PU. M. A. Vol. **22** (2011), No.2, pp. 253 – 268

Computational approaches to consecutive pattern avoidance in permutations

Brian Nakamura
Mathematics Department
Rutgers University
New Brunswick, Piscataway, NJ, USA
e-mail: bnaka@math.rutgers.edu

(Received: February 7, 2011, and in revised form June 8, 2011)

Abstract. In recent years, there has been increasing interest in consecutive pattern avoidance in permutations. In this paper, we introduce two approaches to counting permutations that avoid a set of prescribed patterns consecutively. These algorithms have been implemented in the accompanying Maple package `CAV`, which can be downloaded from the author's website. As a byproduct of the first algorithm, we have a theorem giving a sufficient condition for when two pattern sets are strongly (consecutively) Wilf-equivalent. For the implementation of the second algorithm, we define the cluster tail generating function and show that it always satisfies a certain functional equation. We also explain how the `CAV` package can be used to approximate asymptotic constants for single pattern avoidance.

Mathematics Subject Classification(2010). 05A05.

Keywords: permutation, consecutive pattern, enumeration.

1 Introduction

Let $\sigma = \sigma_1 \cdots \sigma_k$ be a sequence of k distinct positive integers. We define the *reduction* $\mathrm{red}(\sigma)$ to be the length k permutation obtained by relabeling the elements of σ with $\{1, \ldots, k\}$ so that σ and $\mathrm{red}(\sigma)$ are order-isomorphic. For example, $\mathrm{red}(5386) = 2143$. For a permutation π, we will also write $|\pi|$ for the number of elements in the permutation. Let m and n be positive integers with $m \leq n$, and let $\pi \in \mathcal{S}_m$ and $\sigma = \sigma_1 \cdots \sigma_n \in \mathcal{S}_n$. We say that σ *contains* π *consecutively* if $\mathrm{red}(\sigma_i \cdots \sigma_{i+m-1}) = \pi$ for some i where $1 \leq i \leq n - m + 1$. Otherwise, we say that σ *avoids* π *consecutively*. Similarly, if B is a set of permutations, then we say that σ *avoids* B *consecutively* if for every $\pi \in B$, σ avoids the pattern π consecutively. For example, the permutation $123654 \in \mathcal{S}_6$ contains the permutation pattern 1243, since $\mathrm{red}(2365) = 1243$. However, the permutation $12453 \in \mathcal{S}_5$ avoids the pattern 1243 consecutively.

In general, we are interested in counting permutations that avoid a pattern (or a set of patterns). Given a set of patterns B, let $\alpha_n(B)$ be the number of length n permutations that avoid B consecutively. If B consists of only a single pattern π, we may write $\alpha_n(\pi)$ instead, and if no ambiguity would arise, we may just write α_n. For a given set of patterns B, we would like to find the exponential generating function

$$A_B(z) = \sum_{n=0}^{\infty} \alpha_n \frac{z^n}{n!}. \tag{1}$$

If no ambiguity would arise, this may also be denoted by $A(z)$. In addition, we define a more general exponential generating function

$$P_B(z,t) = \sum_{k,n\geq 0} b_{k,n} \frac{z^n t^k}{n!} \tag{2}$$

where $b_{k,n}$ is the number of length n permutations that contain exactly k occurrences of the patterns in B. Again, we may write $P(z,t)$ if the set B is clear. We will also define $\alpha_n(t) = \sum_{k\geq 0} b_{k,n} t^k$. Note that $P(z,0) = A(z)$ and $\alpha_n(0) = \alpha_n$.

In addition, we will say that two sets of patterns B and B' are consecutively Wilf-equivalent (sometimes written c-Wilf-equivalent) if $A_B(z) = A_{B'}(z)$. We will also say that B and B' are strongly c-Wilf-equivalent if $P_B(z,t) = P_{B'}(z,t)$. Since this paper deals solely with consecutive patterns, the word "consecutive" will be omitted in most instances. For the rest of this paper, the reader should assume that all mentions of containment, avoidance, and Wilf-equivalence are consecutive.

In recent years, there has been an increasing amount of research done on consecutive pattern avoidance in permutations. An early paper of Elizalde and Noy [6] finds generating functions $A(z)$ and $P(z,t)$ for certain cases of single pattern avoidance. Using various techniques, additional generating functions for specific single patterns and multi-pattern sets have been found in [1, 3, 9, 10]. There is also recent work in dashed patterns (a generalization of consecutive patterns) using enumeration schemes in [2]. In particular, our approach will resemble the cluster method approach in [3].

The results in this paper utilize an extension of the Goulden-Jackson cluster method [7, 11]. We restate some of the terminology and notation here.

Let B be a set of patterns. Without loss of generality, assume that B contains no trivial redundancies (i.e., there are no $\pi_1, \pi_2 \in B$ with $\pi_1 \neq \pi_2$ such that π_1 contains π_2). We say that an ordered pair $(\pi; [[i_1, j_1], \ldots, [i_m, j_m]])$ is a length k cluster if it satisfies the following:

(a) $\pi \in \mathcal{S}_k$

(b) $i_1 = 1$, $j_m = k$, and $i_n < i_{n+1} < j_n$ for $1 \leq n \leq m-1$ (i.e., each interval overlaps with the neighboring interval, and the intervals cover π)

(c) $\mathrm{red}(\pi_{i_n} \cdots \pi_{j_n}) \in B$ for all $1 \leq n \leq m$.

