Exhaustive Generation of Pattern-Avoiding s-Words

Aaron Williams Williams College

This talk is based on joint work with Samuel Buick, Madeleine Goertz, Amos Lastmann, Kunal Pal, Helen Qian, Sam Tacheny, Leah Williams, and Yulin Zhai (NSF Grant DMS2241623)

We introduce a simple approach to generating Gray codes of pattern-avoiding s-words (i.e., multiset permutations) and corresponding combinatorial objects. It generalizes plain changes and the recent *Combinatorial Generation via Permutation Language* series.

Introduction *Plain changes* is a swap Gray code for S_n the permutations of [n] = {1,...,n}: e.g., 123, 132, 312, 321, 231, 213 where a *swap* moves a large digit past a small digit. It was discovered in the 1600s and is also known as the *Steinhaus-Johnson-Trotter algorithm*. More recently, it has been viewed as a greedy algorithm: "swap the largest value" [5].

A *permutation language* L is a subset of S_n. A *jump* moves a larger digit past **one or more** smaller digits. It is *minimal* for $w \in L$ if its *distance* d (i.e., the number of smaller digits) is minimized to create $w' \in L$. Algorithm J was introduced at *Permutation Patterns 2019*: "minimal jump the largest value" [2]. It generates zig-zag languages e.g., 123, 132, 312, 321, 213 for $\mathsf{Av}_3(231)$. But jumps are limited when working with *s-words* (or *s-permutations*) which have s_i copies of i for $i \in [m]$. For example, $w = 123331$ is a *Stirling s-word* for $s = (2,1,3)$ (i.e., $w \in \mathsf{Av}_s(212)$) and every jump applied to w is *invalid* (i.e., the jump creates a 212 pattern) so the associated flip graph is disconnected. Each $v \in [m]$ is a *value* and each copy of a value in w is a *digit*. The set of all s-words is S_s.

We consider "bumps" which move a **run** of a larger digit. Algorithm B generates many *s-word languages* (i.e., subsets of S_s). Some applications are below and in Figures 1–2.

(a) Gray codes for S_s (e.g., 1122,1221,1212,2112,2121,2211 for $s = (2,2)$) using transpositions.
(b) *Stirling changes* generalizes plain changes to *Stirling s-words* $\mathsf{Av}_s(212)$ using transpositions.
(c) Bump Gray code for regular words counted by k-Catalan numbers $\mathsf{Av}_m^{k-1}(132,121)$ [3, 6].

Our Gray codes lead to efficient algorithms. For example, we generate (b) *looplessly* (i.e., worst-case $O(1)$-time per word) in Algorithm 1. Like the *Permutation Languages* series they also have many applications. For example, (b) leads to a Gray code for s-increasing trees that proves Theorem 1, while (c) gives Gray codes for various k-Catalan objects.

Theorem 1. *Every s-permutohedron has a Hamilton path. (See [1] but with inverted values.)*

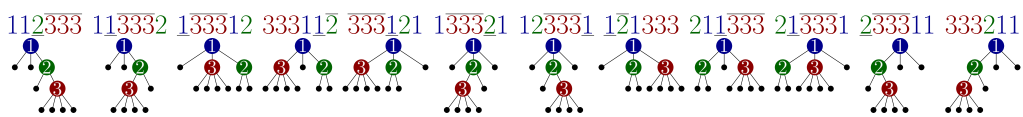


Figure 1: Stirling changes for Stirling s-words with $s = (2,1,3)$ (i.e., permutations of {1,1,2,3,3,3} avoiding 212) as generated by Algorithm B with its corresponding s-increasing tree Gray code.

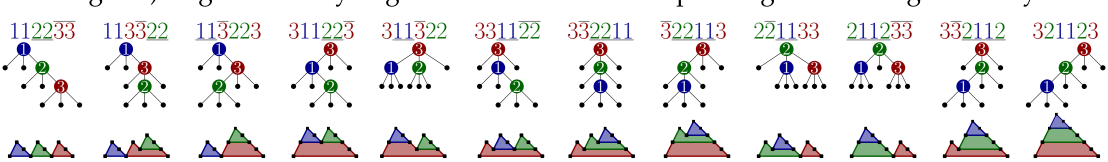


Figure 2: Pattern-avoiding s-words $\mathsf{Av}_s(132,121)$ for $s = (2,2,2)$ with Gray codes for 3-Catalan objects. The ternary trees differ by edge moves preserving inorder (visit self before last child).

