# A Hamilton Cycle in the k-Sided Pancake Network

B. Cameron   J. Sawada   A. Williams

Abstract. We present a Hamilton cycle in the k-sided pancake network and four combinatorial algorithms to traverse the cycle. The network's vertices are coloured permutations $\pi = p_1 p_2 \cdots p_n$, where each p_i has an associated colour in $\{0, 1, \ldots, k-1\}$. There is a directed edge (π_1, π_2) if π_2 can be obtained from π_1 by a "flip" of length j, which reverses the first j elements and increments their colour modulo k. Our particular cycle is created using a greedy min-flip strategy, and the average flip length of the edges we use is bounded by a constant. By reinterpreting the order recursively, we can generate successive coloured permutations in $O(1)$-amortized time, or each successive flip by a loop-free algorithm. We also show how to compute the successor of any coloured permutation in $O(n)$-time. Our greedy min-flip construction generalizes known Hamilton cycles for the pancake network (where $k = 1$) and the burnt pancake network (where $k = 2$). Interestingly, a greedy max-flip strategy works on the pancake and burnt pancake networks, but it does not work on the k-sided network when $k > 2$.

1 Introduction

Many readers will be familiar with the story of Harry Dweighter, the harried waiter who sorts stacks of pancakes for his customers. He does this by repeatedly grabbing some number of pancakes from the top of the stack, and flipping them over. For example, if the chef in the kitchen creates the stack 🥞, then Harry can sort it by flipping over all four pancakes 🥞, and then the top two 🥞.

This story came from the imagination of Jacob E. Goodman [8], who was inspired by sorting folded towels [23]. His original interest was an upper bound on the number of flips required to sort a stack of n pancakes. Despite its whimsical origins, the problem attacted interest from many mathematicians and computer scientists, including a young Bill Gates [11]. Eventually, it also found serious applications, including genomics [10].

A variation of the original story involves burnt pancakes. In this case, each pancake has two distinct sides: burnt and unburnt. When Harry flips the pancakes, the pancakes involved in the flip also turn over, and Harry wants to sort the pancakes so that the unburnt sides are facing up. For example, Harry could sort the stack 🥞 by flipping all four 🥞, then the top two 🥞, and the top one 🥞. Similar lines of research developed around this problem (e.g. [6], [10]). The physical model breaks down beyond two sides, however, many of the same applications do generalize to "k-sided pancakes".

1.1 Pancake Networks

Interconnection networks connect single processors, or groups of processors, together. In this context, the underlying graph is known as the network, and classic graph measurements (e.g. diameter, girth, connectivity) translate to different performance metrics. Two networks related to pancake flipping are in Figure 1.

The pancake network $\mathbb{G}(n)$ was introduced in the 1980s [1] and various measurements were established (e.g. [13]). Its vertex set is the set of permutations of $\{1, 2, \ldots, n\}$ in one-line notation, which is denoted $\mathbb{P}(n)$. For example, $\mathbb{P}(2) = \{12, 21\}$. There is an edge between permutations that differ by a *prefix-reversal of length ℓ*, which reverses the first ℓ symbols. For example, $(3421, 4321)$ is the $\ell = 2$ edge between 🥞 and 🥞. Goodman's original problem is finding the maximum shortest path length to the identity permutation. Since $\mathbb{G}(n)$ is vertex-transitive, this value is simply its diameter.

The burnt pancake network $\overline{\mathbb{G}}(n)$ was introduced in the 1990s [6]. Its vertex set is the set of signed permutations of $\{1, 2, \ldots, n\}$, which is denoted $\overline{\mathbb{P}}(n)$. For example, $\overline{\mathbb{P}}(2) = \{12, 1\overline{2}, \overline{1}2, \overline{1}\,\overline{2}, 21, 2\overline{1}, \overline{2}1, \overline{2}\,\overline{1}\}$ where overlines denote negative symbols. There is an edge between signed permutations that differ by a *sign-complementing prefix-reversal of length ℓ*, which reverses the order and sign of the first ℓ symbols. For example, $(\overline{2}\overline{1}34, 1234)$ is the $\ell = 2$ edge between 🥞 and 🥞.

The *k-sided pancake network* $\mathbb{G}_k(n)$ is a directed graph that was first studied in the 2000s [14]. Its vertex set is the set of k-coloured permutations of $\{1, 2, \ldots, n\}$ in one-line notation, which is denoted $\mathbb{P}_k(n)$. For example, $\mathbb{P}_3(2)$ is illustrated below, where colours the $0, 1, 2$ are denoted using superscripts, or in **black**, **red**, **blue**.

$$\mathbb{P}_3(2) = \{\mathbf{12}, \mathbf{1}\textcolor{red}{\mathbf{2}}, \mathbf{1}\textcolor{blue}{\mathbf{2}}, \textcolor{red}{\mathbf{1}}\mathbf{2}, \textcolor{red}{\mathbf{12}}, \textcolor{red}{\mathbf{1}}\textcolor{blue}{\mathbf{2}}, \textcolor{blue}{\mathbf{1}}\mathbf{2}, \textcolor{blue}{\mathbf{1}}\textcolor{red}{\mathbf{2}}, \textcolor{blue}{\mathbf{12}}, \mathbf{21}, \mathbf{2}\textcolor{red}{\mathbf{1}}, \mathbf{2}\textcolor{blue}{\mathbf{1}}, \textcolor{red}{\mathbf{2}}\mathbf{1}, \textcolor{red}{\mathbf{21}}, \textcolor{red}{\mathbf{2}}\textcolor{blue}{\mathbf{1}}, \textcolor{blue}{\mathbf{2}}\mathbf{1}, \textcolor{blue}{\mathbf{2}}\textcolor{red}{\mathbf{1}}, \textcolor{blue}{\mathbf{21}}\}$$
$$= \{1^0 2^0, 1^0 2^1, 1^0 2^2, 1^1 2^0, 1^1 2^1, 1^1 2^2, 1^2 2^0, 1^2 2^1, 1^2 2^2, 2^0 1^0, \ldots, 2^2 1^1, 2^2 1^2\}.$$

There is a directed edge from $\pi_1 \in \mathbb{P}_k(n)$ to $\pi_2 \in \mathbb{P}_k(n)$ if π_1 can be transformed into π_2 by a *colour-incrementing prefix-reversal of length ℓ*, which reverses the order and increments the colour modulo k of the first ℓ symbols. For example, $(\textcolor{red}{\mathbf{2}}\mathbf{134}, \mathbf{1234}) = (2^1 1^2 3^0 4^0, 1^0 2^2 3^0 4^0)$ is a directed $\ell = 2$ edge.

(a) The pancake network $\mathbb{G}(4)$.   (b) The 3-sided pancake network $\mathbb{G}_3(2)$.
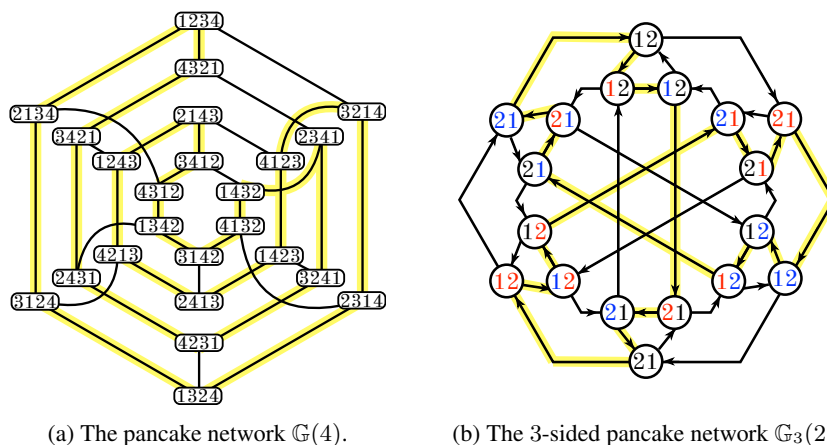
Fig. 1: Hamilton cycles in a pancake network and a coloured pancake network. The highlighted cycles start at $12 \cdots n$ (or $1^0 2^0 \cdots n^0$) and are constructed by the greedy min-flip strategy. The colours $0, 1, 2$ in (b) correspond to black, red, and blue.