Given a cluster $(\pi; [[i_1, j_1], \ldots, [i_m, j_m]])$, we will refer to $\mathrm{red}(\pi_{i_n} \cdots \pi_{j_n})$ as the n-th *marked pattern* in the cluster. Let $\mathcal{C}_k$ be the set of clusters of length k, and for a cluster $w = (\pi; [[i_1, j_1], \ldots, [i_m, j_m]])$, define $weight(w) = (t-1)^m$, where t will be the variable used to track occurrences. Let $C(k) = \sum_{w \in \mathcal{C}_k} weight(w)$. From an adaptation of [7] to the present context of an "infinite" alphabet and exponential generating functions, we have:

THEOREM 1.1

$$P(z,t) = \frac{1}{1 - z - \sum_{k \geq 1} C(k) \frac{z^k}{k!}} \tag{3}$$

In this paper, we will derive a recurrence from this theorem and use this recurrence as the basis for our subsequent algorithms and results.

So far, generating functions have been found for specific single patterns and multi-pattern sets and for certain single pattern families where some specific structure can be exploited. In this paper, we will outline two algorithms to calculate α_n and $\alpha_n(t)$ more efficiently, and both algorithms have been implemented in the accompanying Maple package `CAV`. The Maple package can be downloaded from the author's website. As a result of the first algorithm in Section 2, we get a theorem for proving when two pattern sets are strongly c-Wilf-equivalent. During preparation of this paper, the author learned that this result was also independently proven by Khoroshkin and Shapiro in [8] by slightly different means. To establish the much faster second algorithm in Section 3, we define a new generating function which we refer to as the cluster tail generating function. We show that this generating function always satisfies a certain functional equation and give a constructive approach to finding it. This functional equation is then used to compute values for α_n much more quickly. We use our algorithm to give some asymptotic approximations in Section 4. We conclude with Section 5 by sharing some new conjectures we have based off of experimentation with our `CAV` package. Beyond the theorems and results in this paper though, we hope that the `CAV` Maple package will be a useful tool for others in studying consecutive pattern avoidance in permutations.

2 Consecutive Pattern Avoidance via Clusters

Let B be a set of patterns that we would like to avoid (consecutively). For the rest of this paper, we assume that B contains no redundancies (i.e., there does not exist $p_1, p_2 \in B$ with $p_1 \neq p_2$ such that p_1 contains p_2). Again, α_n will be the number of length n permutations avoiding B, and $\alpha_n(t)$ will be the polynomial in t where the coefficient of t^k is the number of length n permutations with exactly k occurrences of patterns in B.

From the Goulden-Jackson cluster method (Theorem 1.1), we can get the equation

$$P(z,t) = 1 + zP(z,t) + P(z,t) \sum_{k \geq 1} C(k) \frac{z^k}{k!} \tag{4}$$

and by extracting the coefficients of z^n, we get the following recurrence:

$$\alpha_n(t) = n\alpha_{n-1}(t) + \sum_{k=1}^{n} \binom{n}{k} C(k) \alpha_{n-k}(t) \tag{5}$$

Additionally, consider a fixed $p \in B$ and let $m = |p|$. Let $\mathcal{C}_k[p] = \{(\pi; [[i_1, j_1], \ldots, [i_r, j_r]]) \in \mathcal{C}_k : \text{red}(\pi_{i_r} \cdots \pi_{j_r}) = p\}$, the set of length k clusters ending in the pattern p. Let $\mathcal{C}_k[p; [x_1, \ldots, x_m]]$ be the clusters in $\mathcal{C}_k[p]$ with the last m terms $\{x_1, \ldots, x_m\}$, where $x_1 < x_2 < \ldots < x_m$. Similarly, define

$$C(k, p) = \sum_{w \in \mathcal{C}_k[p]} weight(w) \tag{6}$$

$$C(k, p; [x_1, \ldots, x_m]) = \sum_{w \in \mathcal{C}_k[p; [x_1, \ldots, x_m]]} weight(w). \tag{7}$$

If B contains only one pattern, these may be denoted by $C(k)$ and $C(k; [x_1, \ldots, x_m])$, respectively. We will use Equation (5) to compute α_n and, more generally, $\alpha_n(t)$.

2.1 General Algorithm

Computationally, the difficulty in using Equation (5) lies in calculating $C(k)$ quickly. One way to do this is to create a recurrence for $C(k, p; [x_1, \ldots, x_{|p|}])$ for each $p \in B$.

We can do this as follows: for a given cluster w, let p_1 and p_2 be the last marked pattern and the second to last marked pattern in w, respectively. Let j be the length of the overlap of p_1 and p_2 in w, i.e., the tail of length j of p_2 coincides with the head of length j of p_1. We want to "chop off" the last $|p_1| - j$ terms of w and apply the reduction to get a shorter cluster, say $w' \in \mathcal{C}_{k'}[p_2; [x_1, \ldots, x_{|p_2|}]]$, which ends in the pattern p_2. Then, $weight(w) = weight(p_1) \cdot weight(w')$.

Additionally, once we "chop off" the tail of p_1 and apply the reduction to get a shorter cluster w', we actually know what the last j terms of $w' = w'_1 \cdots w'_{k'}$ will be. For each term w'_i with $|w'| - j + 1 \leq i \leq |w'|$, the reduction forces w'_i to be $w_i - (\#$ of terms in w "chopped off" that were less than $w_i)$. Thus, to compute $C(k, p_1; [x_1, \ldots, x_{|p_1|}])$, we need to sum over all possible ways to "fill out" the rest of the terms in the final p_2 pattern of w'. We also need to sum over all possible choices of $p_2 \in B$ and all possible ways that the tails of this p_2 overlap with the heads of the final p_1 pattern.

