Skipping Ropes: An Efficient Gray Code Algorithm for Generating Wiggly Permutations

Original Paper by Vincent Pilaud and Aaron Williams

Atishaya Maharjan

GADA Lab

October 3, 2025

Binary Reflected Gray Codes

Binary Reflected Gray Codes

A ***Binary Reflected Gray Code*** is the sequence of binary numbers that differ by one bit.

Decimal	3-bit Binary Reflected Gray Code
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Binary Reflected Gray Codes

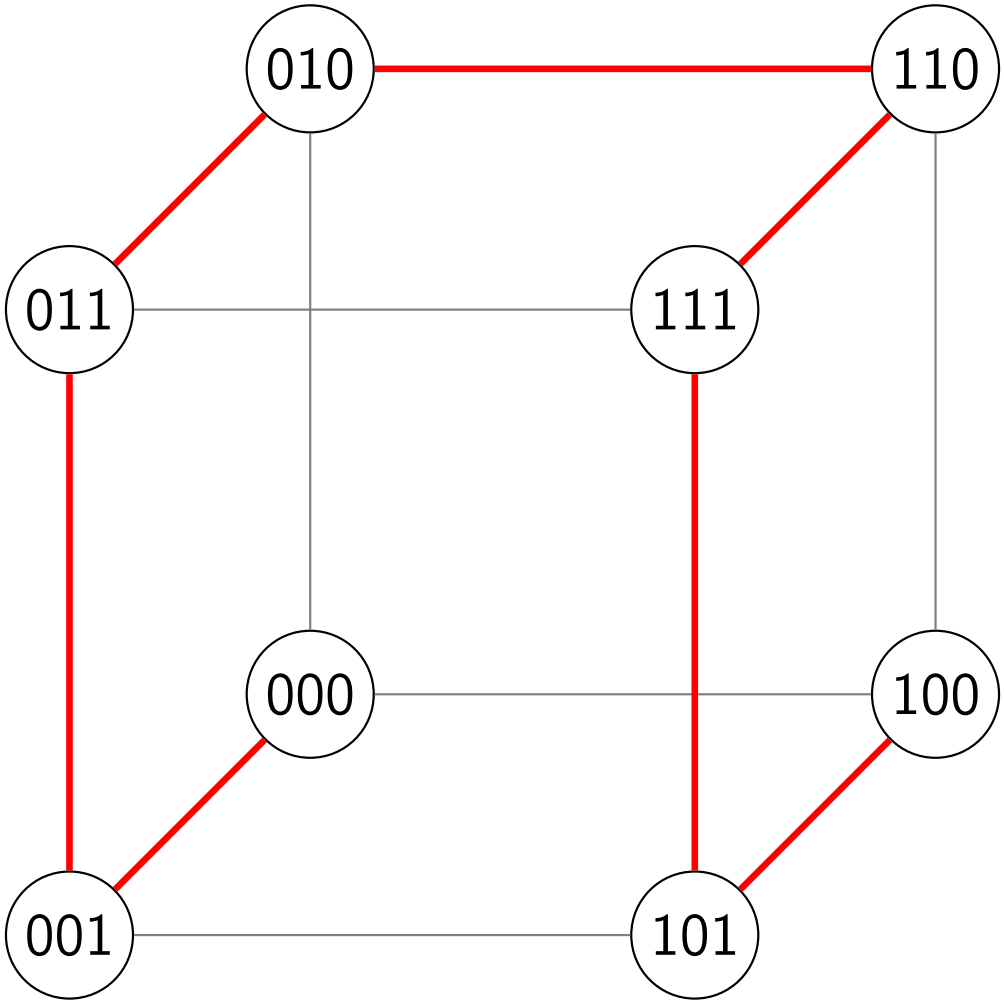


Figure: A Hamiltonian path on Q_3 is equivalent to listing the Gray codes in sequence.

