \text{ and } s_1{<}n-i \quad (4b) \\ s_2 s_3 \cdots s_{n-1} s_1 & \text{otherwise.} \quad (4c) \end{cases}$$

To see why (4) generates 7-order, one can simply compare each of the cases to the recursive definition of seven order:

- (4c) is performed when the largest symbol is in the first position of $\boldsymbol{s}$, and the result is the first symbol of $\boldsymbol{s}$ is moved into the last position;
- (4a) is performed when the largest symbol is in neither of the first two positions of $\boldsymbol{s}$, and the result is the largest symbol moves one position to the left;
- (4b) is performed when the largest symbol is in the second position, and the result is that symbols $s_2 s_3 \cdots s_i$ move one position to the left by recursion, and then the jth symbol moves one position to the left.

Algorithmically, successive iterations of (4) can be generated by a CAT algorithm when $\boldsymbol{s}$ is stored in array. That is, the 7-order for $\Pi(n-1)$ can be generated in $\mathrm{O}((n-1)!)$-time. To do this, one needs to simply keep track of the position of the largest symbol in a variable h. More precisely, given the value of h, (4c) is performed $1 \cdot (n-2)!$ times, and involves $O(n-1)$ work each time. Similarly, (4a) is performed $(n-2) \cdot (n-2)!$ times, and involves $O(1)$ work. Finally, (4b) is performed $1 \cdot (n-2)!$ times, and involves $O(n-1)$ work each time. Therefore, the overall implementation $\mathrm{O}((n-1)!)$-time since

$$n \cdot (n-2)! + n \cdot (n-2)! + (n-2) \cdot (n-2)! = 3n - 2 \cdot (n-2)! \leq 4 \cdot (n-1)!.$$

This proves the following theorem.

Theorem 4. *7-order for $\Pi(n-1)$ can be generated in $O((n-1)!)$-time when the permutations are stored in an array. Using the same algorithm, the bell-ringer shorthand Ucycle for $\Pi(n)$ can be generated in $O((n-1)!)$-time when successive blocks of length n are stored in an array (and the first element of the array is fixed at value n).*

The iterative rule in (4) also allows us to formally prove a beautiful result from the bell-ringing community. This result gives an iterative rule for directly generating the permutations from the bell-ringer Ucycle, and does not appear to have been formally proven in print [5].

Theorem 5. *Let $\boldsymbol{s} = s_1 s_2 \cdots s_n \in \Pi(n)$ be a permutation in the bell-ringer shorthand Ucycle for $\Pi(n)$, where m is the maximum value of s_1 and s_n, and k is the maximum value such that $n\ n-1\ \cdots k$ appears in the permutation as a circular substring. If $k-1 \leq m \leq n-1$, then the next permutation is $s_2 s_3 \cdots s_{n-1} s_1 s_n$. Otherwise, (if $m = n$ or $k-1 < m$) the next permutation is $s_2 s_3 \cdots s_n s_1$.*

Proof. Notice that $s_1 s_2 \cdots s_{n-1}$ is a substring of the bell-ringer Ucycle, and s_n is its missing symbol. Let u_n denote the next symbol after this substring. Recall from earlier discussions that $u_n \in \{s_1, s_n\}$, with $u_n = s_1$ implying the next permutation is $s_2 s_3 \cdots s_{n-1} s_1 s_n$, and $u_n = s_n$ implying the next permutation is $s_2 s_3 \cdots s_{n-1} s_1 s_n$.

If $s_1 = n$ or $s_n = n$, then $u_n = n$ due to the periodic symbol in the bell-ringer shorthand Ucycle. In both of these cases $m = n$, so the claim is verified in these cases. Assume that $m < n$ for the remainder of the proof.

If $s_1 = n-1$, then $u_n = s_1$ by (4c). Similarly, if $s_n = n-1$ then $u_n = s_1$ by (4a) In both of these cases $m = n-1$, so the claim is verified in these cases. Assume that $m < n-1$ for the remainder of the proof.

If $s_1 = k-1$ or $s_n = k-1$, then $u_n = s_1$ by (4b). In both of these cases $m = k-1$, so the claim is verified in these cases.

Finally, if $m < k-1$, then $u_n = s_n$ by (4b).