In summary, the number of length n permutations avoiding set B can be found by, first, generating a cluster recurrence for $C(k, p; [x_1, \ldots, x_{|p|}])$ for each $p \in B$. Next, use the recurrence $\alpha_n = n\alpha_{n-1} + \sum_{k=1}^{n} \binom{n}{k} C(k)\alpha_{n-k}$ using the base cases $\alpha_0 = \alpha_1 = 1$ and $\alpha_n = 0$ if $n < 0$. Use the cluster recurrences to compute $C(k)$ as needed:

$$C(k) = \sum_{p \in B} C(k, p) \tag{8}$$

$$C(k, p) = \sum_{1 \leq x_1 < x_2 < \ldots < x_m \leq k} C(k, p; [x_1, \ldots, x_m]) \tag{9}$$

Also recall that if $w = (\pi; [i_1, j_1], \ldots, [i_m, j_m])$, then $weight(w) = (t-1)^m$ will keep track of occurrences of patterns with variable t, while setting $t = 0$ and using $weight(w) = (-1)^m$ would count only the permutations that avoid the designated pattern set B.

2.2 Example

Let $B = \{2143\}$. Let $w = (\pi; [[i_1, j_1], \ldots, [i_m, j_m]])$ be a length k cluster and $\{x_1, \ldots, x_4\}$ be the last 4 terms of w with $x_1 < \ldots < x_4$ (i.e., $\pi_{k-3} = x_2$, $\pi_{k-2} = x_1$, $\pi_{k-1} = x_4$, and $\pi_k = x_3$). Then, the second to last pattern must also be a 2143 pattern and can have an overlap of length 1 or 2 with the last pattern.

If the overlap is of length 2, let $\pi' = \text{red}(\pi_1 \cdots \pi_{k-2})$ and $w' = (\pi'; [[i_1, j_1], \ldots, [i_{m-1}, j_{m-1}]])$, the cluster found by "chopping off" the tail of the final bad pattern in w and then canonically reducing. Now let $\{y_1, \ldots, y_4\}$ be the last 4 terms of w' with $y_1 < \ldots < y_4$ (i.e., $\pi'_{k-3} = y_2$, $\pi'_{k-2} = y_1$, $\pi'_{k-1} = y_4$, and $\pi'_k = y_3$). Notice that the terms "chopped off" from w were x_4 and x_3. Since both of these are larger than both x_2 and x_1, applying the reduction does not change their values. Thus, $y_4 = x_2$ and $y_3 = x_1$. Summing over all possible tails for w' and accounting for the last pattern that was removed from w, we get

$$\sum_{\substack{1 \le y_1 < \ldots < y_4 \le k-2 \\ y_3 = x_1 \\ y_4 = x_2}} weight(2143) \cdot C(k-2; [y_1, y_2, y_3, y_4]). \tag{10}$$

If the overlap is of length 1, let $\pi' = \text{red}(\pi_1 \cdots \pi_{k-3})$ and $w' = (\pi'; [[i_1, j_1], \ldots, [i_{m-1}, j_{m-1}]])$, since the tail that gets "chopped off" has 3 terms. Again, let $\{y_1, \ldots, y_4\}$ be the last 4 terms of w' with $y_1 < \ldots < y_4$ (i.e., $\pi'_{k-3} = y_2$, $\pi'_{k-2} = y_1$, $\pi'_{k-1} = y_4$, and $\pi'_k = y_3$). The terms "chopped off" from w are x_1, x_4, and x_3. Since exactly one term less than x_2 (only x_1) was removed, applying the reduction would reduce x_2 by 1. Thus, $y_3 = x_2 - 1$. Summing over all possible tails for w' and accounting for the last pattern that was removed from w, we get

$$\sum_{\substack{1 \le y_1 < \ldots < y_4 \le k-3 \\ y_3 = x_2 - 1}} weight(2143) \cdot C(k-3; [y_1, y_2, y_3, y_4]). \tag{11}$$

We combine the two possibilities along with the base cases to get the recurrence.

For $k < 4$:

$$C(k; [x_1, x_2, x_3, x_4]) = 0$$

For $k = 4$:

$$C(k; [x_1, x_2, x_3, x_4]) = weight(2143)$$

For $k > 4$:

$$C(k;[x_1,x_2,x_3,x_4]) = \sum_{\substack{1\le y_1<\ldots<y_4\le k-3\\ y_3=x_2-1}} weight(2143)\cdot C(k-3;[y_1,y_2,y_3,y_4]) + \sum_{\substack{1\le y_1<\ldots<y_4\le k-2\\ y_3=x_1\\ y_4=x_2}} weight(2143)\cdot C(k-2;[y_1,y_2,y_3,y_4]) \tag{12}$$

Using this recurrence, we can compute $C(k)$ for any value of k and compute α_n using Equation (5). To keep track of all occurrences of 2143 with the variable t, let $weight(2143) = t-1$. To only count permutations that avoid 2143, set $t = 0$ so that $weight(2143) = -1$ for the above recurrence.

2.3 Results for c-Wilf-equivalence

Even though Section 2.1 is algorithmic in nature, it yields a strong theoretical byproduct. The cluster recurrence generated by the pattern set B totally determines α_n. In fact, it also totally determines $P(z,t)$. However, the "overlapping" relations between the patterns in B totally determine the cluster recurrence.