Generation of Binary Reflected Gray Codes

We can formulate the Binary Reflected Gray Codes in a recursive manner:

$$B(n) = \begin{cases} 0 \cdot B(n-1),\ 1 \cdot B^R(n-1), & \text{for } n > 1 \\ 0, 1, & \text{for } n = 1 \end{cases}$$

Generation of Binary Reflected Gray Codes

We can formulate the Binary Reflected Gray Codes in a recursive manner:

$$B(n) = \begin{cases} 0 \cdot B(n-1),\ 1 \cdot B^R(n-1), & \text{for } n > 1 \\ 0, 1, & \text{for } n = 1 \end{cases}$$

So for example we would have:

$$B(1) = 0, 1$$
$$B(2) = 00, 01, 11, 10$$
$$B(3) = 000, 001, 011, 010, 110, 111, 101, 100$$

Beyond Binary Strings: Combinatorial Objects

Natural Question: Can we create Gray codes for other combinatorial families?

Examples of Combinatorial Objects:

- **Combinations:** All k-element subsets of $\{1, 2, \ldots, n\}$
- **Permutations:** All orderings of $\{1, 2, \ldots, n\}$
- **Partitions:** All ways to partition a set into blocks
- **Trees:** All spanning trees of a graph
- **Strings:** Words with various constraints

In this talk, we will mostly consider permutations.

Generating permutation sequences using a computer

- In the 20^{th} century, computer scientists were looking to create an algorithm that could generate all $n!$ permutations of a n length string quickly using an algorithm.

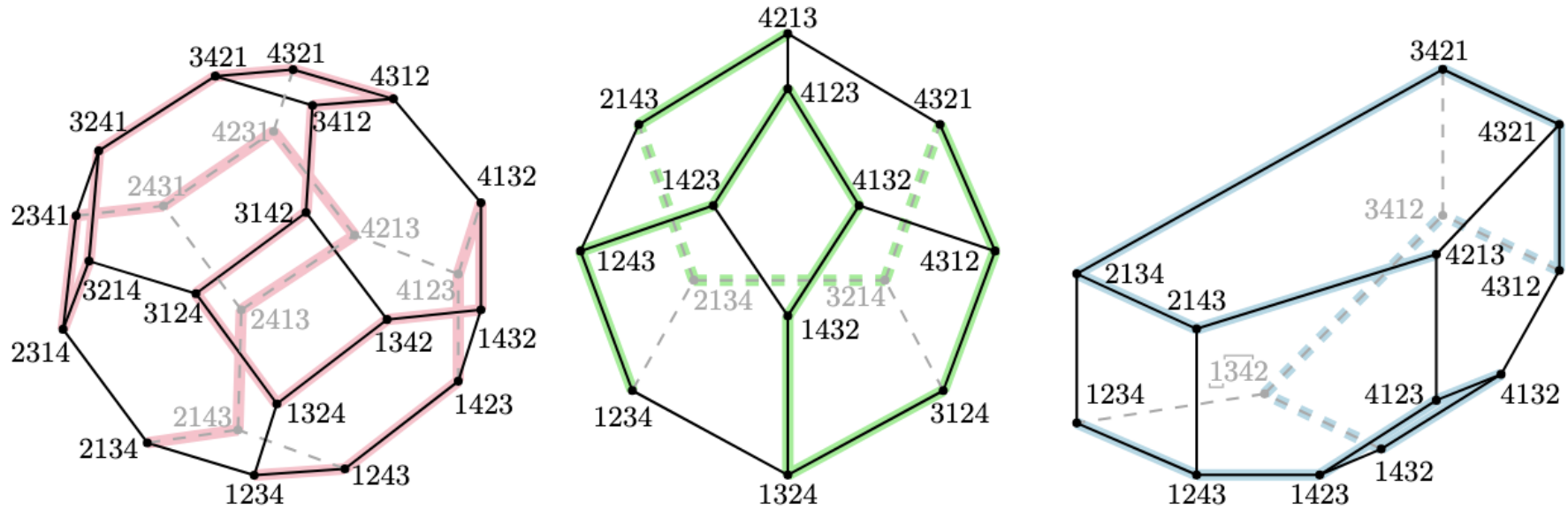
Generating permutation sequences using a computer

- In the 20^{th} century, computer scientists were looking to create an algorithm that could generate all $n!$ permutations of a n length string quickly using an algorithm.
- Plain changes was discovered independently multiple times by a lot of different people, and the pattern became known as the Steinhaus-Johnson-Trotter algorithm [3].