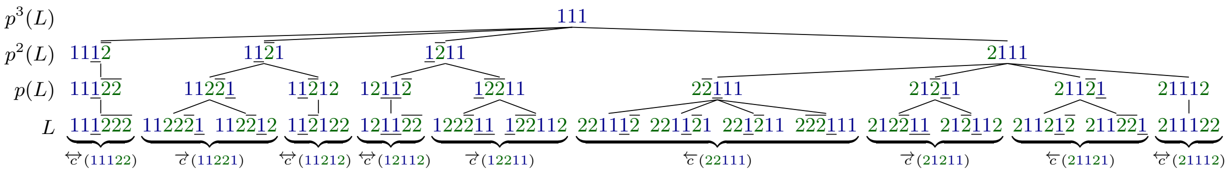


Figure 3: Algorithm B's Gray code for $L = \mathsf{Av}_{3,3}(12121)$ and its ancestor languages. Successive children jump the rightmost largest value while a bump changes the last child of one parent to the first child of the next parent. Note that the rightmost largest value may join these bumps. For example, the jump $2\overline{2}\underline{1}11 = 21211$ in $p(L)$ widens to the bump $2\overline{22}\underline{1}11 = 212211$ in L.

Algorithm B for Bumps Let $w = w_1 w_2 \cdots w_n$ be an s-word with $s = (s_1, s_2, \ldots, s_m)$. The *right-run at index i* is $w_i w_{i+1} \cdots w_j$ with $w_i = \cdots = w_j$ and $j = n$ or $w_j \neq w_{j+1}$. (i.e., a right-maximal run). A *right-bump* moves a right-run to the right past some smaller digits. *Left-run* and *left-bump* are similar. A bump's *value v*, *width w*, *distance d*, and *index i* are its larger digit, # of larger digits, # of smaller digits, and run index. For example, $11\underline{211}\overline{33}322331$ creates $1\overline{33}\underline{1211}333221$ by a left-bump of $v = 3$, $w = 2$, $d = 4$, $i = 7$. Bumps can be jumps ($w = 1$), transpositions ($d = 1$), or swaps ($w = d = 1$).

Let L be an s-word language. Suppose a bump at index i changes $w \in L$ to w'. The bump is *minimal* for its index and direction if it has the shortest distance with $w' \in L$. Minimal bumps on $w \in L$ are uniquely determined by their index and direction. Our generalization of Algorithm J [2] carefully prioritizes values, indices, and directions.

Algorithm *B (Greedy Bumps)*. This algorithm attempts to generate an s-word language L with $s = (s_1, s_2, \ldots, s_m)$ starting from a given or default initial word $w \in L$.
B1. [Initialize] Visit the given initial word w (or by default visit $w = 1^{s_1} 2^{s_2} \cdots m^{s_m}$).
B2. [Greedy] Let w be the most recent word visited. Visit a new word by applying to w a minimal bump prioritized by largest value, then largest index, then rightward over leftward. Halt if no such bump exists. Otherwise, repeat B2.

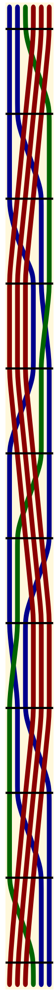
A bump is *maximum* if it uses the longest distance from an index in a direction. A *zig-zag language* is an s-word language closed under maximum bumps (c.f., [2]). This includes $\mathsf{Av}_s(\alpha)$ when α is *tame*: its largest values are *internal* (i.e., not first or last) and *isolated* (i.e., not consecutive). Zig-zag languages are closed under intersection (and union) so the *peakless s-words* $\mathsf{Av}_s(132,231,121)$ are a zig-zag language.

Theorem 2.[1] *If L is zig-zag language of s-words for $s = (s_1, s_2, \ldots, s_m)$, then Algorithm B generates a bump Gray code for L starting from the non-decreasing word $w = 1^{s_1} 2^{s_2} \cdots m^{s_m} \in L$.*
Sketch. Consider the end of an inductive argument on $n = \sum s$. The *parent* $p(s)$ of s is $(s_1, s_2, \ldots, s_m - 1)$ if $s_m > 1$ or $(s_1, s_2, \ldots, s_{m-1})$ if $s_m = 1$. Similarly, $p(w)$ removes the rightmost m from $w \in L$, and $p(L) = \{p(w) \mid w \in L\}$. The *children* of $w' \in p(L)$ are $c(w') = \{w \in L \mid p(w) = w'\}$. Since L is a zig-zag language, $c(w')$ has s-words where the rightmost m is at (a) index n, and (b) index 1 if $s_m = 1$ or beside the rightmost m in w' if $s_m > 1$; these extremes are equal when w' ends in m. Let $\overrightarrow{c}(w')$ list $c(w')$ in lexicographic order (i.e., the rightmost m jumps left-to-right) and $\overleftarrow{c}(w')$ in reverse; use $\overleftrightarrow{c}(w')$ when w' has one child. Note that $p(L)$ is a zig-zag language so at some point Algorithm B generates w'. We claim that Algorithm B generates $\overrightarrow{c}(w')$ or $\overleftarrow{c}(w')$ (or $\overleftrightarrow{c}(w')$) when run on L. It also applies a bump from the last child of one parent to the first child of the next parent. In other words, it uses *local recursion*. See Figure 3. □

[1] This is not a full characterization as Algorithm B generates some non-zig-zag language e.g., $\mathsf{Av}_s(212)$.