Notice that $\mathbb{G}(n)$ and $\mathbb{G}_1(n)$ are isomorphic, while $\overline{\mathbb{G}}(n)$ and $\mathbb{G}_2(n)$ are isomorphic, so long as we view each undirected edge as two opposing directed edges. It also bears mentioning that $\mathbb{G}_k(n)$ is a (connected) directed Cayley graph, and its underlying group is the wreath product of the cyclic group of order k and the symmetric group of order n. Thus, $\mathbb{G}_k(n)$ is expected to have a directed Hamilton cycle by the Lovász conjecture.

When the context is clear, or the distinction is not necessary, we use the term *flip* for prefix-reversal (when $k = 1$), sign-complementing prefix-reversal (when $k = 2$), and colour-incrementing prefix-reversal (when $k > 2$).

1.2   (Greedy) Hamilton Cycles

In this paper, we are not interested in shortest paths in pancake networks, but rather Hamilton cycles. There are myriad ways that researchers attempt to build Hamilton cycles in highly-symmetric graphs, and the greedy approach is perhaps the simplest (see Williams [25]). This approach initializes a path at a specific vertex, then repeatedly extends the path by a single edge. More specifically, it uses the highest priority edge (according to some criteria) that leads to a vertex that is not on the path. The path stops growing when the current vertex is only adjacent to vertices on the path. A Hamilton cycle has been found if every vertex is on the path, and there is an edge from the final vertex to the first vertex. Despite its simplicity, the approach is known to work on many well-known graphs [25].

We show that the greedy approach works for the coloured pancake network $\mathbb{G}_k(n)$ when we prioritize the edges by shortest flip length. More specifically, we start a path at $1^0 2^0 \cdots n^0 \in \mathbb{P}_k(n)$, then repeatedly extend it to a new vertex along the edge that corresponds to the shortest colour-incrementing prefix-reversal. We refer to this as the *greedy min-flip construction*, denoted $\mathsf{GreedyMin}_k(n)$, and it is illustrated in Figure 1. When $k = 1$, the cycle that we create is identical to the one given by Zaks [26], and when $k = 2$, our cycle in the burnt pancake network was previously produced by Suzuki, N. Sawada, and Kaneko [15]; however, both of these papers describe their cycles recursively. The greedy construction of the cycles in the pancake and burnt pancake networks was previously given by J. Sawada and Williams [9, 19].

1.3   Combinatorial Generation

Ostensibly, the primary contribution of this paper is the Hamiltonicity of k-sided pancake networks. However, the authors' primary motivation was not in finding a Hamilton cycle, but rather in investigating its contributions to combinatorial generation. *Combinatorial generation* is the research area devoted to the efficient and clever generation of combinatorial objects. By *efficient* we mean that successive objects can be generated in amortized $O(1)$-time or worst-case $O(1)$-time, regardless of their size. The former is known as *constant amortized time (CAT)*, while the latter is known as *loop-free*. By *clever* we mean that non-lexicographic orders are often desirable. When describing these alternate orders, the authors make liberal use of the term *Gray code* — in reference to the eponymous binary reflected Gray code patented by Frank Gray [12]) — and we refer to our Hamilton cycle as a *colour-incrementing prefix-reversal Gray code* for coloured permutations. Informally, it is a *flip Gray code*.

There are dozens of publications on the efficient generation of permutation Gray codes. In fact, comprehensive discussions on this topic date back to Sedgewick's survey in 1977 [21], with more modern coverage in Volume 4 of Knuth's *The Art of Computer Programming* [16]. However, to our knowledge, there are no published Gray codes for coloured permutations. This is surprising as the combinatorial [4, 5, 7, 17, 18] and algebraic [2, 3, 22] properties of coloured permutations have been of considerable interest. Work on the latter is due to the group theoretic interpretation of $\mathbb{P}_k(n)$ as the wreath product of the cyclic and symmetric group, $\mathbb{Z}_k \wr S_n$. We find our new Grady code of interest for two additional reasons.

1. Other greedy approaches for generating $\mathbb{P}(n)$ do not seem to generalize to $\mathbb{P}_k(n)$.
2. Flips are natural and efficient operations in certain contexts.

To expand on the first point, consider the Steinhaus-Johnson-Trotter (SJT) order of permutations, which dates back to the 1600s [16]. In this order, successive permutations differ by an *adjacent-transition* (or *swap*) meaning that adjacent values in the permutations change place. In other words, the order for $\mathbb{P}(n)$ traces a Hamilton path in the permutohedron of order n. For example, SJT order for $n = 4$ appears below

$$123\underline{4}, 12\underline{4}3, \underline{14}2\underline{3}, 41\underline{23}, \underline{4}132, 1\underline{43}2, 13\underline{42}, \underline{13}24, 31\underline{24}, 3\underline{14}2, \underline{34}12, 43\mathbf{12},$$
$$\underline{4}321, 3\underline{42}1, 32\underline{41}, \mathbf{3}214, 23\underline{14}, 2\underline{34}1, \underline{24}31, 42\underline{31}, \underline{42}13, 2\underline{41}3, 21\underline{43}, \mathbf{2}134.$$

The symbols that are swapped to create the next permutation are underlined, and the larger value is in bold. The latter demarcation shows the order's underlying greedy priorities: Swap the largest value. For example, consider the fourth permutation in the list, 4123. The largest value 4 cannot be swapped to the left (since it is in the leftmost position) or the right (since 1423 is already in the order), so the next option is to consider 3, and it can only be swapped to the left, which gives the fifth permutation 4132. If this description is perhaps too brief, then we refer the reader to [25].

Now consider greedy generalizations of SJT to signed permutations. The most natural generalization would involve the use of *sign-complementing adjacent-transpositions* which swap and complement the sign of two adjacent values. Unfortunately, any approach using these operations is doomed to fail. This is because the operation does not change the parity of positive and negative values. The authors experimented with other types of signed swaps — complementing the leftmost or rightmost value in the swap, or the larger or small value in the swap — without success.

More surprising is the fact that our greedy min-flip strategy works for coloured permutations, but the analogous max-flip strategy does not. For example, the max-flip strategy creates the following path in $\mathbb{G}_3(2)$ before getting stuck.

$$1^0 2^0,\ 2^1 1^1,\ 1^2 2^2,\ 2^0 1^0,\ 1^1 2^1,\ 2^2 1^2,\ 2^0 1^2,\ 1^0 2^1,\ 2^2 1^1,\ 1^2 2^0,\ 2^1 1^0,\ 1^1 2^2,\ \lightning$$

The issue is that the neighbors of last coloured permutation in the path are already on the path. More specifically, a flip of length one transforms $1^1 2^2$ into $1^2 2^2$, and a flip of length two transforms $1^1 2^2$ into $2^0 1^2$, both of which appear earlier. The failure of the max-flip strategy on coloured permutations is surprising due to the fact that it works for both permutations and signed-permutations [9, 19].

To expand on the second point, note that the time required to flip a prefix is proportional its length. In particular, if a permutation over $\{1, 2, \ldots, n\}$ is stored in an array or linked list of length n, then it takes $O(m)$-time to flip a prefix of length m^1. Our min-flip strategy ensures that the shortest possible flips are used. In fact, the average flip length used in our Gray codes is bounded by $e = 2.71828\cdots$ when $k = 1$, and the average is even smaller for $k > 1$.