Generating permutation sequences using a computer

- In the 20^{th} century, computer scientists were looking to create an algorithm that could generate all $n!$ permutations of a n length string quickly using an algorithm.
- Plain changes was discovered independently multiple times by a lot of different people, and the pattern became known as the Steinhaus-Johnson-Trotter algorithm [3].
- We can visualize this problem using flip graphs!

Flip graphs and plain changes



(a) Permutahedron for $n = 4$. **(b)** Associahedron for $n = 4$. **(c)** Wigglyhedron for $n = 4$.

Figure: Flip graph visualizations using different schemes [2]

Beyond Binary Strings: Combinatorial Objects

Redefining Gray Codes

A gray code is simply the order of successive objects on the path of a flip graph.

Beyond Binary Strings: Combinatorial Objects

Redefining Gray Codes

A gray code is simply the order of successive objects on the path of a flip graph.

- There has been a lot of interest in finding out simple and efficient generating algorithms for gray codes for various permutation objects.

Johnson-Trotter Algorithm for All Permutations

Goal: Generate all $n!$ permutations using only adjacent transpositions.

Johnson-Trotter Algorithm for All Permutations

Goal: Generate all $n!$ permutations using only adjacent transpositions.

Algorithm:

1. Start with $\pi = 12 \cdots n$
2. Each element has a direction (left $\leftarrow$ or right $\rightarrow$)
3. An element is *mobile* if it's larger than its neighbor in its direction
4. Repeatedly:
   - Find largest mobile element m
   - Swap m with neighbor in its direction
   - Reverse direction of all elements $> m$

Result: Generates all permutations, each differing from previous by one adjacent swap.

Johnson-Trotter Example: $n = 3$

Step	Permutation	Action
0	$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$	largest mobile is 3
1	$\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}$	3 moved left, largest mobile is 3
2	$\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$	3 moved left, reverse dir of none, largest mobile is 2
3	$\overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{1}$	2 moved left, reverse dir of 3, largest mobile is 3
4	$\overleftarrow{2}\ \overleftarrow{3}\ \overrightarrow{1}$	3 moved left, largest mobile is 3
5	$\overleftarrow{2}\ \overrightarrow{1}\ \overrightarrow{3}$	3 moved left, no mobile

Sequence: $123 \to 132 \to 312 \to 321 \to 231 \to 213$

Permutations: Basic Definitions

Definition

Permutation A permutation of $[n] = \{1, 2, \ldots, n\}$ is a bijection $\pi : [n] \to [n]$.

Permutations: Basic Definitions

Definition

Permutation A permutation of $[n] = \{1, 2, \ldots, n\}$ is a bijection $\pi : [n] \to [n]$.

Example: $\pi = 3142$ means

$$\pi(1) = 3, \quad \pi(2) = 1, \quad \pi(3) = 4, \quad \pi(4) = 2$$

Permutations: Basic Definitions

Definition

Permutation A permutation of $[n] = \{1, 2, \ldots, n\}$ is a bijection $\pi : [n] \to [n]$.

Example: $\pi = 3142$ means

$$\pi(1) = 3, \quad \pi(2) = 1, \quad \pi(3) = 4, \quad \pi(4) = 2$$

Inverse: $\pi^{-1}(j)$ is the position where value j appears.

For $\pi = 3142$: $\pi^{-1}(1) = 2$, $\pi^{-1}(2) = 4$, $\pi^{-1}(3) = 1$, $\pi^{-1}(4) = 3$

Inversions

Definition (Inversion)

For a permutation π of $[n]$, an *inversion* is a pair (i, j) with $i < j$ but $\pi(i) > \pi(j)$. The *inversion set* is $\mathrm{inv}(\pi)$.

Inversions

Definition (Inversion)

For a permutation π of $[n]$, an *inversion* is a pair (i, j) with $i < j$ but $\pi(i) > \pi(j)$. The *inversion set* is $\mathrm{inv}(\pi)$.

Example. $\pi = 3\ 1\ 4\ 2$.

Pairs (positions): $(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)$

Inversions: $(1,2)$ since $3 > 1$; $(1,4)$ since $3 > 2$;

$(3,4)$ since $4 > 2$.