More specifically, let B be the set of patterns we want to avoid, and let $\pi,\sigma \in B$ where $m = |\pi|$ and $n = |\sigma|$. Note that π and σ are not necessarily distinct. Suppose that $\mathrm{red}(\sigma_{n-j+1}\cdots\sigma_n) = \mathrm{red}(\pi_1\cdots\pi_j)$ (the tail of σ and the head of π has an overlap of length j). Then, define the following sets:

$$OverlapMap(\sigma,\pi,j) = \{(\pi_1,\sigma_{n-j+1}),(\pi_2,\sigma_{n-j+2}),\ldots,(\pi_j,\sigma_n)\} \tag{13}$$

$$OverlapMaps(\sigma,\pi) = \{OverlapMap(\sigma,\pi,j) : \mathrm{red}(\sigma_{n-j+1}\cdots\sigma_n) = \mathrm{red}(\pi_1\cdots\pi_j)\} \tag{14}$$

For example, in Section 2.2, the pattern 2143 has self-overlaps of length 1 and 2. For a length 1 overlap, we have $OverlapMap(2143,2143,1) = \{(2,3)\}$. This combined with the length of the pattern, which is 4, and the length of the original cluster, denoted by k, completely determines the first summation in Equation (12). Similarly, for a length 2 overlap, we have $OverlapMap(2143,2143,2) = \{(2,4),(1,3)\}$. Combining this with the length of the pattern, again 4, and the length of the original cluster, again k, completely determines the second summation in Equation (12). Therefore, $OverlapMaps(2143,2143) = \{\{(2,3)\},\{(2,4),(1,3)\}\}$ and $|2143| = 4$ completely determines the cluster recurrence.

Therefore we have the following result based off of our algorithm:

THEOREM 2.1 *Let B and B' be two sets of patterns with $|B| = |B'|$. Suppose there is some labeling of the elements (patterns) in sets B and B', say $B = \{p_1, \ldots, p_k\}$ and $B' = \{p'_1, \ldots, p'_k\}$, such that $|p_i| = |p'_i|$ for $1 \leq i \leq k$, and $OverlapMaps(p_i, p_j) = OverlapMaps(p'_i, p'_j)$ for all $1 \leq i, j \leq k$. Then, B and B' are strongly c-Wilf-equivalent.*

Proof. The cluster recurrence was uniquely determined by how the patterns overlapped (which terms from one pattern overlapped with which terms of another pattern) and by how they reduced after "chopping" the last pattern from the current cluster. The possible ways that two patterns can overlap are encoded by OverlapMaps and the effect of the reduction is determined by how the patterns overlapped and the length of those patterns. □

This result was also independently discovered by Khoroshkin and Shapiro [8].

Using this result, it is possible to classify c-Wilf-equivalences in some cases. For example, it is possible to classify single pattern avoidance for single patterns of length 3, 4, and 5 since all the potential equivalences that occur can be demonstrated using Theorem 2.1. Using the same approach, we can classify almost all single patterns of length 6. All that remains are four possible strong c-Wilf-equivalences that appear true but cannot be rigorously proven through our means. They are the following:

(1) The pattern 123546 appears to belong to the strong c-Wilf-equivalence class $\{124536, 125436\}$.

(2) The pattern 123645 appears to belong to the strong c-Wilf-equivalence class $\{124635, 126435\}$.

(3) The patterns 132465 and 142365 appear to be strongly c-Wilf-equivalent.

(4) The patterns 154263 and 165243 appear to be strongly c-Wilf-equivalent.

The four cases have been experimentally verified for up to length 18 permutations. In total, there are between 92 and 96 c-Wilf-equivalence classes (inclusively) for single patterns of length 6. Other than the conjectured patterns above, every pair of c-Wilf-equivalent patterns is in fact strongly c-Wilf-equivalent. More detailed information (including all the equivalences mentioned above) can be found on the author's website.

2.4 Maple Implementation

The algorithm from Section 2.1 has been implemented in the Maple package `CAV`. Using that algorithm and given a set of patterns B to avoid, you can find the sequence $\alpha_1, \ldots, \alpha_n$ by calling the procedure `CAV(B,n)`, where the patterns in B are represented as lists. For example, for $n = 10$ and $B = \{123, 321\}$, trying `CAV({[1,2,3],[3,2,1]},10);` returns the output:

$$[1, 2, 4, 10, 32, 122, 544, 2770, 15872, 101042]$$

To keep track of the occurrences of patterns from B, use the procedure `CAVt(B,n,t)`. For example, trying `CAVt({[1,2,3],[3,2,1]},6,t);` returns the output:

$$[1, 2, 4 + 2t, 10 + 12t + 2t^2, 58t + 28t^2 + 32 + 2t^3, 300t + 236t^2 + 122 + 60t^3 + 2t^4]$$

Also, most of the main procedures in the Maple `CAV` package have an optional verbose setting. For example, for the verbose outputs, try `CAV({[1,2,3],[3,2,1]},10,true);`.

To generate the cluster recurrence only (encoded in a data structure that we call a cluster scheme), use the procedure `SCHEME(k,B,x,y,t)`. For example, try `SCHEME(k, {[1,2,3], [3,2,1]},x,y,t);`. Note that our cluster schemes are different from enumeration schemes that currently appear in permutation patterns literature.

The overlap maps between two patterns can also be found using `OverlapMaps(p1,p2)`, which checks for overlaps between tails of `p1` with heads of `p2`. For example, try `OverlapMaps([2,1,4,3], [2,1,4,3])`.

To (attempt to) classify pattern sets of m patterns with each pattern length n, we can compute α_N (for some fixed value N) for each of these pattern sets, and if the α_N values coincide, try to apply Theorem 2.1. This has been implemented in the procedure `WilfEqm(n,N,m)`. For example, try `WilfEqm(5,12,1)` (or for the verbose output, `WilfEqm(5,12,1,true)`) to (rigorously) classify c-Wilf-equivalence for all single patterns of length 5. An additional byproduct of Theorem 2.1 is that all instances of c-Wilf-equivalence in single length 5 patterns are actually strong c-Wilf-equivalence. The 25 c-Wilf-equivalence classes can be found on the paper's website.