We also note that flips can be the most efficient operation in certain situations. For example, consider a brute force approach to the undirected travelling salesman problem, wherein every Hamilton path of the n cities is represented by a permutation in $\mathbb{P}(n)$. If we iterate over the permutations using a prefix-reversal Gray code, then successive Hamilton paths differ in a single edge. For example, the edges in 12345678 and 43215678 are identical, except that the former includes $(4, 5)$ while the latter includes $(1, 5)$. Thus, the cost of each Hamilton cycle can be updated from permutation to permutation using one addition and subtraction. More generally, flip Gray codes are the most efficient choice when the cost (or value) of each permutation depends on its unordered pairs of adjacent symbols. Similarly, our generalization will be the most efficient choice when the cost (or value) of each coloured permutation depends on its unordered pairs of adjacent symbols *and* the minimum distance between their colours.

1.4 New Results

We present a flip Gray code for $\mathbb{P}_k(n)$ that corresponds to a Hamilton cycle in the k-sided pancake network. We present the following four different combinatorial algorithms for traversing the Hamilton cycle, each having unique and interesting properties:

1. A greedy algorithm that is easy to describe, but requires an exponential amount of memory.
2. A recursive algorithm, that reveals the structure of the listing and can be implemented in $O(1)$-amortized time.
3. A simple successor rule approach that allows the cycle to start from any vertex (coloured permutation) and takes on average $O(1)$-time amortized over the entire listing.
4. A loop-free algorithm to generate the flip sequence iteratively.

Before we present these algorithms in Section 3, we first present some notation in Section 2. We conclude with a summary and related work in Section 4.

2 Notation

Let $\pi = p_1 p_2 \cdots p_n$ be a coloured permutation where each $p_i = v_i^{c_i}$ has value $v_i \in \{1, 2, \ldots, n\}$ and colour $c_i \in \{0, 1, \ldots, k - 1\}$. Recall that $\mathbb{P}_k(n)$ denotes the set of k-coloured permutations of $\{1, 2, \ldots, n\}$. Observe that $\mathbb{P}_1(n)$ corresponds to regular permutations and $\mathbb{P}_2(n)$ corresponds to signed permutations. For the remainder of this paper, it is assumed that all permutations are coloured.

---

[1] Some unusual data structures can support flips of any lengths in constant-time [24].

As mentioned earlier, a flip of a permutation π, denoted $\mathsf{flip}_i(\pi)$, applies a prefix-reversal of length i on π that also increments the colour of the flipped elements by 1 (modulo k). As an example for $k = 3$:

$$\mathsf{flip}_4(7^0 1^2 6^1 5^0 3^1 4^1 2^1) = \mathbf{5^1 6^2 1^0 7^1} 3^1 4^1 2^1.$$

A *pre-perm* is any prefix of a permutation in $\mathbb{P}_k(n)$, i.e. $\mathbf{p} = p_1 p_2 \cdots p_j$ is a pre-perm if there exist $p_{j+1}, \ldots p_n$ such that $p_1 p_2 \cdots p_n$ is a permutation. Note that if $j = n$, then the pre-perm is a permutation. Let $\mathbf{p} = p_1 p_2 \cdots p_j$ be an arbitrary pre-perm for given a k. For a given element $p_i = v_i^c$, let $p_i^{+s} = v_i^{(c_i + s) \pmod{k}}$. For $0 \le i < k$, let $\mathbf{p}^{+\mathbf{i}}$ denote $p_1^{+i} p_2^{+i} \cdots p_j^{+i}$, i.e. $\mathbf{p}$ with the colour of each symbol p_i incremented by i modulo k. Note, $\mathbf{p}^{+\mathbf{0}} = \mathbf{p}$. Furthermore, let $\rho(\mathbf{p}) = \mathbf{p}^{+(\mathbf{k}-\mathbf{1})} \cdot \mathbf{p}^{+(\mathbf{k}-\mathbf{2})} \cdots \mathbf{p}^{+\mathbf{0}} = r_1 r_2 \cdots r_m$ be a circular string of length $m = kj$ where $\cdot$ denotes the concatenation of symbols. Let $\rho(\mathbf{p})_\mathbf{i}$ denote the length $j-1$ subword ending with r_{i-1}.

Example 1 Consider a pre-perm $\mathbf{p} = 1^0 2^0 3^2$ where $j = 3$ and $k = 4$. Then

$$\rho(\mathbf{p}) = 1^3 2^3 3^1 \cdot 1^2 2^2 3^0 \cdot 1^1 2^1 3^3 \cdot 1^0 2^0 3^2 \quad \text{and} \quad \rho(\mathbf{p})_\mathbf{2} = 3^0 1^3.$$

For any pre-perm $\mathbf{p} = p_1 p_2 \cdots p_j$, let $\overleftarrow{\mathbf{p}}$ denote the reverse of $\mathbf{p}$. i.e. $\overleftarrow{\mathbf{p}} = p_j p_{j-1} \cdots p_2 p_1$. Note that $\overleftarrow{\mathbf{p}}$ is not equivalent to applying a flip of length j to $\mathbf{p}$ when $k > 1$ as the colours of each symbol do not change in $\overleftarrow{\mathbf{p}}$. For the remainder of this paper we will use $\mathbf{p}$ to denote a pre-perm, and when it is clear we will use π to denote a permutation.

3 Constructions of a Cyclic Flip Gray code for $\mathbb{P}_k(n)$

In this section we present four different combinatorial algorithms for generating the same cyclic flip Gray code for $\mathbb{P}_k(n)$. We begin by studying the listing generated by a greedy min-flip algorithm. By studying the underlying recursive structure of the greedy listing, we provide a recursive description of the flip-sequence and prove it is equivalent to the flip-sequence generated by the greedy algorithm. This proves that the greedy algorithm generates all permutations in $\mathbb{P}_k(n)$. We then present a successor-rule that determines the successor of a given permutation in the greedy min-flip listing in expected $O(1)$-time. We conclude by showing how the flip-sequence can be generated via a loop-free algorithm.

3.1 Greedy Algorithm

Recall that $\mathsf{GreedyMin}_k(n)$ denotes the greedy algorithm on $\mathbb{P}_k(n)$ that starts at permutation $1^0 2^0 \cdots n^0$ and prioritizes the neighbors of each permutation in the k-sided pancake network by increasing flip length.

Example 2 The following listing (left of the vertical bar) denotes the output of $\mathsf{GreedyMin}_3(3)$ (read top to bottom, then left to right), where black, red and blue correspond to the colours 0,1 and 2 respectively. This listing is exhaustive and cyclic; the last permutation differs from the first permutation by a flip of length $n = 3$. To the right of the vertical line is the flip length required to get from the permutation in that position to its successor.

```
123 312 231 123 312 231 123 312 231 | 1 1 1 1 1 1 1 1 1
123 312 231 123 312 231 123 312 231 | 1 1 1 1 1 1 1 1 1
123 312 231 123 312 231 123 312 231 | 2 2 2 2 2 2 2 2 2
213 132 321 213 132 321 213 132 321 | 1 1 1 1 1 1 1 1 1
213 132 321 213 132 321 213 132 321 | 1 1 1 1 1 1 1 1 1
213 132 321 213 132 321 213 132 321 | 2 2 2 2 2 2 2 2 2
123 312 231 123 312 231 123 312 231 | 1 1 1 1 1 1 1 1 1
123 312 231 123 312 231 123 312 231 | 1 1 1 1 1 1 1 1 1
123 312 231 123 312 231 123 312 231 | 2 2 2 2 2 2 2 2 2
213 132 321 213 132 321 213 132 321 | 1 1 1 1 1 1 1 1 1
213 132 321 213 132 321 213 132 321 | 1 1 1 1 1 1 1 1 1
213 132 321 213 132 321 213 132 321 | 2 2 2 2 2 2 2 2 2
123 312 231 123 312 231 123 312 231 | 1 1 1 1 1 1 1 1 1
123 312 231 123 312 231 123 312 231 | 1 1 1 1 1 1 1 1 1
123 312 231 123 312 231 123 312 231 | 2 2 2 2 2 2 2 2 2
213 132 321 213 132 321 213 132 321 | 1 1 1 1 1 1 1 1 1
213 132 321 213 132 321 213 132 321 | 1 1 1 1 1 1 1 1 1
213 132 321 213 132 321 213 132 321 | 3 3 3 3 3 3 3 3 3
```

Observe that each column of permutations ends with the same element. Furthermore, the last permutation in each column is a subword of the cyclic word **321321321**.