$\Rightarrow\ \mathrm{inv}(\pi) = \{(1,2), (1,4), (3,4)\},\ |\mathrm{inv}(\pi)| = 3.$

Inversions

Definition (Inversion)

For a permutation π of $[n]$, an *inversion* is a pair (i, j) with $i < j$ but $\pi(i) > \pi(j)$. The *inversion set* is $\mathrm{inv}(\pi)$.

Example. $\pi = 3\ 1\ 4\ 2$.

Pairs (positions): $(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)$

Inversions: $(1,2)$ since $3 > 1$; $(1,4)$ since $3 > 2$;

$(3,4)$ since $4 > 2$.

$\Rightarrow\ \mathrm{inv}(\pi) = \{(1,2), (1,4), (3,4)\},\ |\mathrm{inv}(\pi)| = 3.$

Remark: inversion count measures distance from the identity.

Cover Relation

Definition (Cover Relation)

σ *covers* π if $\pi < \sigma$ and there is no τ with $\pi < \tau < \sigma$. In our context this means $\mathrm{inv}(\sigma) = \mathrm{inv}(\pi) \cup \{(u, v)\}$ — exactly one new inversion is added.

Cover Relation

Definition (Cover Relation)

σ *covers* π if $\pi < \sigma$ and there is no τ with $\pi < \tau < \sigma$. In our context this means $\mathrm{inv}(\sigma) = \mathrm{inv}(\pi) \cup \{(u, v)\}$ — exactly one new inversion is added.

Example (n=3):

$$123 \quad \longrightarrow \quad 132$$

- $\mathrm{inv}(123) = \varnothing, \quad \mathrm{inv}(132) = \{(3, 2)\}$.
- So 132 covers 123 and the new inversion is $(3, 2)$.

Cover Relation

Definition (Cover Relation)

σ *covers* π if $\pi < \sigma$ and there is no τ with $\pi < \tau < \sigma$. In our context this means $\mathrm{inv}(\sigma) = \mathrm{inv}(\pi) \cup \{(u, v)\}$ — exactly one new inversion is added.

Example (n=3):

$$123 \quad \longrightarrow \quad 132$$

- $\mathrm{inv}(123) = \varnothing, \quad \mathrm{inv}(132) = \{(3, 2)\}$.
- So 132 covers 123 and the new inversion is $(3, 2)$.

Takeaway: In our case, cover edges correspond to adding a single inversion.

Edge labeling for the greedy rule

Label of a cover edge (π, σ). For a cover (one inversion added) the edge is labeled with the *larger value* of the new inversion. Equivalently: if (u, v) is the new inversion with $u < v$, label the edge by v.

Edge labeling for the greedy rule

Label of a cover edge (π, σ). For a cover (one inversion added) the edge is labeled with the *larger value* of the new inversion. Equivalently: if (u, v) is the new inversion with $u < v$, label the edge by v.

Why this label? The greedy algorithm picks, at each step, an available cover edge with *largest* label. Intuition: "move the largest value first" (analogue of Johnson–Trotter).

Worked example (from the paper):

$$\pi = 1\ 3\ 4\ 2 \quad \longrightarrow \quad \sigma = 1\ 4\ 3\ 2$$

- $\mathrm{inv}(\pi) = \{(3,2),(4,2)\}$, $\mathrm{inv}(\sigma) = \{(3,2),(4,2),(4,3)\}$.
- New inversion: $(4,3)$ so the edge (π, σ) gets label 4.