3 Cool-lex Construction

This section discusses the *cool-lex order* for $\Pi(n)$ [20]. Given a permutation, a *prefix left-shift* moves the symbol at a given position into the first position of the resulting permutation. This operation is denoted by $\lhd$ as follows

$$\lhd(s_1 s_2 \cdots s_n,\ j) = s_j s_1 s_2 \cdots s_{j-1} s_{j+1} s_{j+2} \cdots s_n.$$

A *prefix right-shift* is the inverse operation and involves moving the symbol in the first position into a given position. This operation is denoted by $\rhd$ as follows

$$\rhd(s_1 s_2 \cdots s_n,\ j) = s_2 s_3 \cdots s_{j-1} s_1 s_{j+1} s_{j+2} \cdots s_n.$$

Cool-lex order is generated by a single operation that is repeated over and over again. The operation was originally stated in terms of prefix left-shifts, but for the purposes of this document it will be useful to restate the definition in terms of prefix right-shifts. Both the *cool left-shift* and *cool right-shift* involve the notion of a *non-increasing prefix* which is defined below.

Definition 2 ($\urcorner\!\!\llcorner$). *If $\boldsymbol{s} = s_1 s_2 \ldots s_n$ is a string, then the* non-increasing prefix *of $\boldsymbol{s}$ is*

$$\urcorner\!\!\llcorner(\boldsymbol{s}) = s_1 s_2 \cdots s_j$$

where j is the maximum value such that $s_{j-1} \geq s_j$ for all $2 \leq j \leq \urcorner\!\!\llcorner(\boldsymbol{s})$.

For example, $\urcorner\!\llcorner(55432413) = 55432$ and $\urcorner\!\llcorner(33415312) = 33$.

Given a list of strings, the *reflected* list begins with the last string in the original list and ends with the first string in the original list. In particular, the reflected cool-lex order for $\Pi(n)$ is generated by repeated applications of the cool right-shift which is defined below.

Definition 3 (Cool right-shift ($\overrightarrow{\mathsf{cool}}$)). *Let* $\boldsymbol{s} = s_1 \ldots s_n$ *and* $\boldsymbol{s}' = s_2 s_3 \cdots s_n$ *and* $k' = |\urcorner\!\llcorner(\boldsymbol{s}')|$. *Then,*

$$\overrightarrow{\mathsf{cool}}(\boldsymbol{s}) = \begin{cases} \triangleright(\boldsymbol{s},\ k'+1) & \textit{if } k' \le n-2 \textit{ and } s_1 > s_{k'+1} & \text{(5a)} \\ \triangleright(\boldsymbol{s},\ k'+2) & \textit{if } k' \le n-2 \textit{ and } s_1 < s_{k'+1} & \text{(5b)} \\ \triangleright(\boldsymbol{s},\ n) & \textit{otherwise (if } k' \ge n-1\textit{)}. & \text{(5c)} \end{cases}$$

For example, $\overrightarrow{\mathsf{cool}}$ circularly generates the following list of $\Pi(3)$ which is the reflected list found in (9). These lists for $\Pi(n)$ are denoted by $\overrightarrow{\mathcal{C}}(n)$ as seen below

$$\overrightarrow{\mathcal{C}}(3) = 321\ 213\ 123\ 231\ 312\ 132. \tag{6}$$

3.1 Proof that cool-lex order of permutations is recyclable

This section proves that $\overrightarrow{\mathcal{C}}(n)$ becomes a universal cycle for $\Pi_n(n+1)$ after prefixing $n+1$ as a periodic symbol. For example, when $n = 3$

$$\begin{aligned} \overrightarrow{\mathcal{C}}(3) &= \ 321\ \ 213\ \ 123\ \ 231\ \ 312\ \ 132 \\ \mathsf{R}(\overrightarrow{\mathcal{C}}(3)) &= 4321421341234231431 24132 \end{aligned}$$

and $\mathsf{R}(\overrightarrow{\mathcal{C}}(3))$ is a universal cycle for the 3-permutations of $\langle 4 \rangle$. In general, if $\mathcal{L}$ is a list of $\Pi(n)$ then $\mathsf{R}(\mathcal{L})$ denotes the result of prefixing $n+1$ to every permutation in $\mathcal{L}$ and then concatenating the resulting strings together. Using this notation, the main result can be stated as follows.