Similarly, we can use the `WilfEqm` procedure to discover the following:

PROPOSITION 2.2 *Let B_1 and B_2 both be sets containing two patterns of length 3. Then B_1 is c-Wilf-equivalent to B_2 if and only if they are trivially equivalent by reversal and/or complementation.*

Proof. Run "`WilfEqm(3,10,2,true);`" using the `CAV` Maple package. □

PROPOSITION 2.3 *Let B_1 and B_2 both be sets containing two patterns of length 4. Then B_1 is c-Wilf-equivalent to B_2 if and only if they are trivially equivalent by reversal and/or complementation.*

Proof. Run "`WilfEqm(4,10,2,true);`" using the `CAV` Maple package. □

PROPOSITION 2.4 *Let B_1 and B_2 both be sets containing three patterns of length 3. Then B_1 is c-Wilf-equivalent to B_2 if and only if they are trivially equivalent by reversal and/or complementation.*

Proof. Run "`WilfEqm(3,10,3,true);`" using the `CAV` Maple package. □

Similarly, nearly all c-Wilf-equivalences could be classified for sets containing three patterns of length 4. Four pairs of sets appear c-Wilf-equivalent but cannot be proven through our means. They are the following:

(1) The pattern sets $\{1234, 1243, 1342\}$ and $\{1234, 1243, 1432\}$ appear to be strongly c-Wilf-equivalent.

(2) The pattern sets $\{1234, 1243, 2341\}$ and $\{1234, 1243, 2431\}$ appear to be strongly c-Wilf-equivalent.

(3) The pattern sets $\{1324, 1342, 1423\}$ and $\{1324, 1423, 1432\}$ appear to be strongly c-Wilf-equivalent.

(4) The pattern sets $\{1324, 1423, 2341\}$ and $\{1324, 1423, 2431\}$ appear to be strongly c-Wilf-equivalent.

The four cases have been experimentally verified for up to length 14 permutations, and the rest of the classification can be found on the paper's website.

3 Consecutive Pattern Avoidance via the Cluster Tail Generating Function

Computationally, the cluster recurrence is faster than the naive approach of checking every single permutation, but the approach is still very inefficient. For a fixed length k, not every combination of tails gives rise to a possible cluster. For example, if $B = \{123\}$, the only possible underlying permutation in a length 9 cluster is 123456789. The only possible tail is 789, but using the recurrence, we essentially try all $\binom{9}{3}$ possible tails. Each such possible tail gives its contribution of 0 only after it has recursed down to the base cases of $k \leq 3$.

We can, however, gain a substantial speed-up by considering a more complicated generating function. For a fixed pattern $p \in B$ with length m, the cluster tail generating function will be defined as:

$$F(k, p; [z_1, \ldots, z_m]) = \sum_{1 \leq x_1 < \ldots < x_m \leq k} C(k, p; [x_1, \ldots, x_m]) z_1^{x_1} \cdots z_m^{x_m} \tag{15}$$

If B is a single pattern set, this may also be denoted as $F(k; [z_1, \ldots, z_m])$. Otherwise, we also define:

$$F(k; [z_1, \ldots, z_m]) = \sum_{p \in B} F(k, p; [z_1, \ldots, z_m]) \tag{16}$$

Note that $F(k, p; [1, \ldots, 1]) = C(k, p)$ and $F(k; [1, \ldots, 1]) = C(k)$. In fact, we can always find a functional equation for $F(k, p; [z_1, \ldots, z_m])$ of a certain form. We can then combine this with Equation (5) to more quickly compute α_n. We begin with an illustrative example and then present the general algorithm.

3.1 Example

Let $B = \{132\}$ and suppose we want to only count permutations that completely avoid 132. We will set $t = 0$ which gives us $weight(132) = -1$. We then can find a functional equation for $F(k; [z_1, z_2, z_3])$ as follows. Using the procedure `SCHEME` in the Maple package `CAV`, we can get the following cluster recurrence:

$$\begin{aligned} C(k; [x_1, x_2, x_3]) &= -\sum_{\substack{1 \le y_1 < y_2 < y_3 \le k-2 \\ y_2 = x_1}} C(k-2; [y_1, y_2, y_3]) \\ &= -\sum_{\substack{1 \le y_1 < x_1 \\ x_1 < y_3 \le k-2}} C(k-2; [y_1, x_1, y_3]) \end{aligned}$$

with the base cases $C(k; [x_1, x_2, x_3]) = 0$ if $k < 3$ and $C(k; [x_1, x_2, x_3]) = -1$ if $k = 3$. Substituting into Equation (15) and applying the finite geometric series formula as needed, we get:

$$\begin{aligned} F(k; [z_1, z_2, z_3]) &= \sum_{1 \le x_1 < x_2 < x_3 \le k} C(k; [x_1, x_2, x_3]) z_1^{x_1} z_2^{x_2} z_3^{x_3} \\ &= -\sum_{x_1=1}^{k-2} \sum_{x_2=x_1+1}^{k-1} \sum_{x_3=x_2+1}^{k} \sum_{\substack{1 \le y_1 < x_1 \\ x_1 < y_3 \le k-2}} C(k-2; [y_1, x_1, y_3]) z_1^{x_1} z_2^{x_2} z_3^{x_3} \\ &= -\sum_{x_1=1}^{k-2} \sum_{x_2=x_1+1}^{k-1} \sum_{\substack{1 \le y_1 < x_1 \\ x_1 < y_3 \le k-2}} C(k-2; [y_1, x_1, y_3]) z_1^{x_1} z_2^{x_2} \sum_{x_3=x_2+1}^{k} z_3^{x_3} \\ &= -\frac{z_3}{1-z_3} \sum_{x_1=1}^{k-2} \sum_{x_2=x_1+1}^{k-1} \sum_{\substack{1 \le y_1 < x_1 \\ x_1 < y_3 \le k-2}} C(k-2; [y_1, x_1, y_3]) z_1^{x_1} z_2^{x_2} (z_3^{x_2} - z_3^{k}) \\ &= -\frac{z_3}{1-z_3} \sum_{x_1=1}^{k-2} \sum_{\substack{1 \le y_1 < x_1 \\ x_1 < y_3 \le k-2}} C(k-2; [y_1, x_1, y_3]) z_1^{x_1} \sum_{x_2=x_1+1}^{k-1} z_2^{x_2} (z_3^{x_2} - z_3^{k}) \end{aligned}$$