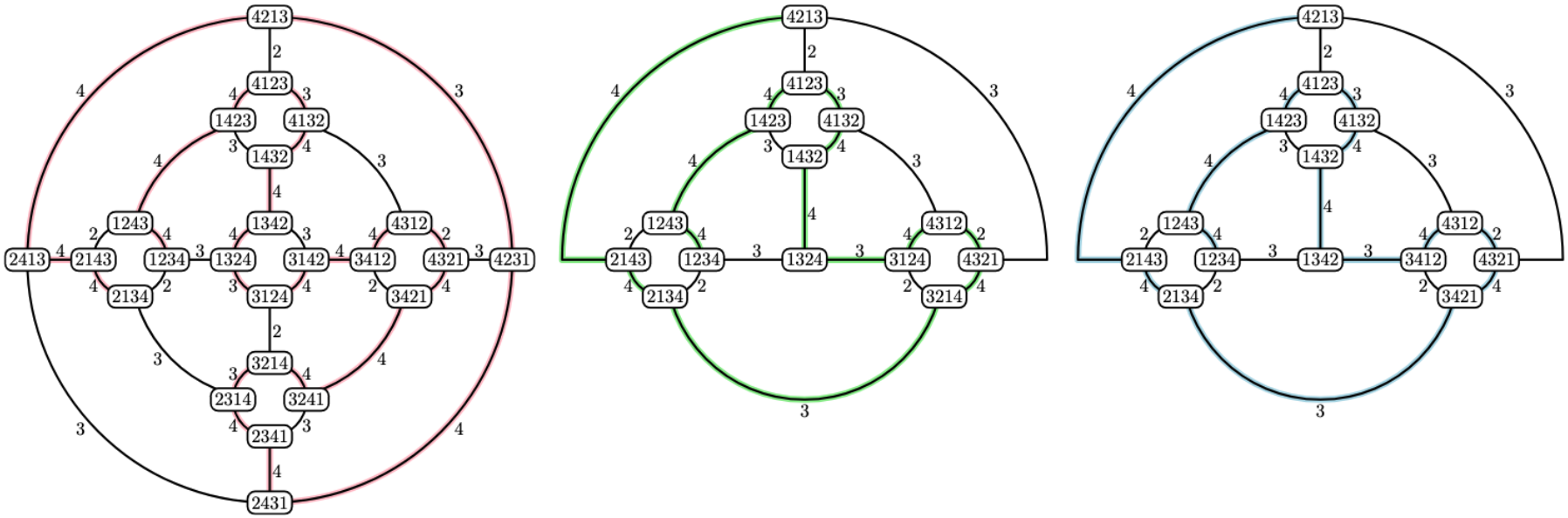
S_3 tiny table: covers and labels

Permutations of 3 and their cover-edges (weak order covers).

perm π	covers σ	edge label(s)
123	132	3
123	213	2
132	312	3
132	231	2
213	231	3
213	312	2

Greedy walk idea: from a node choose outgoing cover with largest label (e.g. from 123 choose label 3-edge $123 \to 132$).

Edge labelled algorithm visualization



(a) S_4 with swaps. **(b)** $\mathsf{Av}_4(231)$ with minimal jumps. **(c)** W_4 with wiggly flips.

Local Moves for Permutations

Common local moves:

1. **Adjacent transposition:** Swap π_i and π_{i+1}

$$\pi = \cdots \pi_i \, \pi_{i+1} \cdots \;\rightarrow\; \cdots \pi_{i+1} \, \pi_i \cdots$$

Example: $3142 \rightarrow 1342$ (swap positions 1 and 2)

Local Moves for Permutations

Common local moves:

1. **Adjacent transposition:** Swap π_i and π_{i+1}

$$\pi = \cdots \pi_i \, \pi_{i+1} \cdots \;\rightarrow\; \cdots \pi_{i+1} \, \pi_i \cdots$$

   Example: $3142 \rightarrow 1342$ (swap positions 1 and 2)

2. **Arbitrary transposition:** Swap any two elements

Local Moves for Permutations

Common local moves:

1. **Adjacent transposition:** Swap π_i and π_{i+1}

$$\pi = \cdots \pi_i \, \pi_{i+1} \cdots \;\rightarrow\; \cdots \pi_{i+1} \, \pi_i \cdots$$

   Example: $3142 \rightarrow 1342$ (swap positions 1 and 2)

2. **Arbitrary transposition:** Swap any two elements
3. **Star transposition:** Swap π_1 with π_i for any i

Local Moves for Permutations

Common local moves:

1. **Adjacent transposition:** Swap π_i and π_{i+1}

$$\pi = \cdots \pi_i \, \pi_{i+1} \cdots \rightarrow \cdots \pi_{i+1} \, \pi_i \cdots$$

   Example: $3142 \rightarrow 1342$ (swap positions 1 and 2)
2. **Arbitrary transposition:** Swap any two elements
3. **Star transposition:** Swap π_1 with π_i for any i
4. **Reversal:** Reverse a contiguous subsequence

Note: Adjacent transpositions are most restrictive, hence most interesting for Gray codes!

