Greedy Pancake Flipping

Aaron Williams\textsuperscript{1}

Department of Mathematics and Statistics
McGill University
Montreal, Canada

Joe Sawada\textsuperscript{3,2}

School of Computer Science
University of Guelph
Guelph, Canada

Abstract
We prove that a stack of pancakes can be rearranged in all possible ways by a greedy process: Flip the maximum number of topmost pancakes that gives a new stack. We also show that the previous Gray code for rearranging pancakes (S. Zaks, A New Algorithm for Generation of Permutations BIT 24 (1984), 196-204) is a greedy process: Flip the minimum number of topmost pancakes that gives a new stack.

Keywords: greedy algorithm, pancake sorting, Gray code, permutations, prefix-reversal

1 Introduction
Take a stack of pancakes, numbered 1, 2, \ldots, n by increasing diameter, and repeat the following: Flip the maximum number of topmost pancakes that gives a new stack. For example, if the first stack is 12345, then the second stack is created by flipping all five pancakes to give 54321. To create the third stack

\textsuperscript{1} Email: haron@uvic.ca
\textsuperscript{2} Research partially funded by an NSERC discovery grant.
\textsuperscript{3} Email: jsawada@uoguelph.ca
from the second stack, we cannot flip all five pancakes (since it would recreate 12345), however we can flip the top four pancakes to give 23451. Each ‘flip’ is formally known as a prefix-reversal. Figure 1 illustrates this process.

![Fig. 1. The result of greedily flipping the maximum number of topmost pancakes when n = 5. The order is read from left-to-right. Previously generated stacks that are considered by the algorithm appear as crossed out strings.](image)

A Gray code is an order of combinatorial objects in which consecutive objects are close to each other. The term ‘close’ often refers to one application of a given operation, such as a swap, rotation, or reversal. Gray codes have many applications, including efficient algorithms for generating combinatorial objects. See Sedgewick [5], Savage [4], and Section 7.2.1.2 of Knuth [3] for surveys. We consider $P(n)$, the permutations of $\{1, 2, \ldots, n\}$ written as strings in one-line notation. For example, $P(3) = \{123, 132, 213, 231, 312, 321\}$. We prove that our maximum flip process always generates $P(n)$. In other words, greedily reversing maximum length prefixes generates a prefix-reversal Gray code. For example, the order for $n = 4$ appears below, where overlines denote the prefix-reversal that generates each successive string

\[
\begin{align*}
1234 & \quad 4321 \quad 2341 \quad 1432 \quad 3412 \quad 2143 \quad 4123 \quad 3214, \\
2134 & \quad 4132 \quad 3142 \quad 2413 \quad 1423 \quad 3241 \quad 4231 \quad 1324, \\
3124 & \quad 4213 \quad 1243 \quad 3421 \quad 2431 \quad 1342 \quad 4312 \quad 2134.
\end{align*}
\]

Prefix-reversal Gray codes are used in interconnection networks, where the “pancake network” is a popular topology (see Siegel [6]). The only prior prefix-reversal Gray code known to the authors is Zaks [7]. Sorting by prefix-reversals was recently shown to be NP-hard by Bulteau, Fertin, and Rusu [1].

To understand the significance of our maximum flip algorithm, let us consider a similar greedy process. A left prefix-rotation of length $k$ (or simply a rotation of length $k$) circularly moves a string’s first $k$ symbols one position to the left. For example, $654321 = 543621$ illustrates a rotation of length 4. The minimum rotation process builds a list of $P(n)$ by repeatedly rotating minimum length prefixes. This process generates the following list from 1234 (with arrows showing successive rotations): $\overleftarrow{1234}, \overleftarrow{2134}, \overleftarrow{1324}, \overleftarrow{3124}, \overleftarrow{1243}, 1243, 2143, \overleftarrow{1423}, 4123$. The process terminates at 4123 since $\overleftarrow{4123} = 1423,$
4123 = 1243, and 4123 = 1234 have already been generated. Similarly, it is easy to verify that the maximum rotation process does not generate \( P(4) \).

In Section 2 we prove that the maximum flip process always generates \( P(n) \). Section 3 discusses additional results involving efficient algorithms, the analogous minimum flip process, and how our main result fits into a new framework on greedy Gray codes.

### 2 Maximum Greedy Flipping

In this section we investigate the maximum flip process starting from \( 12 \cdots n \).

Pseudocode for MaxFlip(\( n \)) and its output for \( n = 5 \) appear in Figure 2.