Unlike the max-flip approach, we will prove that $\mathsf{GreedyMin}_k(n)$ exhaustively generates all permutations in $\mathbb{P}_k(n)$ for all $n, k \geq 1$. We also show that the last permutation in the listing differs by a flip of length n from the first permutation, so the listing is a cyclic flip Gray code. To prove this result, we study the underlying recursive structure of the resulting listings and examine the flip-sequences.

3.2 Recursive Construction

By applying the two observations made following the listing of $\mathsf{GreedyMin}_3(3)$ in Example 2, we arrive at the following recursive definition for a given pre-perm $\mathbf{p}$ of a permutation in $\mathbb{P}_k(n)$:

$$\mathbf{Rec}_k(\mathbf{p}) = \mathbf{Rec}_k(\rho(\mathbf{p})_\mathbf{m}) \cdot r_m, \ \mathbf{Rec}_k(\rho(\mathbf{p})_{\mathbf{m-1}}) \cdot r_{m-1}, \ldots, \ \mathbf{Rec}_k(\rho(\mathbf{p})_\mathbf{1}) \cdot r_1, \quad (1)$$

where $\mathbf{Rec}_k(p_1) = p_1^{+0}, p_1^{+1}, p_1^{+2}, \ldots, p_1^{+(k-1)}$. We will prove that $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$ generates the same (exhaustive) listing of permutations as $\mathsf{GreedyMin}_k(n)$.

Lemma 1. *Let $\mathbf{p} = p_1 p_2 p_3 \cdots p_j$ be a pre-perm of a permutation in $\mathbb{P}_k(n)$ for some $n \geq j$. Then the first and last pre-perms in the listing $\mathbf{Rec}_k(\mathbf{p})$ are $\mathbf{p}$ and $\overleftarrow{\mathbf{p}^{+(\mathbf{k-1})}}$, respectively.*

Proof. The proof proceeds by induction on j. When $j = 1$, we have $\mathbf{p} = \overleftarrow{\mathbf{p}} = p_1$, so $\mathbf{Rec}_k(\mathbf{p}) = \mathbf{p}, \mathbf{p^{+1}}, \mathbf{p^{+2}}, \ldots, \mathbf{p^{+(k-1)}}$. Since $\mathbf{p^{+(k-1)}} = \overleftarrow{\mathbf{p^{+(k-1)}}}$ the claim holds. Now for $1 \leq j < n$ and any pre-perm $\mathbf{p} = p_1 p_2 \cdots p_j$ of a permutation in $\mathbb{P}_k(n)$, suppose that the first and last pre-perms in $\mathbf{Rec}_k(\mathbf{p})$ are $\mathbf{p}$ and $\overleftarrow{\mathbf{p^{+(k-1)}}}$ respectively. Let $\mathbf{p} = p_1 p_2 \cdots p_j p_{j+1}$ be a pre-perm of a permutation in $\mathbb{P}_k(n)$. By definition, the first pre-perm of $\mathbf{Rec}_k(\mathbf{p})$ is the first pre-perm of $\mathbf{Rec}_k(\rho(\mathbf{p})_\mathbf{m}) \cdot r_m$ where $m = (j+1)k$. By definition of $\rho(\mathbf{p})$ and $\rho(\mathbf{p})_\mathbf{m}$, it is clear that $r_m = p_{j+1}$ and $\rho(\mathbf{p})_\mathbf{m} = p_1 p_2 \cdots p_{j-1} p_j$. Applying the inductive hypothesis, the first pre-perm of $\mathbf{Rec}_k(p_1 p_2 \cdots p_{j-1} p_j)$ is $p_1 p_2 \cdots p_{j-1} p_j$. Therefore, the first pre-perm of $\mathbf{Rec}_k(\mathbf{p})$ is $p_1 p_2 \cdots p_{j-1} p_j \cdot p_{j+1} = \mathbf{p}$. Similarly, the last pre-perm of $\mathbf{Rec}_k(\mathbf{p})$ is the last pre-perm of $\mathbf{Rec}_k(\rho(\mathbf{p})_\mathbf{1}) \cdot r_1$. Now, $r_1 = p_1^{+(k-1)}$ and $\rho(\mathbf{p})_\mathbf{1} = p_2 p_3 \cdots p_n p_{n+1}$ and, by the inductive hypothesis, the last pre-perm in $\mathbf{Rec}_k(\rho(\mathbf{p})_\mathbf{1})$ is $p_{n+1}^{+(k-1)} p_n^{+(k-1)} \cdots p_2^{+(k-1)}$. Therefore, the last pre-perm of $\mathbf{Rec}_k(\mathbf{p})$ is $\overleftarrow{\mathbf{p^{+(k-1)}}}$. $\square$

Define the sequence σ_n^k recursively as

$$\sigma_n^k = \begin{cases} 1^{k-1} & \text{if } n = 1 \\ (\sigma_{n-1}^k, n)^{kn-1}, \sigma_{n-1}^k & \text{if } n > 1. \end{cases} \tag{2}$$

We will show that σ_n^k is the flip-sequence for both $\mathbf{Rec}_k(\mathbf{p})$ and $\mathsf{GreedyMin}_k(n)$. This flip-sequence is a straightforward generalization of the recurrences for non-coloured permutations [26] and signed permutations [19].

Lemma 2. *For $n \geq 1$, $k \geq 1$, and $\pi \in \mathbb{P}_k(n)$, the flip-sequence for $\mathbf{Rec}_k(\pi)$ is σ_n^k.*

Proof. By induction on n. In the base case $\mathbf{Rec}_k(p_1) = p_1, p_1^{+1}, p_1^{+2}, \ldots, p_1^{+(k-1)}$ and the flip-sequence is $\sigma_1^k = 1^{k-1}$. For $n \geq 1$ assume that the sequence of flips used to create $\mathbf{Rec}_k(p_1 p_2 p_3 \cdots p_n)$ is given by σ_n^k. Consider $\mathbf{Rec}_k(\pi)$ where $\pi = p_1 p_2 p_3 \cdots p_{n+1} \in \mathbb{P}_k(n+1)$. By the inductive hypothesis, it suffices to show that the last permutation of $\mathbf{Rec}_k(\rho(\pi)_i) \cdot r_i$ and the first permutation of $\mathbf{Rec}_k(\rho(\pi)_{i-1}) \cdot r_{i-1}$ differ by a flip of length $n + 1$ for $i = 2, 3, \ldots, m$ $(= k(n+1))$. By definition, $\rho(\pi)_i = r_{i-n} r_{i-(n-1)} \cdots r_{i-2} r_{i-1}$ where the indices are taken modulo m. Therefore, by Lemma 1, the last permutation in $\mathbf{Rec}_k(\rho(\pi)_i)$ is $(r_{i-1} r_{i-2} \cdots r_{i-(n-1)} r_{i-n})^{+(k-1)}$. Applying a flip of length $n + 1$ to $\mathbf{Rec}_k(\rho(\pi)_i) \cdot r_i$ yields

$$r_i^{+1} r_{i-n} r_{i-(n-1)} \cdots r_{i-2} r_{i-1}. \tag{3}$$

By Lemma 1, the first permutation of $\mathbf{Rec}_k(\rho(\pi)_{i-1})$ is $r_{i-(n+1)} r_{i-n} \cdots r_{i-3} r_{i-2}$. By the definition of $\rho(\pi)$, it follows that $r_{i-(n+1)} = r_i^{+1}$. Thus, from (3), it follows that $\mathbf{Rec}_k(\rho(\pi)_i) \cdot r_i$ and the first permutation of $\mathbf{Rec}_k(\rho(\pi)_{i-1}) \cdot r_{i-1}$ differ by a flip of length $n + 1$. By applying the inductive hypothesis, the flip-sequence for $\mathbf{Rec}_k(\pi)$ is $(\sigma_n^k, n+1)^{k(n+1)-1}, \sigma_n^k$ which is exactly σ_{n+1}^k. $\square$