Pattern-Avoiding Permutations

Definition. A permutation π *avoids* a pattern σ if π contains no subsequence order-isomorphic to σ.

Example: $\pi = 2413$ avoids 123 because no three positions have increasing values.

Classical results:

- Permutations avoiding 123 are counted by Catalan numbers
- Many pattern classes have elegant enumeration formulas
- Rich connection to algebraic combinatorics

Definition: Wiggly Permutations

Definition [1] A permutation $\pi \in S_n$ is *wiggly* if for all odd j with $1 \leq 2j \leq n$:

$$\begin{aligned} &\textbf{If } 2j-1 \text{ appears } \textbf{left} \text{ of } 2j \text{ in } \pi: && \pi^{-1}(2j-1) < \pi^{-1}(2j) \\ &\textbf{If } 2j-1 \text{ appears } \textbf{right} \text{ of } 2j \text{ in } \pi: && \pi^{-1}(2j-1) > \pi^{-1}(2j) \end{aligned}$$

Intuition: For each pair $(2j-1, 2j)$, the *relative order* in the permutation must match the *relative position*.

Notation: W_n = set of all wiggly permutations of $[n]$.

Example: Verifying $\pi = 2413$ is Wiggly

Given: $\pi = 2413$ (so $\pi_1 = 2, \pi_2 = 4, \pi_3 = 1, \pi_4 = 3$)

Inverse function:

- $\pi^{-1}(1) = 3$ (value 1 is at position 3)
- $\pi^{-1}(2) = 1$ (value 2 is at position 1)
- $\pi^{-1}(3) = 4$ (value 3 is at position 4)
- $\pi^{-1}(4) = 2$ (value 4 is at position 2)

Check pair $(1, 2)$ (where $j = 1$):

- Value 1 appears at position 3, value 2 at position 1
- So 2 appears *left* of 1 in the permutation
- Need: $\pi^{-1}(1) > \pi^{-1}(2)$ (right appears right)
- Check: $3 > 1$

Check pair $(3, 4)$ (where $j = 2$):

- Value 3 appears at position 4, value 4 at position 2
- So 4 appears *left* of 3 in the permutation
- Need: $\pi^{-1}(3) > \pi^{-1}(4)$ (right appears right)
- Check: $4 > 2$

Small Examples of Wiggly Permutations

n	W_n (all wiggly permutations)
1	1
2	12
3	132, 231
4	1324, 2143, 2413, 3142, 3412

Observation: $|W_n|$ grows much slower than $n!$

- $|W_1| = 1,\ |W_2| = 1,\ |W_3| = 2,\ |W_4| = 5,\ |W_5| = 16,\ |W_6| = 61$
- **Note:** The full sequence remains unknown and there is no work (Per my knowledge) of a closed form solution for $|W_n|$ as n grows.

Blocks, Jumps and Hops

Block. A block is a contiguous subsequence of a permutation.

Moves used in wiggly flips:

- **Jump:** move a block of length 1 (a single element) past an adjacent block.
- **Hop:** move a block of length 2 (two adjacent elements) past an adjacent block.

Example (n=4):

$$\pi = 1\ 3\ 4\ 2$$

- Jump: move element 4 left past 3 $\Rightarrow$ 1 4 3 2.
- Hop: move block [3 4] left past 1 $\Rightarrow$ 3 4 1 2.

Remark: hops/jumps are minimal legal moves that preserve the wiggly constraints.

Problem Statement

Question: Does there exist a Gray code for W_n using wiggly flips (jumps and hops)?

More precisely: Can we list all wiggly permutations such that:

1. Each permutation appears exactly once
2. Consecutive permutations differ by a single legal wiggly flip (jump or hop)
3. The flip preserves the wiggly property

Equivalently: Does the flip graph of W_n (with edges for legal wiggly flips) have a Hamiltonian path?

Main Theorem

Theorem [2]. For all $n \geq 1$, there exists a Gray code of wiggly permutations (a Hamilton path in the wiggly flip graph).