and since

$$\sum_{x_2=x_1+1}^{k-1} z_2^{x_2} (z_3^{x_2} - z_3^{k}) = \left(\frac{(z_2 z_3)^{x_1+1} - (z_2 z_3)^{k}}{1 - z_2 z_3} - z_3^{k} \frac{z_2^{x_1+1} - z_2^{k}}{1 - z_2} \right)$$

we get

$$
\begin{aligned}
F(k;[z_1,z_2,z_3]) &= -\frac{z_2z_3^2}{(1-z_3)(1-z_2z_3)}\sum_{x_1=1}^{k-2}\sum_{\substack{1\le y_1<x_1\\ x_1<y_3\le k-2}} C(k-2;[y_1,x_1,y_3])(z_1z_2z_3)^{x_1}\\
&\quad+\frac{z_2^kz_3^{k+1}}{(1-z_3)(1-z_2z_3)}\sum_{x_1=1}^{k-2}\sum_{\substack{1\le y_1<x_1\\ x_1<y_3\le k-2}} C(k-2;[y_1,x_1,y_3])z_1^{x_1}\\
&\quad+\frac{z_2z_3^{k+1}}{(1-z_3)(1-z_2)}\sum_{x_1=1}^{k-2}\sum_{\substack{1\le y_1<x_1\\ x_1<y_3\le k-2}} C(k-2;[y_1,x_1,y_3])(z_1z_2)^{x_1}\\
&\quad-\frac{z_2^kz_3^{k+1}}{(1-z_3)(1-z_2)}\sum_{x_1=1}^{k-2}\sum_{\substack{1\le y_1<x_1\\ x_1<y_3\le k-2}} C(k-2;[y_1,x_1,y_3])z_1^{x_1}\\
&= -\frac{z_2z_3^2}{(1-z_3)(1-z_2z_3)}F(k-2;[1,z_1z_2z_3,1])\\
&\quad+\frac{z_2^kz_3^{k+1}}{(1-z_3)(1-z_2z_3)}F(k-2;[1,z_1,1])\\
&\quad+\frac{z_2z_3^{k+1}}{(1-z_3)(1-z_2)}F(k-2;[1,z_1z_2,1])\\
&\quad-\frac{z_2^kz_3^{k+1}}{(1-z_3)(1-z_2)}F(k-2;[1,z_1,1]).
\end{aligned}
$$

We can then use the functional equation to compute $C(k) = F(k;[1,1,1])$ for whatever k we need and then find α_n for the desired n by Equation (5).

3.2 General Algorithm

In general, if we can find a functional equation for $F(k,p;[z_1,\ldots,z_m])$ that relates it to cluster generating functions with lower order k', we can use it to compute $\alpha_n(t)$ using Equation (5). One can see that most of what was done in the above example can be extended to any pattern (or pattern set by finding a functional equation for each pattern individually). The outline of the general procedure is as follows:

First, find the cluster recurrence for the initial summand $C(k,p;[x_1,\ldots,x_m])$ (as in Section 2.1) and substitute this into the summation in Equation (15). Split the summation over each summand $C(k',p';[y_1,\ldots,y_{m'}])$, and handle each one separately. Rewrite the summations over $x_1,\ldots,x_m$ and apply the finite geometric series formula as needed. Finally, express the remaining summations as cluster tail generating functions of lower order k'.

The only part that is not immediate is whether the summations for $x_1, \ldots, x_m$ can be ordered properly and whether the lower and upper bounds for each summation index can be chosen properly so that we can adequately apply the finite geometric series formula. This can in fact always be done, and the ordering and choice of bounds can be done as follows:

Let $x_{i_1}, \ldots, x_{i_j}$ be the entries from the original last pattern p in the length k cluster that coincide with entries from the new last pattern p' in the length k' cluster. In other words, $x_{i_1}, \ldots, x_{i_j}$ are the terms that occur in the y_i's of $C(k', p'; [y_1, \ldots, y_{m'}])$. Let $x_{i_{j+1}}, \ldots, x_{i_m}$ be the terms that were "chopped off" from the length k cluster. Also, assume that $x_{i_1} < \ldots < x_{i_j}$ and $x_{i_{j+1}} < \ldots < x_{i_m}$. Note that in the example in Section 3.1, $x_{i_1} = x_1$ (not "chopped") while $x_{i_2} = x_2$ ("chopped") and $x_{i_3} = x_3$ ("chopped").

Order of summations:
The summations will be ordered (from outermost to innermost) as x_{i_1} to x_{i_j} followed by $x_{i_{j+1}}$ to x_{i_m}. Thus, the outermost summation is indexed by x_{i_1}, the next summation inward is indexed by x_{i_2}, and so on. This places the summations over $x_{i_{j+1}}, \ldots, x_{i_m}$ to be on the "inside" so that they can be moved inward to apply the finite geometric series formula.

Lower/Upper bounds for $x_{i_1}, \ldots, x_{i_m}$:
For each l with $j+1 \leq l \leq m$, let $b_l = k$ if $i_l > i_j$; otherwise, let $b_l = \min(\{i_1, \ldots, i_j\} \backslash \{1, \ldots, i_l\})$, and let c_l be the index of i (so $i_{c_l} = b_l$).