Theorem 6 (Universal cycles using reflected cool-lex order).
The string $\mathsf{R}(\overrightarrow{\mathcal{C}}(n))$ *is a universal cycle for* $\Pi_n(n+1)$.

This result follows from Lemma 1 that is stated informally as follows: For every value of j satisfying $0 \le j \le n-1$, there are consecutive strings in $\overrightarrow{\mathcal{C}}(n)$ that contain the last j symbols of $\boldsymbol{p}$ as a prefix of one string, and the first $(n-1)-j$ symbols as a suffix of the previous string. To illustrate this lemma, let $n = 7$ and $\boldsymbol{p} = 254367 \in \Pi_6(7)$, and consider the following examples involving ordered pairs of strings, $\boldsymbol{s}$ followed by $\boldsymbol{t}$, within $\overrightarrow{\mathcal{C}}(7)$

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$\boldsymbol{s}$:	$\underline{254637}1$	$17\underline{25463}$	$137\underline{2546}$	$1637\underline{254}$	$34671\underline{25}$	$354671\underline{2}$	
$\boldsymbol{t}$:		$\underline{7}251463$	$\underline{37}12546$	$\underline{637}1254$	$\underline{4637}125$	$\underline{54637}12$	$1\underline{254637}$.

The extremal examples involve only one string, otherwise $\boldsymbol{t} = \overrightarrow{\mathsf{cool}}(\boldsymbol{s})$. Furthermore, the underlined string wrapping around from the end $\boldsymbol{s}$ to the beginning of $\boldsymbol{t}$ is equal to $\boldsymbol{p}$. For example, in the $j = 1$ column the underlined string contains five symbols from $\boldsymbol{s}$ and one symbol from $\boldsymbol{t}$. To see how these examples relate to universal cycles, consider $\mathsf{R}(\overrightarrow{\mathcal{C}}(7))$. Since the value of $n+1 = 8$ is appended to the front of every string within the original list $\overrightarrow{\mathcal{C}}(7)$, then the above cases ensure that the following substrings are contained within $\mathsf{R}(\overrightarrow{\mathcal{C}}(7))$

$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$8\underline{254637}$	$\underline{25463}8\underline{7}$	$\underline{2546}8\underline{37}$	$\underline{254}8\underline{637}$	$\underline{25}8\underline{4637}$	$\underline{2}8\underline{54637}$	$\underline{254637}8$.

In other words, the above substrings include every possible way of inserting 8 into $\boldsymbol{p} = 254367$ to obtain a 7-permutation of $\langle 8\rangle$ that contains 8. By repeating the same argument for every $\boldsymbol{p} \in \Pi_6(7)$, the result is that $\mathsf{R}(\overrightarrow{\mathcal{C}}(7))$ must contain every 7-permutation of $\langle 8\rangle$ that contains 8. Since the 7-permutations of $\langle 8\rangle$ that do not contain 8 can be obtained directly from the original strings in $\overrightarrow{\mathcal{C}}(7)$, then $\mathsf{R}(\overrightarrow{\mathcal{C}}(7))$ must be a universal cycle for $\Pi_7(8)$.

In general, the same argument shows how Theorem 6 follows from Lemma 1. The proof of Lemma 1 is somewhat tricky and involves several cases. For this reason, the (non-extremal) examples above are now restated along with additional information pertaining to the proof.

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$\boldsymbol{s}$:	$17\underline{25463}$	$137\underline{2546}$	$1637\underline{254}$	$34671\underline{25}$	$354671\underline{2}$
$\boldsymbol{t}$:	$\underline{7}251463$	$\underline{37}12546$	$\underline{637}1254$	$\underline{4637}125$	$\underline{54637}12$
	$k=1$	$k=1$	$k=2$	$k=1$	$k=2$
	case 2)	case 3a)	case 3a)	case 1)	case 1).

In all of these examples, $\boldsymbol{p} = 254367$ and $x = 1$ is its symbol missing from $\langle 7\rangle$. The underlined portion of $\boldsymbol{s}$ represents $p_{j+2}p_{j+3}\cdots p_n$. Similarly, the underlined portion of $\boldsymbol{t}$ represents $p_1p_2\cdots p_j$. Finally, k is the length of $\daleth(p_1p_2\cdots p_j)$ and is shown above.