Theorem 1. *For $n \geq 1$, $k \geq 1$, and $\pi \in \mathbb{P}_k(n)$, $\mathbf{Rec}_k(\pi)$ is a cyclic flip Gray code for $\mathbb{P}_k(n)$, where the first and last permutations differ by a flip of length n.*

Proof. From Lemma 2, the flip-sequence for $\mathbf{Rec}_k(\pi)$ is given by σ_n^k. Inductively, it is easy to see that the length of the flip-sequence σ_n^k is $k^n n! - 1$ and that each permutation of $\mathbf{Rec}_k(\pi)$ is unique. Thus, each of the $k^n n!$ permutations must be listed exactly once and, from Lemma 1, the first and last permutations of the listing differ by a flip of length n, making $\mathbf{Rec}_k(\pi)$ a cyclic flip Gray code for permutations. $\qquad\square$

Lemma 3. *For $n \geq 1$ and $k \geq 1$, the flip-sequence for* $\mathsf{GreedyMin}_k(n)$ *is σ_n^k.*

Proof. By contradiction. Suppose the sequence of flips used by $\mathsf{GreedyMin}_k(n)$ differs from σ_n^k and let j be the smallest value such that the j-th flip used to create $\mathsf{GreedyMin}_k(n)$ differs from the j-th value of σ_n^k. Let these flip lengths be s and t respectively. Since $\mathsf{GreedyMin}_k(n)$ follows a greedy minimum-flip strategy and because σ_n^k produces a valid flip Gray code for permutations by Theorem 1 where no permutation is repeated, it must be that $s < t$. Let $\pi = p_1 p_2 p_3 \cdots p_n$ denote the j-th permutation in the listing $\mathsf{GreedyMin}_k(n)$, i.e. the permutation immediately prior to the j-th flip. Since σ_n^k is the flip-sequence for $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$ by Lemma 2, from the recursive definition it follows inductively that all other permutations with suffix $p_t p_{t+1} \cdots p_n$ appear before π in $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$, since no permutations are repeated by Theorem 1. Since σ_n^k and the sequence of flips used by $\mathsf{GreedyMin}_k(n)$ agree until the j-th value, all other permutations with suffix $p_t p_{t+1} \cdots p_n$ appear before π in $\mathsf{GreedyMin}_k(n)$. Therefore, flipping π by a flip of length $s < t$ results in a permutation already visited in $\mathsf{GreedyMin}_k(n)$ before index j contradicting the fact that $\mathsf{GreedyMin}_k(n)$ produces a list of permutations without repetition. $\qquad\square$

By definition, $\mathsf{GreedyMin}_k(n)$ starts with the permutation $1^0 2^0 \cdots n^0$ and by Lemma 1, $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$ also starts with $1^0 2^0 \cdots n^0$. Since they are each created by the same flip-sequence by Lemma 2 and Lemma 3, we get the following corollary.

Corollary 1. *For $n \geq 1$ and $k \geq 1$, the listings* $\mathsf{GreedyMin}_k(n)$ *and* $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$ *are equivalent.*

3.3 Successor Rule

In this section, we will generalize the successor rules found for non-coloured permutations and signed permutations in [20] for $\mathsf{GreedyMin}_k(n)$ for $k > 2$. We say a permutation in $\mathbb{P}_k(n)$ is *increasing* if it corresponds to a length n subword of the circular string $\rho(1^0 2^0 \cdots n^0)$. For example if $n = 6$ and $k = 4$, then the following permutations are all increasing:

$$2^3 3^3 4^3 5^3 6^3 1^2 \qquad 5^1 6^1 1^0 2^0 3^0 4^0 \qquad 1^0 2^0 3^0 4^0 5^0 6^0 \qquad 5^0 6^0 1^3 2^3 3^3 4^3.$$

A permutation is *decreasing* if it is a reversal of an increasing permutation. A pre-perm is *increasing* (*decreasing*) if it corresponds to a subsequence of an increasing (decreasing) permutation (when the permutation is thought of as a sequence). For example, $5^1 6^1 2^0 4^0$ is an increasing pre-perm, but $5^1 2^0 4^0 6^0$ and $1^2 2^2 3^1 4^0$ are not. Given a permutation π_2, let $\mathsf{succ}(\pi_2)$ denote the successor of π_2 in $\mathbf{Rec}_k(\pi)$ when the listing is considered to be cyclic.

Lemma 4. *Let $\pi_2 = q_1 q_2 \cdots q_n$ be a permutation in the (cyclic) listing $\mathbf{Rec}_k(\pi)$, where $\pi = p_1 p_2 \cdots p_n$ is increasing. Let $q_1 q_2 \cdots q_j$ be the longest prefix of π_2 that is decreasing. Then $\mathsf{succ}(\pi_2) = \mathsf{flip}_j(\pi_2)$.*

Proof. By induction on n. When $n = 1$, the result follows trivially as only flips of length 1 can be applied. Now, for $n > 1$, we focus on the permutations whose successor is the result of a flip of length n and the result will follow inductively by the recursive definition of $\mathbf{Rec}_k(\pi)$. By Lemma 2, the successor of π_2 will be $\mathsf{flip}_n(\pi_2)$ if and only if it is the last permutation in one of the recursive listings of the form $\mathbf{Rec}_k(\rho(\pi)_\mathbf{i}) \cdot r_i$. Recall that r_i is the i-th element in $\rho(\pi)$ when indexed from $r_1 = p_1^{+(k-1)}$ to $r_m = p_n$. As it is clear that at most one permutation is decreasing in each recursive listing, it suffices to show that the last permutation in each listing is decreasing to prove the successor rule holds for flips of length n. By Lemma 1, the last permutation in $\mathbf{Rec}_k(\rho(\pi)_\mathbf{i}) \cdot r_i$ is $\overleftarrow{\mathbf{s}} \cdot r_i$ where $\mathbf{s} = \rho(\pi)_\mathbf{i}^{+(\mathbf{k-1})}$. Since π is increasing, it is clear that $\rho(\pi)_\mathbf{i}$ is increasing and therefore that $\mathbf{s}$ is increasing. Hence, $\overleftarrow{\mathbf{s}}$ is decreasing by definition. Furthermore, by the definition of the circular word $\rho(\pi)$, the element immediately before $r_{i-1-(n-1)}^{+(k-1)}$ in $\rho(\pi)$ is r_i (note the subscript $i - 1 - (n-1)$ is considered modulo nk here). Therefore, $\overleftarrow{\mathbf{s}} \cdot r_i$ is decreasing. Therefore, the successor rule holds for flips of length n and thus for flips of all lengths by induction. $\qquad\square$

Example 3   With respect to the listing $\mathbf{Rec}_{10}(1^0 2^0 3^0 4^0 5^0 6^0)$,

$$\mathsf{succ}(3^8 2^8 5^9 4^9 1^7 6^3) = \mathsf{flip}_4(3^8 2^8 5^9 4^9 1^7 6^3) = 4^0 5^0 2^9 3^9 1^7 6^3$$

and

$$\mathsf{succ}(1^8 3^7 2^6 5^5 4^3 6^2) = \mathsf{flip}_1(1^8 3^7 2^6 5^5 4^3 6^2) = 1^9 3^7 2^6 5^5 4^3 6^2.$$

By applying the previous lemma, computing $\mathsf{succ}(\pi_2)$ for a permutation in the listing $\mathbf{Rec}_k(\pi)$ can easily be done in $O(n)$-time as described in the pseudocode given in Algorithm 1.