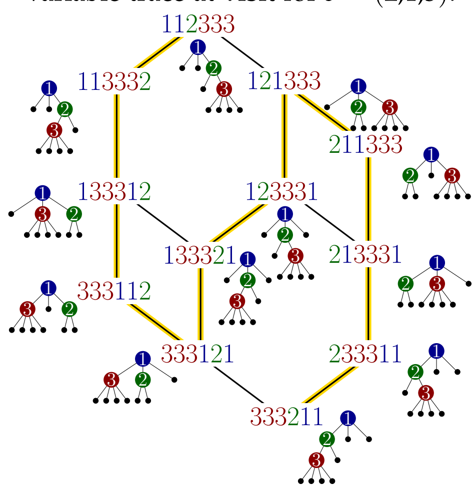
Algorithm 1 Loopless generation of Stirling s-words $\mathrm{Av}_s(212)$ in Stirling changes order.

```
def Stirling(s1, s2, ..., sm)            ▷ 1-based indexing
  t1, t2, t3, ..., tm ← 0, s1, s1+s2, ..., s1+···+s(m−1)
  perm ← 1^s1 2^s2 ··· m^sm               ▷ s-word (s-perm)
  left ← [t1+1, ..., t(m−1)+1]            ▷ left indices 1,...,m
  inv ← 0^m   ▷ # of smaller digits right of 1,2,...,m
  fs ← 12···m ▷ focus pointers for 1,...,m (see [4])
  dirs ← −1^m                             ▷ bump directions for 1,...,m
  v ← fs[m]                               ▷ larger value in next bump
  while v > 1 do                          ▷ 1 is never the larger value
    d ← dirs[v]                           ▷ direction of v
    if d = 1 then                         ▷ right-bump v's run?
      i ← left[v]                         ▷ first index of v
      j ← left[v] + s[v]                  ▷ index after v's run
    else                                  ▷ left-bump v's run
      i ← left[v] + s[v] − 1              ▷ last index of v
      j ← left[v] − 1                     ▷ index before v's run
    u ← perm[j]                           ▷ smaller digit to bump
    visit perm                            ▷ next s-word in Gray code
    perm[i] ← u                           ▷ apply bump to the s-word
    perm[j] ← v                           ▷ (as a transposition)
    left[v] ← left[v]+d                   ▷ leftmost v moved
    if left[u] = j then                   ▷ leftmost u moved?
      left[u] ← left[u]−d·s_v             ▷ u passed all v
    inv[v] ← inv[v]−d                     ▷ v's run passed one u
    if inv[v] = 0 or inv[v] = t_v then    ▷ v limit?
      dirs[v] ← −d                        ▷ change v's direction
      fs[v] ← fs[v−1]                     ▷ inherit focus of v−1
      fs[v−1] ← v−1                       ▷ reset focus of v−1
    v ← fs[m]                             ▷ larger value in next bump
    fs[m] ← m                             ▷ reset focus of m
  visit perm                              ▷ last s-word in Gray code
```



perm	v	u	i	j	left	inv	fs	dirs
112333	3	2	6	3	134	000	123	---
113332	3	1	5	2	163	001	123	---
133312	3	1	4	1	162	002	123	---
333112	2	1	6	5	461	003	123	--+
333121	3	1	1	4	451	013	123	--+
133321	3	2	2	5	152	012	123	--+
123331	3	1	3	6	123	011	123	--+
121333	2	1	2	1	124	010	123	---
211333	3	1	6	3	214	020	113	-+-
213331	3	1	5	2	213	021	113	-+-
233311	3	2	4	1	512	022	113	-+-
333211	1				541	023	123	-++

Variable trace at **visit** for $s = (2,1,3)$.



Hamilton path (2,1,3)-permutohedron (or $s = (3,1,2)$ for decreasing trees [1]).

References

[1] Cesar Ceballos and Viviane Pons. The s-weak order and s-permutahedra. In *31st International Conference on Formal Power Series and Algebraic Combinatorics*, 2019.

[2] Elizabeth Hartung, Hung Hoang, Torsten Mütze, and Aaron Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Transactions of the American Mathematical Society*, 375(4):2255–2291, 2022.

[3] Markus Kuba and Alois Panholzer. Enumeration formulae for pattern restricted Stirling permutations. *Discrete Mathematics*, 312(21):3179–3194, 2012.

[4] Yuan Qiu and Aaron Williams. Generating signed permutations by twisting two-sided ribbons. In *LATIN 2024 (LNCS 14578)*, pages 114–129. Springer, 2024.

[5] Aaron Williams. The greedy Gray code algorithm. In *Workshop on Algorithms and Data Structures*, pages 525–536. Springer, 2013.

[6] Aaron Williams. Pattern avoidance for k-Catalan sequences. In *Proceedings of the 21st International Conference on Permutation Patterns*, pages 147–149, 2023.