Lemma 1. *Suppose $\boldsymbol{p} \in \Pi_{n-1}(n)$ and j is within $0 \le j \le n-1$. Then, there exists $\boldsymbol{s} = s_1s_2\cdots s_n \in \Pi(n)$ followed by $\boldsymbol{t} = t_1t_2\cdots t_n \in \Pi(n)$ in $\overrightarrow{\mathcal{C}}(\mathcal{L})$ such that*

$$s_{j+2}s_{j+3}\cdots s_nt_1t_2\cdots t_j = \boldsymbol{p}. \tag{7}$$

In other words, there exist two consecutive strings in $\overrightarrow{\mathcal{C}}(n)$ whose concatenation contains $\boldsymbol{p}$ as a substring, and moreover, the substring uses j symbols from the second string.

Proof. The proof of this lemma may be found in the appendix.

4 Concluding Remarks

This paper has provided the third explicit universal cycle for the $(n-1)$-permutations of $\langle n\rangle$, and for this reason it is natural to ask which universal cycle is "best". One way of differentiating the various solutions is to consider the sum of the binary string representation. For example, in the cycling application rotations of length n are somewhat simpler than rotations of length $n-1$, and so it would be desirable to minimize the sum of the binary representation.

Each of the explicit universal cycles also rely upon n being a periodic symbol. For this reason it is natural to ask if there is a construction that does not use a periodic symbol. (In general, it is not possible for a universal cycle of $\Pi_{n-1}(n)$ to have two or more periodic symbols.)

Finally, the explicit universal cycles also hint at a relationship between cyclic Gray codes of $\Pi(n-1)$ using prefix-shifts of length n and $n-1$ and the construction of universal cycles of $\Pi_{n-1}(n)$. Is it possible that these Gray codes (with an added condition) could always be extended to universal cycles using a periodic symbol?

5 Acknowledgement

The authors are indebted to Alexander E. Holroyd for telling us about the bell-ringers algorithm.

References

1. F. Chung, P. Diaconis, and R. Graham, *Universal cycles for combinatorial structures*, Discrete Mathematics, 110 (1992) 43–59.
2. P. F. Corbett, *Rotator Graphs: An Efficient Topology for Point-to-Point Multiprocessor Networks*, IEEE Transactions on Parallel and Distributed Systems, 3 (1992) 622–626
3. N.G. de Bruijn, *A Combinatorial Problem*, Koninkl. Nederl. Acad. Wetensch. Proc. Ser A, 49 (1946) 758–764.
4. R. Duckworth and Fabian Stedman, *Tintinnalogia*, 1668.
5. A. Holroyd, *Personal Communication* (02/2009).
6. B. Jackson, *Universal cycles of k-subsets and k-permutations*, Discrete Mathematics, 149 (1996) 123–129.
7. M. Jiang and F. Ruskey, *Determining the Hamilton-connectedness of certain vertex-transitive graphs*, Discrete Mathematics, 133 (1994) 159–170.
8. R. Johnson, *Universal cycles for permutations*, Discrete Mathematics, in press.
9. S. M. Johnson, *Generation of Permutations by Adjacent Transpositions*, Mathematics of Computation, 17 (1963) 282–285.
10. D.E. Knuth, *The Art of Computer Programming, Volume 4, Generating All Tuples and Permutations*, Fascicle 2, Addison-Wesley, 2005.
11. D.E. Knuth, *The Art of Computer Programming, Volume 4, Generating All Combinations and Partitions*, Fascicle 3, Addison-Wesley, 2005.
12. D.E. Knuth, *The Art of Computer Programming, Volume 4, Generating All Trees / History of Combinatorial Generation*, Fascicle 4, Addison-Wesley, 2005.
13. I. Pak and R. Radoičić, *Hamiltonian paths in Cayley Graphs*, Discrete Mathematics, 309 (2009) 5501–5508 .
14. F. Ruskey, and A. Williams, *The coolest way to generate combinations*, Discrete Mathematics, 309 (2009) 5305–5320. .
15. F. Ruskey, and A. Williams, *Generating Balanced Parentheses and Binary Trees by Prefix Shifts*, CATS '08: Fourteenth Computing: The Australasian Theory Symposium, Wollongong, Australia, CRPIT, ACS, 77 (2008).
16. F. Ruskey and A. Williams, *An explicit universal cycle for the $(n-1)$-permutations of an n-set*, ACM Transactions on Algorithms, in press.
17. R. Sedgewick, *Permutation Generation Methods*, Computing Surveys, 9 (1977) 137-164.
18. H. Steinhaus, *One Hundred Problems in Elementary Mathematics*, Pergamon Press, 1963.
19. H. Trotter, *Algorithm 115: Perm*, Comm. ACM 5 8 (1962) 434–435.
20. A. Williams, *Loopless Generation of Multiset Permutations Using a Constant Number of Variables by Prefix Shifts*, Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009, pp. 987–996.