For $x_{i_1}, \ldots, x_{i_j}$:

$$\begin{aligned} x_{i_1} &= i_1 \text{ to } k - m + i_1 \\ x_{i_2} &= x_{i_1} + i_2 - i_1 \text{ to } k - m + i_2 \\ &\cdots \\ x_{i_j} &= x_{i_{j-1}} + i_j - i_{j-1} \text{ to } k - m + i_j \end{aligned}$$

For $x_{i_{j+1}}$:
Lower bound is 1 if $i_{j+1} = 1$, and $x_{i_{j+1}-1} + 1$ otherwise. Upper bound is $k - m + i_{j+1}$ if $b_{j+1} = k$, and $b_{j+1} - c_{j+1} + i_{j+1}$ otherwise.

For x_{i_l} with $l > j+1$:
Lower bound is $x_{i_l - 1} + 1$. Upperbound is $k - m + i_l$ if $b_l = k$, and $b_l - c_l + i_l$ otherwise.

One can see that the indices $x_{i_1}, \ldots, x_{i_j}$ range over all necessary values and can also verify that $x_{i_{j+1}}, \ldots, x_{i_m}$ will cover all necessary values as well. Additionally, for each r, the lower and upper bounds for x_{i_r} never depends on any x_{i_s} where $s > r$. If we applied the above approach to the example in Section 3.1, we would get $x_{i_1} = x_1$ going from 1 to $k-2$, $x_{i_2} = x_2$ going from $x_1 + 1$ to $k-1$, and $x_{i_3} = x_3$ going from $x_2 + 1$ to k.

3.3 Additional Results

We get a couple more immediate byproducts from the algorithm in Section 3.2. First, the method provided for finding a functional equation always works, so we get the following:

THEOREM 3.1 *Let B be a pattern set and $p \in B$. Then, there always exists a functional equation for $F(k, p; [z_1, \ldots, z_{|p|}])$ of the form:*

$$F(k, p; [z_1, \ldots, z_{|p|}]) = (t-1) \sum_{p' \in B} \sum_{i \in I(p')} R_i \cdot F(k_i, p'; [M_1^i, \ldots, M_{|p'|}^i])$$

where $I(p')$ is a finite index set for each $p' \in B$, $I(p')$ and $I(p'')$ are disjoint if $p' \neq p''$, each M_j^i is a specific monomial in $z_1, \ldots, z_{|p|}$, each R_i is a specific rational expression in $z_1, \ldots, z_{|p|}$, and $k_i < k$ for each i.

Additionally, we get an immediate corollary of Theorem 1.1.

COROLLARY 3.2 *Let B be a set of patterns we would like to avoid. Without loss of generality, assume that B contains no redundancies. Then by setting weight$(p) = t - 1$ for each $p \in B$, we get:*

$$P(z,t) = \frac{1}{1 - z - \sum_{k \geq 1} \sum_{p \in B} F(k, p; [1, \ldots, 1]) \frac{z^k}{k!}}.$$

Given that we can find a functional equation given any pattern set B, in a sense, we have an expression for the exponential generating function $P(z,t)$ for any pattern set.

3.4 Maple Implementation

The algorithm from Section 3.2 has also been implemented in the Maple package `CAV`. Using that algorithm, you can find the sequence $\alpha_1, \ldots, \alpha_n$ avoiding a set of patterns B by calling the procedure `CAVT(B,n)`, where the patterns in B are represented as lists. For example, for $n = 10$ and $B = \{123, 321\}$, try `CAVT({[1,2,3],[3,2,1]},10);`. To keep track of the occurrences of patterns from B, use the procedure `CAVTt(B,n,t)`. For example, try `CAVTt({[1,2,3],[3,2,1]},10,t);`. To generate the cluster tail functional equation only (encoded again in a data structure that we call a cluster scheme), use the procedure `MakeTailFE(B,k,z,t)`. For example, try `MakeTailFE({[1,3,2]},k,z,t);`.

Computationally, the algorithm in Section 3.2 is much more efficient than the one in Section 2.1, so the `CAVT` procedure is much faster than the `CAV` procedure. In general, `CAVT` should be used instead of `CAV` for computing α_n values and, similarly, `CAVTt` should be used instead of `CAVt` for $\alpha_n(t)$.

4 Asymptotic Approximations Using CAV

Let $B = \{p\}$ be a set containing a single pattern. In [12], Warlimont gave a conjecture on the asymptotics of α_n:

$$\alpha_n \sim \gamma \cdot \rho^n \cdot n! \tag{17}$$

where γ and ρ are constants depending only on the single pattern p. Some initial asymptotic results for α_n were proven by Elizalde in [5]. Recently, Ehrenborg, Kitaev, and Perry prove this conjecture in [4]. With this result established, we can compute approximate values of γ and ρ for various single patterns.

Elizalde and Noy gave some approximations of γ and ρ for length 3 and a few length 4 patterns in [6]. Aldred, Atkinson, and McCaughan also gave approximations for the ρ values of the single length 4 patterns. Using the Maple package `CAV`, we can empirically verify these approximations and also quickly produce many new approximations. For example, the procedure `AsymApprox(p,N,d)` will give approximate values (up to d decimal digits) for γ and ρ for the pattern p by computing α_{N-2}, α_{N-1}, and α_N and computing their ratios. For example, try `AsymApprox([1,2,4,3],50,20)`.