In this article, we denote individual permutations in bold, and index their symbols using italics and subscripts. Thus, if \( p = 213 \), then \( p_1 = 2 \), \( p_2 = 1 \), and \( p_3 = 3 \). Bold subscripts denote distinct permutations, so \( p_1, p_2, p_3 \) is an order of three permutations. The prefix-reversal of length \( i \) is denoted \( \text{flip}_i \), so if \( p = p_1p_2 \cdots p_n \in P(n) \), then \( \text{flip}_i(p) = p_ip_{i-1} \cdots p_1p_{i+1}p_{i+2} \cdots p_n \in P(n) \).

Finally, \( \cdot \) denotes concatenation, so if \( p = 123 \), then \( p \cdot 4 = 1234 \).

Successive permutations in each column of Figure 2 differ by alternately applying \( \text{flip}_5 \) and \( \text{flip}_4 \). This alternation is greedy since MaxFlip(\( n \)) cannot apply \( \text{flip}_n \) twice in a row. Observe that \( \text{flip}_n \) then \( \text{flip}_{n-1} \) rotates a string to the left. Similarly, \( \text{flip}_{n-1} \) then \( \text{flip}_n \) rotates a string to the right. For example,

\[
\begin{align*}
\text{flip}_4(\text{flip}_5(12345)) &= \text{flip}_4(54321) = 23451 \quad \text{and} \quad 12345 = 23451 \\
\text{flip}_5(\text{flip}_4(12345)) &= \text{flip}_5(43215) = 51234 \quad \text{and} \quad 12345 = 51234.
\end{align*}
\]

This explains why every second string in the first column of Figure 2 is a successive left rotation of 12345. Similarly, the column’s remaining strings are successive right rotations of its reverse 54321. Collectively, the strings in the first column form a bracelet, meaning the set is closed under rotation and reversal. \( P(n) \) partitions into \( \frac{(n-1)!}{2} \) bracelets, each containing \( 2n \) strings. In fact, each column of Figure 2 constitutes a bracelet. Next focus on the order of the bracelets. Notice that every second string in \( 1 \cdots 1234 \), 2341, 3412, 4123, 2314, and so on — contributes to the first string in each column of Figure 2 — 12345, 23415, 34125, 41235, 23145, and so on. This discussion shows how the order generated by MaxFlip(\( n \)) can be understood recursively.

Towards our main result, define the bracelet order of \( p_1 \in P(n) \) as

\[
\text{bracelet}(p_1) = p_1, p_2, \ldots, p_{2n} \quad \text{such that} \quad p_i = \begin{cases} 
\text{flip}_n(p_{i-1}) & \text{if } i \text{ is even} \\
\text{flip}_{n-1}(p_{i-1}) & \text{if } i > 1 \text{ is odd}.
\end{cases}
\]
Function \textup{MaxFlip}(n)

Initialize permutation $p \leftarrow 1 \cdots n$, and list $L \leftarrow p$, and integer $i \leftarrow n$

\begin{verbatim}
while $i \geq 2$ do
  \textbf{if} flip$_i(p)$ is not already in $L$ \textbf{then}
    Set $p \leftarrow \text{flip}_i(p)$, and add $p$ to the end of $L$, and set $i \leftarrow n$
  \textbf{else}
    Set $i \leftarrow i - 1$
\end{verbatim}

end while

Thus, \textup{bracelet}(p_1) alternately applies \text{flip}_n and \text{flip}_{n-1} as in Figure 2’s columns and (1)’s rows. In particular, the last string in \textup{bracelet}(p_1) is \text{flip}_{n-1}(p_1).

**Theorem 2.1** \textup{MaxFlip}(n) generates every string in $\mathbb{P}(n)$. In particular, the order can be understood recursively by \textup{MaxFlip}(2) = 12, 21 and the following

\begin{equation}
\textup{MaxFlip}(n-1) = q_1 \cdot q_2 \cdot \ldots \cdot q_{(n-1)!} \text{ implies } \textup{MaxFlip}(n) = \textup{bracelet}(q_1 \cdot n), \textup{bracelet}(q_3 \cdot n), \ldots, \textup{bracelet}(q_{(n-1)!-1} \cdot n). \tag{2}
\end{equation}

Furthermore, every second prefix-reversal used in the order is \text{flip}_n.

**Proof.** Suppose (2) is an order of $\mathbb{P}(n-1)$ and every second prefix-reversal is \text{flip}_{n-1}. Thus, \text{flip}_{n-1}(q_{2i-1}) = q_{2i-2}$ for $0 \leq i \leq (n-1)!-1$. First we claim that the sequence \text{bracelet}(q_1 \cdot n), \text{bracelet}(q_3 \cdot n), \ldots, \text{bracelet}(q_{(n-1)!-1} \cdot n) contains every string in $\mathbb{P}(n)$ exactly once. Since the set of strings in the sequence is closed under rotation, this claim is true if the sequence includes every string in $\mathbb{P}(n)$ that ends in $n$. Clearly, the sequence contains $q_1 \cdot n, q_4 \cdot n, \ldots, q_{(n-1)!-1} \cdot n$, so we must show it also contains $q_2 \cdot n, q_4 \cdot n, \ldots, q_{(n-1)!-1} \cdot n$. This follows from
the fact that \( \text{bracelet}(q_{2i+1} \cdot n) \) ends with \( \text{flip}_{n-1}(q_{2i+1} \cdot n) = q_{2i+2} \cdot n \) for \( 0 \leq i \leq (n-1)! - 1 \). Therefore, if we prove (3), then MaxFlip\((n)\) contains all of \( \mathbb{P}(n) \).