A Additional Proofs

In this appendix we provide a proof of Lemma 1 which is restated below. We will need the idea of a cool-left shift.

Definition 4 (Cool left-shift ($\overleftarrow{\mathsf{cool}}$)). *Let $\boldsymbol{s} = s_1 \ldots s_n$ and $k = \urcorner\!\!\llcorner(\boldsymbol{s})$. Then,*

$$\overleftarrow{\mathsf{cool}}(\boldsymbol{s}) = \begin{cases} \lhd(\boldsymbol{s},\ k+1) & \text{if } k \leq n-2 \text{ and } s_{k+2} > s_k & \text{(8a)} \\ \lhd(\boldsymbol{s},\ k+2) & \text{if } k \leq n-2 \text{ and } s_{k+2} \leq s_k & \text{(8b)} \\ \lhd(\boldsymbol{s},\ n) & \text{otherwise (if } k \geq n-1). & \text{(8c)} \end{cases}$$

By repeating this operation every permutation of a given set is guaranteed to be visited exactly once [20]. (In fact, the operation works for the permutations of any multiset, and variations of this operation have been used to generate combinations [14] and balanced parentheses and binary trees [15].) For example, the following list of $\Pi(3)$ is circularly generated by repeated applications of $\overleftarrow{\mathsf{cool}}$

$$132\ 312\ 231\ 123\ 213\ 321. \tag{9}$$

Since the definition only involves the relative order of the symbols, then the set of symbols need not be $\{1, 2, \ldots, n\}$. One important aspect of $\overleftarrow{\mathsf{cool}}$ is that it does not change certain suffixes as stated by the following lemma.

Lemma 2 (Cool left-shift invariant). *Suppose $\boldsymbol{s} = s_1 s_2 \cdots s_n \in \Pi(n)$ and $\urcorner\!\!\llcorner(\boldsymbol{s}) = k$. Then,*

$$\overleftarrow{\mathsf{cool}}(s_1 s_2 \cdots s_n) = \overleftarrow{\mathsf{cool}}(s_1 s_2 \cdots s_{k+2}) \cdot s_{k+3} s_{k+4} \cdots s_n.$$

In other words, applying $\overleftarrow{\mathsf{cool}}$ to a string is equivalent to applying $\overleftarrow{\mathsf{cool}}$ to its first $k+2$ symbols and then appending its remaining $n-(k+2)$ symbols, where k represents the length of the non-increasing prefix.

Lemma 3. *Suppose $\boldsymbol{p} \in \Pi_{n-1}(n)$ and j is within $0 \leq j \leq n-1$. Then, there exists $\boldsymbol{s} = s_1 s_2 \cdots s_n \in \Pi(n)$ followed by $\boldsymbol{t} = t_1 t_2 \cdots t_n \in \Pi(n)$ in $\overrightarrow{\mathcal{C}}(\mathcal{L})$ such that*

$$s_{j+2} s_{j+3} \cdots s_n t_1 t_2 \cdots t_j = \boldsymbol{p}.$$

In other words, there exist two consecutive strings in $\overrightarrow{\mathcal{C}}(n)$ whose concatenation contains $\boldsymbol{p}$ as a substring, and moreover, the substring uses j symbols from the second string.