Moreover, there is an algorithm that:

- Generates all wiggly permutations in Gray code order.
- Uses $\mathcal{O}(n)$ space.
- Runs in worst case $\mathcal{O}(n)$ time per permutation (Conjectured to be $\mathcal{O}(1)$ amortized).
- Is constructive and can be implemented efficiently.

Algorithm F — Greedy edge-label walk (conceptual)

Purpose: Produce a Hamilton path of the wiggly flip graph (existence proof). Algorithm 2 will actually implement this.

Key ingredients:

- Label each cover edge (flip) by the *larger value* involved in the new inversion added.
- From the current permutation, consider all legal wiggly flips (jumps/hops) and their labels.
- **Greedy rule:** always perform an available flip of *maximum* label (deterministic tie-break).

Outcome: The greedy walk (Algorithm F) visits every wiggly permutation exactly once (produces a Hamilton path). (paper: existence theorem)

Short intuition: moving larger values first creates a structured sweep (like Johnson–Trotter) and enforces global invariants that prevent revisiting vertices.

Algorithm 2 / WiggleGray — implementation (loopless style)

Goal: Implement the Algorithm F order as a generator with provable resource bounds.

High level:

- Use the recursive insertion-based strategy (build W_n from W_{n-1}).
- Maintain the set of currently-available flips and their labels incrementally (no full scan).
- At each step pick the maximal-label flip, perform it, update only local candidates.

For theoretical bound guarantees: Refer to the paper.

Algorithm Pseudocode

WiggleGray(n):

1. **Base case:** If $n = 1$, output (1) and return
2. Let $L \leftarrow$ WiggleGray($n - 1$) (recursive call)
3. Initialize direction dir $\leftarrow$ forward
4. **For each** $\pi \in L$ (traversed according to dir):
   1. Compute set P of valid positions for inserting n into π
   2. Order positions in P according to dir
   3. **For each** position $p \in P$:
      - Output: π with n inserted at position p
   4. Toggle: dir $\leftarrow$ (forward $\leftrightarrow$ backward)

Key: Alternating directions ensures outputs connect via single adjacent swaps / wiggly flips.

Example: W_3 Gray Code

Start: $W_2 = \{12\}$

Build W_3 by inserting 3:

- Base permutation: $\pi' = 12$
- Valid positions: need to check wiggly constraints for pair $(2, 3)$
- Try position 2: 132
  - $\pi^{-1}(2) = 3, \pi^{-1}(3) = 2$
- Try position 3: 123
  - $\pi^{-1}(2) = 2, \pi^{-1}(3) = 3$

perm	inv	word	fs	dirs
1234	1234	0000	1234	−−−−
1243	1243	0001	1234	−−−−
1423	1342	0002	1234	−−−−
4123	2341	0003	1233	−−−+
4132	2431	0013	1234	−−−+
1432	1432	0012	1234	−−−+
1342	1423	0011	1233	−−−−
3412	3412	0022	1224	−−+−
4312	3421	0023	1232	−−++
4321	4321	0123	1134	−+++
3421	4312	0122	1133	−+−−
2134	2134	0100	1214	−+−−
2143	2143	0101	1214	−+−−
4213	3241	0103	1231	−+−+

Figure: Wiggly changes of 1234 with the Algorithm visualized [2]

Future Directions

- Tighter analysis of the algorithm will probably yield better average or amortized results.

[1] Asilata Bapat and Vincent Pilaud. “Wigglyhedra”. In: *Mathematische Zeitschrift* 310.3 (2025), p. 54.

[2] Vincent Pilaud and Aaron Williams. “Skipping Ropes: An Efficient Gray Code Algorithm for Generating Wiggly Permutations”. In: *19th International Symposium on Algorithms and Data Structures (WADS 2025)*. Ed. by Pat Morin and Eunjin Oh. Vol. 349. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 46:1–46:20. ISBN: 978-3-95977-398-0. DOI: 10.4230/LIPIcs.WADS.2025.46. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.WADS.2025.46.

[3] Hugo Steinhaus. *One Hundred Problems in Elementary Mathematics*. New York, NY: Courier Corporation, 1979.