Now suppose that MaxFlip\((n)\) begins by generating the following strings

\[
\text{bracelet}(q_1 \cdot n), \text{bracelet}(q_3 \cdot n), \ldots, \text{bracelet}(q_{2i+1} \cdot n)
\]

(4)

for some \( 0 \leq i < (n-1)! - 1 \). (This supposition is clearly true for \( i = 0 \).) As previously argued, the strings in (4) that end with \( n \) are precisely \( q_1 \cdot n, q_2 \cdot n, \ldots, q_{2i+1} \cdot n, q_{2i+2} \cdot n \), where \( q_{2i+2} \cdot n \) is the last string in (4). Now consider which string (if any) will be generated next by MaxFlip\((n)\). It cannot apply \( \text{flip}_n \) to \( q_{2i+2} \cdot n \) since the result equals the second-last string of \( \text{bracelet}(q_{2i+1} \cdot n) \) in (4). Therefore, the next string generated by MaxFlip\((n)\) must end with \( n \).

Due to our assumption (2), MaxFlip\((n-1)\) follows \( q_{2i+2} \cdot n \) by generating \( q_{2i+3} \cdot n \). Therefore, MaxFlip\((n)\) must follow \( q_{2i+2} \cdot n \) by generating \( q_{2i+3} \cdot n \). Since \( q_{2i+3} \cdot n \) is the first string in its bracelet generated by MaxFlip\((n)\), it must be that MaxFlip\((n)\) extends (4) by generating \( \text{bracelet}(q_{2i+3} \cdot n) \). Therefore, (4) is true when \( i+1 \) replaces \( i \), and hence (4) is true for \( i = (n-1)! - 1 \) by induction. This proves (3). Furthermore, every second prefix-reversal is \( \text{flip}_n \) due to its use of bracelet orders. Therefore, the theorem follows by induction.

\[ \square \]

### 3 Future Work

We have shown that the maximum flip process generates a prefix-reversal Gray code for \( \mathbb{P}(n) \). In the expanded version of this article, we address the efficient generation of our new Gray code. Specific results include the following:

- The order can be \textit{ranked} and \textit{unranked} in \( O(n^2) \)-time. This means that the position of any \( p \in \mathbb{P}(n) \) in the order can be computed in \( O(n^2) \)-time, and the specific \( p \in \mathbb{P}(n) \) at a given position can be computed in \( O(n^2) \)-time.

- The \textit{successor} can be computed in \( O(n) \)-time. This means that the permutation following a given \( p \in \mathbb{P}(n) \) can be computed in \( O(n) \)-time without any additional information.

- Successive prefix-reversal lengths can be generated by a \textit{loopless} algorithm. This means that successive values in the sequence \( \text{flip}_{i_1}, \text{flip}_{i_2}, \ldots, \text{flip}_{i_{n-1}} \) that creates the Gray code are generated in worst-case \( O(1) \)-time.

- Successive permutations in the Gray code can be generated by a \textit{constant amortized time} algorithm. This means that successive permutations in the Gray code \( p_1, p_2, \ldots, p_{n!} \) are visited in amortized \( O(1) \)-time. In this algo-
algorithm, we alternately visit two “current” permutations stored in a linked list.

Besides these algorithmic results, we also complement Theorem 2.1 by considering the greedy minimum flip process: Flip the minimum number of topmost pancakes that gives a new stack. Figure 3 illustrates the process for $n = 5$.

Fig. 3. The result of greedily flipping the minimum number of topmost pancakes when $n = 5$. The order is read from left-to-right. Previously generated stacks that are considered by the algorithm appear as crossed out strings.

We will prove that the order generated by $\text{MinFlip}(n)$ is in fact the prefix-reversal Gray code by Zaks, which was previously defined recursively.

**Theorem 3.1** $\text{MinFlip}(n)$ generates every string in $P(n)$ in Zak’s order [7].

We conclude this article by mentioning that Theorem 2.1 represents the first new Gray code discovered using the greedy Gray code algorithm, which is a meta-heuristic that has given new interpretations to many classic Gray codes (see Graham and Williams [2]).

**References**