Proof. Let x be the symbol that is missing from $\{1, 2, \ldots, n\}$ within $\boldsymbol{p}$. The proof first considers the extreme values of j, namely $j = 0$ and $j = n-1$. According to (7), the substring $\boldsymbol{p}$ must be contained entirely within $\boldsymbol{s}$ or entirely within $\boldsymbol{t}$. That is,

$$\begin{aligned} s_{j+2} s_{j+3} \cdots s_n t_1 t_2 \cdots t_j &= s_2 s_3 \cdots s_n && \text{when } j = 0 \\ s_{j+2} s_{j+3} \cdots s_n t_1 t_2 \cdots t_j &= t_1 t_2 \cdots t_{n-1} && \text{when } j = n-1. \end{aligned}$$

For the $j = 0$ case, consider $\boldsymbol{s} = x \cdot \boldsymbol{p}$. Certainly $\boldsymbol{s} \in \Pi(n)$ since $\boldsymbol{p} \in \Pi_{n-1}(n)$. Furthermore,

$$s_2 s_3 \cdots s_n = \boldsymbol{p}$$

and so this case is proven by $\boldsymbol{s} = x \cdot \boldsymbol{p}$ (and $\boldsymbol{t} = \overrightarrow{\mathsf{cool}}(\boldsymbol{s})$). For the $j = n-1$ case, consider $\boldsymbol{t} = \boldsymbol{p} \cdot x$. Certainly $\boldsymbol{t} \in \Pi(n)$ since $\boldsymbol{p} \in \Pi_{n-1}(n)$. Furthermore,

$$t_1 t_2 \cdots t_{n-1} = \boldsymbol{p}$$

and so this case is proven by $\boldsymbol{t} = \boldsymbol{p} \cdot x$ (and $\boldsymbol{s} = \overleftarrow{\mathsf{cool}}(\boldsymbol{p})$).

In the remaining cases, the substring $\boldsymbol{p}$ must overlap the two strings $\boldsymbol{s}$ and $\boldsymbol{t}$. To facilitate the arguments let

$$\boldsymbol{p} = p_{j+2}p_{j+3}\cdots p_n p_1 p_2 \cdots p_j.$$

The indexing is chosen in this way so that it will match up with the indexing within the values found for $\boldsymbol{s}$ and $\boldsymbol{t}$. The result is now proven for each value of j by finding $\boldsymbol{s} = s_1 s_2 \cdots s_n \in \Pi(n)$ where $\boldsymbol{t} = t_1 t_2 \cdots t_n = \overrightarrow{\mathsf{cool}}(\boldsymbol{s})$ such that

$$s_{j+2}s_{j+3}\cdots s_n = p_{j+2}p_{j+3}\cdots p_n \text{ and } t_1 t_2 \cdots t_j = p_1 p_2 \cdots p_j. \tag{10}$$

Let $k = |\urcorner\!\llcorner(p_1 p_2 \cdots p_j)|$. Notice that $k \leq j$. The proof is now divided into three cases depending on the value of k. The third case is divided into two subcases.

Case 1: $k \leq j-2$. Consider

$$\begin{aligned}
\boldsymbol{s} &= \overleftarrow{\mathsf{cool}}(p_1 p_2 \cdots p_j) \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n \\
\boldsymbol{t} &= p_1 p_2 \cdots p_j \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n
\end{aligned}$$

Certainly $\boldsymbol{s}, \boldsymbol{t} \in \Pi(n)$ since $\boldsymbol{p} \in \Pi_{n-1}(n)$. Furthermore, $\boldsymbol{s}$ and $\boldsymbol{t}$ satisfy (10). Therefore, it remains only to prove that $\boldsymbol{t} = \overrightarrow{\mathsf{cool}}(\boldsymbol{s})$ which is done below

$$\begin{aligned}
&\overrightarrow{\mathsf{cool}}(\boldsymbol{s}) \\
&= \overrightarrow{\mathsf{cool}}(\overleftarrow{\mathsf{cool}}(p_1 p_2 \cdots p_j) \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n) \\
&= \overrightarrow{\mathsf{cool}}(\overleftarrow{\mathsf{cool}}(p_1 p_2 \cdots p_j \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n)) && \text{by Lemma 2 and } k \leq j-2 \\
&= p_1 p_2 \cdots p_j \cdot x \cdot p_{j+2}p_{j+2}\cdots p_n \\
&= \boldsymbol{t}.
\end{aligned}$$

Case 2: $k = j$. Consider

$$\begin{aligned}
\boldsymbol{s} &= x \cdot p_1 p_2 \cdots p_j \cdot p_{j+2}p_{j+3}\cdots p_n \\
\boldsymbol{t} &= \overrightarrow{\mathsf{cool}}(\boldsymbol{s}).
\end{aligned}$$