Theorem 2. SUCCESSOR*(π) returns the length of the flip required to obtain the successor of π in the listing $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$ in $O(n)$-time.*

Though the worst case performance of SUCCESSOR(π) is $O(n)$-time, on average it is much better. Let $\overline{\sigma}_n^k$ denote (σ_n^k, n), i.e. the sequence of flips used to generate the listing $\mathbf{Rec}_k(\pi)$ with an extra flip of length n at the end to return to the starting permutation. Our goal is to determine the average flip length of $\overline{\sigma}_n^k$, denoted $avg(n,k)$. Note that our analysis generalize the results for $avg(n,1)$ [26] and $avg(n,2)$ [19].

Lemma 5. *For $n \geq 1$ and $k \geq 1$,*

$$avg(n,k) = \sum_{j=0}^{n-1} \frac{1}{k^j j!}.$$

Algorithm 1 Computing the successor of π in the listing $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$

1: **function** SUCCESSOR(π)
2: $\quad$ $incr \leftarrow 0$
3: $\quad$ **for** $j \leftarrow 1$ **to** $n-1$ **do**
4: $\quad\quad$ **if** $v_j < v_{j+1}$ **then** $incr \leftarrow incr + 1$
5: $\quad\quad$ **if** $incr = 2$ **or** ($incr = 1$ **and** $v_{j+1} < v_1$) **then return** $\mathsf{flip}_j(\pi)$
6: $\quad\quad$ **if** $k > 1$ **and** $v_j < v_{j+1}$ **and** ($(c_{j+1} - c_j + k) \bmod k \neq 1$) **then return** $\mathsf{flip}_j(\pi)$
7: $\quad\quad$ **if** $k > 1$ **and** $v_j > v_{j+1}$ **and** ($c_j \neq c_{j+1}$) **then return** $\mathsf{flip}_j(\pi)$
8: $\quad$ **return** $\mathsf{flip}_n(\pi)$

Moreover, $avg(n, k) < \sqrt[k]{e}$.

Proof. By definition of σ_n^k, it is not difficult to see that $\overline{\sigma}_{n+1}^k$ is equivalent to the concatenation of $(n+1)k$ copies of $\overline{\sigma}_n^k$ with the last element in every copy of $\overline{\sigma}_n^k$ incremented by 1. Therefore, we have

$$
\begin{aligned}
avg(n+1, k) &= \frac{\left(1 + \sum_{f \in \overline{\sigma}_n^k} f\right)(n+1)k}{(n+1)! k^{n+1}} \\
&= \frac{\sum_{f \in \overline{\sigma}_n^k} f}{n! k^n} + \frac{1}{n! k^n} \\
&= avg(n, k) + \frac{1}{n! k^n}.
\end{aligned}
$$

Hence, with the trivial base case that $avg(1, k) = 1$, we have

$$
avg(n, k) = \sum_{j=0}^{n-1} \frac{1}{k^j j!}.
$$

Therefore,

$$
avg(n, k) < \sum_{j=0}^{\infty} \frac{1}{k^j j!} = \sqrt[k]{e}
$$

by applying the well-known Maclaurin series expansion for e^x. $\qquad\square$

Observe that the SUCCESSOR function runs in expected $O(1)$-time when the permutation is passed by reference because the average flip length is bounded above by the constant $\sqrt[k]{e}$ as proved in Lemma 5. Thus, by repeatedly applying the successor rule, we obtain a CAT algorithm for generating $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$.

3.4 Loop-free Generation of the Flip-Sequence σ_n^k

Based on the recursive definition of the flip-sequence σ_n^k given in (2), Algorithm 2 will generate σ_n^k in a loop-free manner. The algorithm generalizes a similar algorithm presented by Zaks for non-coloured permutations [26]. The next flip length x is computed using an array of counters $c_1, c_2, \ldots, c_{n+1}$ initialized to 0, and an array of flip lengths $f_1, f_2, \ldots, f_{n+1}$ with each f_i initialized to i. For a formal proof of correctness, we invite the readers to see the simple inductive proof for the non-coloured case in [26], and note the primary changes required to generalize to coloured permutations are in handling of the minimum allowable flip lengths (when $k = 1$, the smallest allowable flip length is 2) corresponding to lines 5-6 and adding a factor of k to line 8.

Algorithm 2 Loop-free generation of the flip-sequence σ_n^k

1: **procedure** FLIPSEQ
2:     $c_1, c_2, \ldots, c_{n+1} \leftarrow 0, 0, \ldots, 0$
3:     $f_1, f_2, \ldots, f_{n+1} \leftarrow 1, 2, \ldots, n+1$
4:     **repeat**
5:         **if** $k = 1$ **then** $x \leftarrow f_2$; $f_2 \leftarrow 2$
6:         **else** $x \leftarrow f_1$; $f_1 \leftarrow 1$
7:         $c_x \leftarrow c_x + 1$
8:         **if** $c_x = kx - 1$ **then**
9:             $c_x \leftarrow 0$
10:        $f_x \leftarrow f_{x+1}$
11:        $f_{x+1} \leftarrow x + 1$
12:        OUTPUT(x)
13:     **until** $x > n$

Theorem 3. *The algorithm* FLIPSEQ *is a loop-free algorithm to generate the flip-sequence σ_n^k one element at a time.*

Since the average flip length in σ_n^k is bounded by a constant, as determined in the previous subsection, Algorithm 2 can be modified to generate $\mathbf{Rec}_k(\pi)$ by passing the initial permutation π as a parameter, outputting π at the start of the **repeat** loop, and updating $\pi \leftarrow \mathsf{flip}_x(\pi)$ at the end of the loop instead of outputting the flip length.

Corollary 2. *The algorithm* FLIPSEQ *can be modified to generate successive permutations in the listing $\mathbf{Rec}_k(\pi)$ in $O(1)$-amortized time.*

4 Summary and Related Work

We presented four different combinatorial algorithms for traversing a specific Hamilton cycle in the k-sided pancake network. The Hamilton cycle corresponds to a flip Gray code listing of coloured permutations. Given such combinatorial listings, it is desirable to have associated ranking and unranking algorithms. Based on the recursive description of the listing in (1), such algorithms are relatively straightforward to derive and implement in $O(n^2)$-time. Details are outlined in Appendix A, along with a C implementation of our algorithms in Appendix B.