To approximate γ and ρ values (up to d decimal digits) for all length n patterns and then rank them by the size of ρ, use `AsymApproxRank(n,N,d)`. For example, `AsymApproxRank(4,30,10)` gives us the approximations for the γ and ρ values for length 4 patterns:

Pattern	γ	ρ
1 2 3 4	1.1176930011	0.9630055289
2 4 1 3	1.1375931232	0.9577180134
2 1 4 3	1.1465405299	0.9561742431
1 3 2 4	1.1510444988	0.9558503134
1 4 2 3	1.1567436851	0.9548260509
1 3 4 2 $\sim$ 1 4 3 2	1.1561985648	0.9546118344
1 2 4 3	1.1696577874	0.9528914233

Table 1: Approximate asymptotics for length 4 patterns

Similarly, `AsymApproxRank(5,25,20)` would give us the approximations for the γ and ρ values for length 5 patterns. The output can be found on the paper's website.

5 Further Work

In this paper, we outlined the key procedures in the `CAV` Maple package. The cluster tail generating function was defined, and a constructive approach was demonstrated in finding a functional equation for it. Using this functional equation, we were able to more quickly count permutations avoiding a prescribed set of patterns. In addition, by applying Theorem 2.1, we were able to totally classify c-Wilf-equivalences in single patterns of length 3, 4, and 5 rigorously, while nearly classifying single patterns of length 6. We were also able to classify c-Wilf-equivalences in a few cases of multiple pattern sets. Finally, we were able to use the faster algorithm to compute approximate values for asymptotic

constants.

Despite this, there is a lot of room for improvement algorithmically and quite a few new open problems/conjectures arise. Some of the conjectures are listed below.

Elizalde and Noy provided the following conjecture in [6]:

CONJECTURE 5.1 For a fixed pattern length k, the increasing pattern $\sigma = 12\dots k$ is the "maximal" pattern, in the sense that $\alpha_n(\sigma) \geq \alpha_n(\pi)$ for all $\pi \in S_k$ and all n.

Based off of experimentation, we also have the following analogous conjectures:

CONJECTURE 5.2 For a fixed pattern length k, the pattern $\sigma = 12\dots(k-2)(k)(k-1)$ is the "minimal" pattern, in the sense that $\alpha_n(\pi) \geq \alpha_n(\sigma)$ for all $\pi \in S_k$ and all n.

CONJECTURE 5.3 For a fixed pattern length k, the pattern set $B = \{12\dots k, 23\dots k1\}$ is the "maximal" pattern set among sets of 2 patterns, in the sense that $\alpha_n(B) \geq \alpha_n(B')$ for all $B' \in \binom{S_k}{2}$ and all n.

CONJECTURE 5.4 For a fixed pattern length k, the pattern set $B = \{12\dots(k-2)(k)(k-1), 12\dots(k-3)(k-1)(k)(k-2)\}$ is the "minimal" pattern set among sets of 2 patterns, in the sense that $\alpha_n(B') \geq \alpha_n(B)$ for all $B' \in \binom{S_k}{2}$ and all n.

CONJECTURE 5.5 For a fixed pattern length k, the pattern set $B = \{12\dots k, 23\dots k1, k12\dots(k-1)\}$ is the "maximal" pattern set among sets of 3 patterns, in the sense that $\alpha_n(B) \geq \alpha_n(B')$ for all $B' \in \binom{S_k}{3}$ and all n.

In addition, based off of empirical evidence for single pattern avoidance up to length 6 patterns, we believe the following:

CONJECTURE 5.6 For any two patterns π_1 and π_2 of the same length, either π_1 and π_2 are strongly c-Wilf-equivalent or they are not c-Wilf-equivalent at all.

This certainly holds for single patterns of length 3, 4, and 5. This also appears to hold for single length 6 patterns.

Finally, the author would like to thank Doron Zeilberger for his suggestions, comments, and encouragement towards the work in this paper.

References

[1] R. E. L. ALDRED, M. D. ATKINSON AND D. J. MCCAUGHAN, *Avoiding consecutive patterns in permutations*, Adv. in Appl. Math., 45 (2010) 449–461.

[2] A. BAXTER AND L. PUDWELL, *Enumeration schemes for dashed patterns*, in preparation.

[3] V. Dotsenko and A. Khoroshkin, *Anick-type Resolutions and Consecutive Pattern Avoidance*, arXiv:1002.2761.

[4] R. Ehrenborg, S. Kitaev and P. Perry, *A Spectral Approach to Consecutive Pattern-Avoiding Permutations*, arXiv:1009.2119.

[5] S. Elizalde, *Asymptotic enumeration of permutations avoiding generalized patterns*, Adv. in Appl. Math. 36 (2006) 138–155.

[6] S. Elizalde and M. Noy, *Consecutive Patterns in Permutations*, Adv. in Appl. Math., 30 (2003) 110–125.

[7] I. P. Goulden and D. M. Jackson, *An Inversion Theorem for Cluster Decompositions of Sequences with Distinguished Subsequences*, J. Lond. Math. Soc. (2), 20 (1979) 567–576.

[8] A. Khoroshkin and B. Shapiro, *Using Homological Duality in Consecutive Pattern Avoidance*, arXiv:1009.5308.

[9] J. Liese and J. Remmel, *Generating Functions for Permutations Avoiding a Consecutive Pattern*, Ann. Comb., 14 (2010) 123–141.

[10] A. Mendes and J. Remmel, *Permutations and words counted by consecutive patterns*, Adv. in Appl. Math., 37 (2006) 443–480.

[11] J. Noonan and D. Zeilberger, *The Goulden-Jackson Cluster Method: Extensions, Applications and Implementations*, J. Difference Equ. Appl., 5 (1999) 355–377.

[12] R. Warlimont, *Permutations avoiding consecutive patterns*, Ann. Univ. Sci. Budapest. Sect. Comput., 22 (2003) 373–393.