Certainly $\boldsymbol{s}, \boldsymbol{t} \in \Pi(n)$ since $\boldsymbol{p} \in \Pi_{n-1}(n)$. Furthermore, since $\boldsymbol{t} = \overrightarrow{\mathsf{cool}}(\boldsymbol{s})$ then $\boldsymbol{s}$ and $\boldsymbol{t}$ are consecutive within $\overrightarrow{\mathcal{C}}(n)$. Therefore, it remains only to prove that $\boldsymbol{s}$ and $\boldsymbol{t}$ satisfy (10). To prove this fact, notice that $k = j$ implies that $p_1 p_2 \cdots p_j$ is non-increasing. Therefore, when applying $\overrightarrow{\mathsf{cool}}$ to $\boldsymbol{s}$, the first symbol x will be right-shifted at least past all of the symbols in $p_1 p_2 \cdots p_j$. Therefore,

$$t_1 t_2 \cdots t_j = p_1 p_2 \cdots p_j$$

and so $\boldsymbol{s}$ and $\boldsymbol{t}$ satisfy (10).

Case 3a): $k = j-1$ and $x < p_{j-1}$. Consider

$$\begin{aligned}
\boldsymbol{s} &= x \cdot p_1 p_2 \cdots p_{j-1} p_j \cdot p_{j+2}p_{j+3}\cdots p_n \\
\boldsymbol{t} &= p_1 p_2 \cdots p_{j-1} p_j \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n.
\end{aligned}$$

Certainly $\boldsymbol{s}, \boldsymbol{t} \in \Pi(n)$ since $\boldsymbol{p} \in \Pi_{n-1}(n)$. Furthermore, $\boldsymbol{s}$ and $\boldsymbol{t}$ satisfy (10). Therefore, it remains only to prove that $\boldsymbol{t} = \overrightarrow{\mathsf{cool}}(\boldsymbol{s})$. In order to do this, first notice that $k = j-1$ implies that $p_1p_2\cdots p_{j-1}$ is non-increasing but $p_1p_2\cdots p_j$ is not. Therefore,

$$|\urcorner\!\llcorner(s_2s_3\cdots s_n)| = |\urcorner\!\llcorner(p_1p_2\cdots p_{j-1}p_j \cdot p_{j+2}p_{j+3}\cdots p_n)| = j-1. \tag{11}$$

Therefore,

$$\begin{aligned}
&\overrightarrow{\mathsf{cool}}(\boldsymbol{s}) \\
&= \overrightarrow{\mathsf{cool}}(x \cdot p_1p_2\cdots p_{j-1}p_j \cdot p_{j+2}p_{j+3}\cdots p_n) \\
&= \rhd(x \cdot p_1p_2\cdots p_{j-1}p_j \cdot p_{j+2}p_{j+3}\cdots p_n,\ j+1) && \text{by (11), } x < p_{j-1}\text{, and (5b)} \\
&= \overrightarrow{x \cdot p_1p_2\cdots p_{j-1}p_j} \cdot p_{j+2}p_{j+3}\cdots p_n \\
&= p_1p_2\cdots p_{j-1}p_j \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n \\
&= \boldsymbol{t}.
\end{aligned}$$

Case 3b): $k = j-1$ and $x > p_{j-1}$. Consider

$$\begin{aligned}
\boldsymbol{s} &= p_j \cdot p_1p_2\cdots p_{j-1} \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n \\
\boldsymbol{t} &= p_1p_2\cdots p_{j-1} \cdot p_j \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n.
\end{aligned}$$

Certainly $\boldsymbol{s}, \boldsymbol{t} \in \Pi(n)$ since $\boldsymbol{p} \in \Pi_{n-1}(n)$. Furthermore, $\boldsymbol{s}$ and $\boldsymbol{t}$ satisfy (10). Therefore, it remains only to prove that $\boldsymbol{t} = \overrightarrow{\mathsf{cool}}(\boldsymbol{s})$. In order to do this, first notice that $k = j-1$ implies that

$$p_j > p_{j-1}. \tag{12}$$

Second, $j = k$ also implies that $p_1p_2\cdots p_{j-1}$ is non-increasing, and $p_1p_2\cdots p_j$ is not since since $x > p_{j-1}$. Therefore,

$$|\urcorner\!\llcorner(s_2s_2\cdot s_n)| = |\urcorner\!\llcorner(p_1p_2\cdots p_{j-1} \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n)| = j-1. \tag{13}$$