References

1. S. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *Computers, IEEE Transactions on*, 38(4):555–566, 1989.
2. C. A. Athanasiadis. Binomial Eulerian polynomials for colored permutations. *J. Combin. Theory Ser. A*, 173:105214, 2020.
3. E. Bagno, D. Garber, and T. Mansour. On the group of alternating colored permutations. *Electron. J. Combin.*, 21(2):Paper 2.29, 2014.
4. A. Borodin. Longest increasing subsequences of random colored permutations. *Electron. J. Combin.*, 6(13):12pp, 1999.
5. W. Y. C. Chen, H. Y. Gao, and J. He. Labeled partitions with colored permutations. *Discrete Math.*, 309(21):6235–6244, 2009.
6. D. S. Cohen and M. Blum. On the problem of sorting burnt pancakes. *Discrete Appl. Math.*, 61(2):105–120, July 1995.
7. A. Duane and J. Remmel. Minimal overlapping patterns in colored permutations. *Electron. J. Combin.*, 18(2):Paper 25, 38, 2011.
8. H. Dweighter. Problem E2569. *American Mathematical Monthly*, 82:1010, 1975.
9. H. Essed and W. Therese. The harassed waitress problem. In A. Ferro, F. Luccio, and P. Widmayer, editors, *Fun with Algorithms*, volume 8496 of *Lecture Notes in Computer Science*, pages 325–339. Springer International Publishing, 2014.
10. G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. MIT Press, August 2009.
11. W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Math.*, 27(1):47–57, 1979.
12. F. Gray. Pulse code communication. *U.S. Patent 2,632,058*, 1947.
13. M. H. Heydari and I. H. Sudborough. On the diameter of the pancake network. *J. Algorithms*, 25(1):67–94, 1997.
14. M. P. Justan, F. P. Muga, and I. H. Sudborough. On the generalization of the pancake network. In *Proceedings International Symposium on Parallel Architectures, Algorithms and Networks. I-SPAN'02*, pages 173–178, 2002.
15. K. Kaneko. Hamiltonian cycles and Hamiltonian paths in faulty burnt pancake graphs. *IEICE - Trans. Inf. Syst.*, E90-D(4):716–721, Mar. 2007.
16. D. E. Knuth. *The Art of Computer Programming*, volume 4: Combinatorial Algorithms, Part 1. Addison-Wesley, 2010.
17. T. Mansour. Pattern avoidance in coloured permutations. *Sém. Lothar. Combin.*, 46:Art. B46g, 12, 2001/02.
18. T. Mansour. Coloured permutations containing and avoiding certain patterns. *Ann. Comb.*, 7(3):349–355, 2003.
19. J. Sawada and A. Williams. Greedy flipping of pancakes and burnt pancakes. *Discrete Appl. Math.*, 210:61–74, 2016.
20. J. Sawada and A. Williams. Successor rules for flipping pancakes and burnt pancakes. *Theoret. Comput. Sci.*, 609(part 1):60–75, 2016.
21. R. Sedgewick. Permutations generation methods. *ACM Comput. Surv.*, 9(2):137–164, 1977.
22. H. Shin and J. Zeng. Symmetric unimodal expansions of excedances in colored permutations. *European J. Combin.*, 52(part A):174–196, 2016.
23. S. Singh. Flipping pancakes with mathematics. *The Guardian*, 2013.
24. A. Williams. O(1)-time unsorting by prefix-reversals in a boustrophedon linked list. In *5th International Conference on FUN with Algorithms*, volume 6099 of *Lecture Notes in Computer Science*, pages 368–379. 2010.
25. A. Williams. The greedy Gray code algorithm. In *Algorithms and Data Structures Symposium, WADS 2013*, volume LNCS 8037, pages 525–536, 2013.
26. S. Zaks. A new algorithm for generation of permutations. *BIT*, 24(2):196–204, 1984.

A Ranking and unranking

A.1 Efficient Ranking

Let $\mathsf{Rank}(\pi)$ denote the rank of permutation $\pi = p_1 p_2 \cdots p_n \in \mathbb{P}_k(n)$ in the listing $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$. The rank of π can be computed by considering the recursive structure of this listing given in (1). Each of the nk recursive sub-lists contain $k^{n-1} \cdot (n-1)!$ permutations and there are $n(c_n + 1) - p_n$ sub-lists that appear before the sub-list containing all permutations ending with p_n. Since each recursive sub-list follows the same structure, using appropriate relabelling we map $p_1 p_2 \cdots p_{n-1}$ to the permutation $q_1 q_2 \cdots q_n$ that appears at the same rank in the respective sub-list starting with $1^0 2^0 \cdots (n-1)^0$. In other words π appears at rank $(n(c_n + 1) - v_n) \cdot k^{n-1} \cdot (n-1)! + \mathsf{Rank}(q_1 q_2 \cdots q_{n-1})$. Formally:

$$\mathsf{Rank}(p_1 p_2 \cdots p_n) = \begin{cases} c_i + 1 & \text{if } n = 1; \\ (n(c_n+1) - p_n) \cdot k^{n-1}(n-1)! \ + \ \mathsf{Rank}(q_1 q_2 \cdots q_{n-1}) & \text{otherwise,} \end{cases}$$

where $q_i = (v_i', c_i')$ and $v_i' = (v_i - v_n) \bmod n$ and $c_i' = (c_i - c_t) \bmod k$ if $v_i < v_n$ and $c_i' = (c_i - c_t - 1) \bmod k$ otherwise.

Example 4 Consider the listing $\mathbf{Rec}_3(\mathbf{123})$ and the coloured permutation $\pi = \mathbf{1}\mathbf{3}\mathbf{2}$. Notice from Example 2 that $\mathbf{1}\mathbf{3}\mathbf{2}$ is the 12th permutation in the sub-list of permutations ending with $\mathbf{2}$. Similarly $\mathbf{2}\mathbf{1}\mathbf{3}$ is the 12th permutation in the first sublist ending with $\mathbf{3}$. The first permutation in the listing that ends with $\mathbf{2}$ is $\mathbf{3}\mathbf{1}\mathbf{2}$. By relabelling this permutation to correspond to $\mathbf{123}$ we consider the shifted sequence $r_1 \cdots r_{27}$ to determine how to relabel each element q_i.

$r_1 r_2 \cdots r_{27}$	$\mathbf{1\,2\,3}$	$\mathbf{3\,1\,2}$	$\mathbf{2\,3\,1}$	$\mathbf{1\,2\,3}$	$\mathbf{3\,1\,2}$	$\mathbf{2\,3\,1}$	$\mathbf{1\,2\,3}$	$\mathbf{3\,1\,2}$	$\mathbf{2\,3\,1}$
shifted $r_1 r_2 \cdots r_{27}$	$\mathbf{2\,3\,1}$	$\mathbf{1\,2\,3}$	$\mathbf{3\,1\,2}$	$\mathbf{2\,3\,1}$	$\mathbf{1\,2\,3}$	$\mathbf{3\,1\,2}$	$\mathbf{2\,3\,1}$	$\mathbf{1\,2\,3}$	$\mathbf{3\,1\,2}$

It is not difficult to see how this shifting produces the mapping to the q_i. From this table: $p_1 = \mathbf{1}$ maps to $q_1 = \mathbf{2}$, and $p_2 = \mathbf{3}$ maps to $q_2 = \mathbf{1}$. The recursive decomposition for computing $\mathsf{Rank}(\mathbf{1}\mathbf{3}\mathbf{2})$ in the ordering $\mathbf{Rec}_3(\mathbf{123})$ is as follows:

$$\begin{aligned} \mathsf{Rank}(\mathbf{1}\mathbf{3}\mathbf{2}) &= (3 \cdot 3 - 2) \cdot 3^2 \cdot 2! + \mathsf{Rank}(\mathbf{2}\mathbf{1}) \\ &= 126 + (2 \cdot 2 - 1) \cdot 3^1 \cdot 1! + \mathsf{Rank}(\mathbf{1}) \\ &= 126 + 9 + 3 = 138 \end{aligned}$$

To compute $\mathsf{Rank}(p_1 p_2 \cdots p_n)$ requires n recursive calls, where $O(n)$ work is required at each call.

Theorem 4. *The algorithm* $\mathsf{Rank}(\pi)$ *returns the rank of the permutation π in the listing* $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$ *in $O(n^2)$ time.*

A.2 Efficient Unranking

To find the permutation $\pi = p_1 p_2 \cdots p_n \in \mathbb{P}_k(n)$ at position $rank$ in the listing $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$, we essentially inverse the operations applied in the ranking procedure: We recursively determine the last symbol given the current rank, then appropriately re-label a permutation found (recursively) at a specific rank in $\mathbf{Rec}_k(1^0 2^0 \cdots (n-1)^0)$. Pseudocode for such an unranking algorithm is as follows.