Therefore,

$$\begin{aligned}
&\overrightarrow{\mathsf{cool}}(\boldsymbol{s}) \\
&= \overrightarrow{\mathsf{cool}}(p_j \cdot p_1p_2\cdots p_{j-1} \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n) \\
&= \rhd(p_j \cdot p_1p_2\cdots p_{j-1} \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n,\ j) && \text{by (13), (12), and (5a)} \\
&= \overrightarrow{p_j \cdot p_1p_2\cdots p_{j-1}} \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n \\
&= p_1p_2\cdots p_{j-1} \cdot p_j \cdot x \cdot p_{j+2}p_{j+3}\cdots p_n \\
&= \boldsymbol{t}.
\end{aligned}$$

The proof is now complete since all possible values of k and j satisfying $k \leq j$ have been considered.

B Implementations

This section provides a C-implementation for generating 7-order for the permutations of $\langle n\rangle$ and the associated bell-ringer shorthand universal cycle for the permutations of $\langle n+1\rangle$, as well as cool-lex order (using right-shifts) for the permutations of $\langle n\rangle$ and its associated shorthand universal cycle for the permutations of $\langle n+1 \rangle$. The program should be saved as `perms.c`. As presented, "perms n" will generate cool-lex order for the permutations of $\langle n \rangle$. Comment out "#define PERMS" to generate the shorthand universal cycle, and comment out "#define COOL" to generate 7-order (or the bell-ringer shorthand universal cycle). Regardless of these options, the program will run in O($n!$)-time (excluding the printing in the `visit()` routines).

```
#include<stdlib.h>
#include<stdio.h>

#define PERMS
#define COOL

void bellperms();
void coolperms();
void visit();
void usage_error();
int *perm;
int n;

void main(int argc, char *argv[]) {
   int x;
   if (argc == 1) {usage_error();}
   n = atoi(argv[1]);
   if (n < 1) {usage_error();}
   perm = (int *)malloc((n+1)*sizeof(int));
   for(x=0; x<=n; x++) {perm[x] = n+1-x;}
   #ifdef COOL
   coolperms();
   #else
   bellperms();
   #endif
}

void coolperms() {
   int x;
   int first;
   first = perm[1];
   while (1) {
      visit();
      first = perm[1];
      perm[1] = perm[2];
      for (x=2; x<n && perm[x]>=perm[x+1]; x++) {perm[x]=perm[x+1];}
      if (first == 1 && x==n) {perm[x] = first; return;}
      if (x<n && first<perm[x]) {perm[x]=perm[x+1]; x++;}
      perm[x] = first;
   }
}

void bellperms() {
   int h,i,j,x,temp;
   h = 1;
   while (1) {
      visit();
      if (h == 1) {
         for (x=1; x < n; x++) {perm[x] = perm[x+1];}
         perm[n] = n;
         h = n;
      } else if (h > 2) {
```

```
         perm[h] = perm[h-1];
         perm[h-1] = n;
         h--;
      } else {
         temp = perm[1];
         perm[1] = perm[2];
         for (i=2; i < n && perm[i+1] == n-i+1; i++) {
            perm[i] = n-i+1; // printf("i: %d  \n", i);
         }
         if (i == n) {
            return;
         }
         perm[i] = temp;
         if (perm[i] == n-i+1) {
            for (x=i; x < n; x++) {perm[x] = perm[x+1];}
            perm[n] = n-i+1;
         } else {
            for (j=i; perm[j] < n-i+1; j++);
            temp = perm[j-1];
            perm[j-1] = perm[j];
            perm[j] = temp;
         }
         h = 1;
      }
   }
}

void visit() {
   int x;
   #ifndef PERMS
   printf("%d ", perm[0]);
   #endif
   for (x=1; x<=n; x++) {
      printf("%d ", perm[x]);
   }
   #ifdef PERMS
   printf("\n");
   #endif
}

void usage_error()
{
   printf("usage: perms n for permutations of [n] (or shorthand Ucycle for permutationso of [n+1]).\n");
   exit(EXIT_FAILURE);
}
```