1: **function** UNRANK($rank, t$)
2:      **if** $t = 1$ **then return** 1^{rank-1}
3:      $x \leftarrow \lfloor \frac{rank-1}{k^{t-1}(t-1)!} \rfloor$
4:      $p_t \leftarrow (t - (x \bmod t))^{\lfloor x/t \rfloor}$
5:      $q_1 q_2 \cdots q_{t-1} \leftarrow$ UNRANK($rank - xk^{t-1}(t-1)!, \; t-1$)
6:      **for** $j \leftarrow 1$ **to** $t-1$ **do**
7:          $p_j \leftarrow 1 + ((v_j + v_t - 1) \bmod t)$
8:          **if** $v_j < v_t$ **then** $c_j \leftarrow (c_j + c_t) \bmod k$
9:          **else** $c_j \leftarrow (c_j + c_t + 1) \bmod k$
10:      **return** $p_1 p_2 \cdots p_t$

> **Example 5**   Consider the permutation $\pi = p_1 p_2 p_3$ at $rank = 138$ in the listing $\mathbf{Rec}_3(\mathbf{123})$. Stepping through the function UNRANK(138,3), the variable x is assigned $\lfloor 137/(9 \cdot 2!) \rfloor = 7$. Thus p_3 will be the last element in the 8th sublist from the recursive description in (1) which is given by $v_3 = 3 - (7 \bmod 3) = 2$ and $c_3 = \lfloor 7/3 \rfloor = 2$. Thus $p_3 = \mathbf{2}$. Subtracting the $7 \cdot (9 \cdot 2!) = 126$ permutations found in the first 7 sub-lists from the $rank = 138$, we are interested in the 12th permutation in the 8th sub-list ending with $\mathbf{2}$. This permutation is computed by recursively determining the 12th permutation in $\mathbf{Rec}_3(12)$, which is $\mathbf{21}$ and applying an appropriate relabelling to obtain $p_1 p_2 = \mathbf{13}$. Thus $\pi = \mathbf{132}$.

Theorem 5. *The algorithm* UNRANK*($rank, n$) returns the permutation at position $rank$ in the listing $\mathbf{Rec}_k(1^0 2^0 \cdots n^0)$ in $O(n^2)$ time.*

B Implementation of Algorithms in C

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_N 20

int n, k, a[MAX_N], color[MAX_N], f[MAX_N], c[MAX_N];
long int rank, total, powK[MAX_N], factorial[MAX_N];

//------------------------------------------------
// OPERATIONS ON PERMUTATIONS
//------------------------------------------------
void Flip(int t) {
    int i, b[MAX_N];

    for (i=1; i<=t; i++) b[i] = a[t-i+1];
    for (i=1; i<=t; i++) a[i] = b[i];

    for (i=1; i<=t; i++) b[i] = color[t-i+1];
    for (i=1; i<=t; i++) color[i] = (b[i]+1) % k;
}
//----------------------------------------
void Print() {
    int i;
    for (i=1; i<=n; i++)  printf("(%d,%d) ", a[i], color[i]);
    total++;
}
//----------------------------------------
// RANKING
//----------------------------------------
long int Rank(int t){
    int j;

    if (t == 1) return color[1] + 1;

    for (j=1; j<t; j++)  {
        if (a[j] <  a[t]) color[j] = (color[j] - color[t] + k) % k;
        if (a[j] >  a[t]) color[j] = (color[j] - color[t] - 1 + k) % k;
        a[j] = (a[j] - a[t] + t) % t;
    }
    return (( (color[t]+1) * t - a[t]) * factorial[t-1] * powK[t-1] + Rank(t-1));
}
//----------------------------------------
// UNRANKING
//----------------------------------------
void UnRank(long int rank, int t){
    int j,x;

    if (t == 1) {
        a[1] = 1;
        color[1] = rank - 1;
        return;
    }

    x = (rank-1) / (factorial[t-1] * powK[t-1]);
    rank = rank - x * factorial[t-1] * powK[t-1];

    // x ranges from 0 to kn-1
    a[t] = t - (x % t);
    color[t] = (int) x/t;

    UnRank(rank, t-1);

    for (j=1; j<t; j++)  {
        a[j] =  1 + (a[j] + a[t] - 1) % t;
        if (a[j] <  a[t]) color[j] = (color[j] + color[t]) % k;
        if (a[j] >  a[t]) color[j] = (color[j] + color[t] + 1) % k;
```

```c
    }
}
//---------------------------------------
// SUCCESSOR
//---------------------------------------
void Successor() {
    int j, incr=0;

    for (j=1; j<n; j++) {
        if (a[j] < a[j+1]) incr++;
        if (incr == 2 || (incr == 1 && a[j+1] < a[1])) break;

        if (k > 1 && a[j] < a[j+1] && (color[j+1] - color[j] + k) % k != 1) break;
        if (k > 1 && a[j] > a[j+1] &&  color[j+1] != color[j]) break;
    }
    Flip(j);
}
//---------------------------------------
// GENERATION - BY FLIP SEQUENCE
//---------------------------------------
int Next() {
    int j;

    if (k == 1) { j = f[2];    f[2] = 2;  }
    else        { j = f[1];    f[1] = 1;  }

    c[j]++;
    if (c[j] == k*j-1) {
        c[j] = 0;
        f[j] = f[j+1];
        f[j+1] = j+1;
    }
    return(j);
}
//---------------------------------------
void Gen() {
    int j;

    // INITIAL PERM
    for (j=1; j<=n; j++) a[j] = j;
    for (j=1; j<=n; j++) color[j] = 0;

    for (j=1; j<=n+1; j++) c[j] = 0;
    for (j=1; j<=n+1; j++) f[j] = j;

    do {
        Print();  printf("\n");
        j= Next();
        Flip(j);

    } while (j <= n);

    printf("Total = %ld\n\n", total);
}
//==============================================================================
int main() {
    int i,j,type;

    //---------------------------------------
    // INPUT
    //---------------------------------------
    printf("  1. Generation by flips \n");
    printf("  2. Ranking \n");
    printf("  3. UnRanking \n");
    printf("  4. Successor rule \n");
    printf("\n ENTER selection #: ");    scanf("%d", &type);

    if (type < 0 || type > 4)  { printf("\n INVALID ENTRY\n\n");  exit(0); }
```

```c
    // long int constraints: MAX n seems to be 20 or 16 (k=2)
    printf(" ENTER order n: ");    scanf("%d", &n);
    printf(" ENTER colors k: ");   scanf("%d", &k);

    factorial[0] = 1;
    for (j=1; j< MAX_N; j++) factorial[j] = factorial[j-1] * j;
    powK[0] = 1;
    for (j=1; j< MAX_N; j++) powK[j] = powK[j-1] * k;

    if (type == 2 || type == 4) {
        printf(" ENTER coloured permutation of form:  p_1,c_1  p_2,c_2  ... ");
        for (i=1; i<=n; i++) scanf("%d,%d", &a[i], &color[i]);
    }
    if (type == 3) {
        rank = factorial[n] * powK[n];
        printf(" ENTER rank (between 1 and %ld): ", rank);
        scanf("%ld", &rank);
    }
    printf("\n");
    //----------------------------------

    if (type == 1) Gen();
    if (type == 2) {
        printf("The rank of permutation ");
        Print();
        rank = Rank(n);
        printf(" is:  %ld\n\n", rank);
    }
    if (type == 3) {
        UnRank(rank,n);
        printf("The permutation of rank %ld is: ", rank);
        Print(); printf("\n");
    }
    if (type == 4) {
        printf("The successor of permutation ");
        Print();
        Successor();
        printf(" is:  ");
        Print(); printf("\n");
    }
}
```